2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00112

Enhancing Cryptocurrency Blocklisting: A Secure, Trustless, and Effective Realization

Yuefeng Du*†, Anxin Zhou*†, and Cong Wang*†
*City University of Hong Kong, Hong Kong; †City University of Hong Kong Shenzhen Research Institute, China

***Abstract*—The flourishing development of blockchain and cryptocurrency has made it a hotbed for cyber-criminals to implement virtually untraceable scams. Consequently, the blockchain ecosystem urgently needs an effective method to help users stay away from scams in order to create an enticing investment environment. Despite the massive deployment of blocklist query APIs for malicious and scam domains/URLs in the industry, we identify two core reasons why existing blocklist services find it difficult to thrive in the cryptocurrency paradigm: 1) the compelling need to protect a user query due to sensitivity and high value of query content, i.e., payment addresses; 2) the thorny issue of evaluating the quality of blocklists effectively, in the face of common practices of incompetent providers.**

To this end, we first provide a private and highly efficient blocklist query scheme as a basic design, which conveniently achieves backward compatibility with current blockchain payment systems at a considerably low cost. Based on this design, we propose a new framework for shareholders to evaluate the quality of blocklists. Our framework provides stronger security guarantees than other similar works, as it is capable of suppressing both individual biasing and coercive manipulation at the same time. We provide a complete game-theoretic analysis and demonstrate comprehensive evaluation results to confirm the effectiveness and efficiency of our solutions, under the settings of a practical number of shareholders.

I. Introduction

The past decade has witnessed the prolific development of blockchain applications and the value of cryptocurrencies such as Bitcoin [1] and Ethereum [2]. Using cryptocurrencies, users can transfer money without going through trusted financial intermediation. Besides, they also enjoy the benefits of pseudonyms (called payment addresses). However, the decentralized nature of cryptocurrencies also facilitates the implementation of criminal activity, which is difficult for ordinary users to detect because of the pseudonymity of payment addresses. Among cybercrimes, phishing scams [3] and Ponzi schemes [4]–[6] have been found to entice a large base of victims. As analyzed in a recent report [7], cryptocurrency-related crime is primarily made up of scams, whose annual revenue amounts to billions of US dollars, with the annual number of victims exceeding 7 million by the end of 2020.

Large-scale blocklist services have proven helpful to keep users alerted to scams over the years [8], [9], with wide implementation by modern browsers [10]. However, many issues would arise when off-the-shelf blocklist solutions are applied directly to decentralized cryptocurrency payment systems. After reviewing prominent public cryptocurrency blocklists [11], [12], we identify two crucial reasons for the lack of acceptance of cryptocurrency blocklist services available nowadays.

Prevailing economics-driven providers. A typical cryptocurrency blocklist service consists of two parties, namely the users of the service and the moderators on behalf of platform providers that aggregate suspicious payment addresses and provide query APIs. In contrast with domain/URL blocklists, cryptocurrency addresses do not present unique characteristics for analysis, let alone automatic detection of suspicious addresses, which makes the practices of blocklisting indispensable. Needless to say, privacy becomes a major concern of great severity, especially for users with complex *economic* relationships. Specifically, the platform is very much motivated to *monetize* the information of plaintext queries made by large volumes of users, as it is beneficial to infer payment habits and build high-value user profiles [13]. In other words, as user queries for payment addresses are of potential value, addressing privacy concerns is even more beneficial for cryptocurrency users than users of conventional blocklists.

Overlooked incompetence of providers. The other major obstacle to the widespread deployment is the quality of cryptocurrency blocklists, which is more challenging to deal with in reality. As evidenced by a recent work [3], detection of phishing attacks on Ethereum is greatly hampered by the vast data imbalance exhibited in open-sourced cryptocurrency blocklists. Notably, state-of-the-art work like [14] has extensively studied the problem of how to improve the quality of domain/URL blocklists for accuracy increase and broader coverage. Most of the efforts implictly assume trust anchor upon platforms. Specifically, these platforms that provide blocklist query services are heavily trusted for their expertise. But for our scenario, it is such a strong assumption to solely trust a centralized platform for their competence. One might intuitively think: "is it possible to derive an unbiased algorithm to determine which blocklist provider should be trusted and favored more?" We argue instead of this, it is more critical to ask: *How to effectively evaluate the quality of cryptocurrency blocklists in the absence of trusted parties?*

Addressing privacy concerns. We present in this paper our preliminary efforts towards building a blockchain-empowered framework to address the above key challenges and hopefully facilitate the adoption of more effective cryptocurrency blocklists. Our primary focus is to achieve a privacy-preserving yet sufficiently light query design, complementing the existing services of cryptocurrency blocklists with minimal changes. From a technical perspective, at the heart of our query protocol is a hybrid design that seamlessly combines the provable security framework of oblivious computation and the data anonymization guarantees provided by k-anonymity. As a result, our query design can effectively obfuscate user queries while minimizing the use of complex and computationally expensive cryptographic primitives. This serves as a baseline solution for cryptocurrency blocklisting, assuming blocklists are maintained and managed by a centralized and honest party that constantly provides high-quality blocklists.

Decentralized evaluation of quality. The evaluation of cryptocurrency blocklists is a very challenging task, essentially because there is simply no analytical ground truth as well as algorithmic solution. To this end, we allow shareholders selected from all users to play a central role in the collaborative decision making, instead of blindly trusting providers.

Following the convention of decentralized applications, we also rely on economic rationality instantiated with cryptocurrency. Intuitively, interested players have to deposit sufficient cryptocurrencies to become shareholders. At the end of evaluation procedures, the incentive mechanism rewards the shareholders who share the same intention with the final outcome grounded on quorum-based voting, and discourages the others by confiscating portions of deposits. We stress this model has been widely adopted in numerous endeavors [15]–[17] with various focuses. However, these conventional solutions have largely overlooked two disturbing *security implications*: 1) the final outcome could be biased due to asynchronous procedures, as an individual party can submit its own share after inferring others; 2) a group of coercive force could also tamper the integrity of the system by deliberating bribing and colluding in order to manipulate the final result as intended.

Major technical contributions. Inspired by the latest researches [18], [19], we address the former by promoting the secrecy of the system. More specifically, we provide a protection mechanism aimed at the confidentiality of intermediate results throughout the evaluation procedures. Although state-of-the-art constructions of generic frameworks based on *zk-snarks* [20]–[23] are available, we are more concerned with concrete efficiency constraints in this work. To this end, we leverage a highly customized *zero knowledge proof* (ZKP) protocol to attest to the opening of cryptographic commitments for confidentiality protection of submitted shares.

Grounded on in-depth *game-theoretic* analysis, we also provide a new cryptoeconomics framework to suppress coercive behaviors, which to our best knowledge, is an important aspect unaddressed in previous works. As an affordable countermeasure, we first anonymize the identities of shareholders and provide feasible bridging methods based on efficient ZKP to utilize existing private payment frameworks. In addition, we further provide methods to increase the costs of coercing one shareholder by blending all shareholders into a larger pool of candidates with the use of *verifiable random function*.

Our empirical results. As part of our research efforts to understand the practical performance of our designs, we have created system prototypes and conducted extensive experiments. Our basic privacy-preserving design for cryptocurrency blocklist query brings a negligible amount of overhead, with merely a few milliseconds of processing time on the user end. At the same time, the solution also scales well on the server side. As for the decentralized procedures for the evaluation of blocklist quality, each shareholder's computation time is well within 50 ms for a *medium*-size group of 15 representatives selected from users. The monetary cost risen from public verification to effectively evaluate a target blocklist is reasonable as well, i.e., around 16 USD per shareholder as of April 2022.

II. Preliminaries

Oblivious pseudorandom function. A useful protocol [24], [25] to enable two parties, one with an input m and the other with a secret key sk, to jointly compute a pseudorandom function (PRF) $\mathsf{F}(\mathsf{sk},\mathsf{m})$: $\{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^l$. The computation of this PRF is operated obliviously in the sense that the former party only learns the output, whereas the latter party learns nothing from the interaction. Instead of directly applying this protocol, our design uses this primitive as a building block and aims to improve performance in terms of key factors such as communication overhead and throughput.

Homomorphic commitment. A commitment scheme [26] that allows a party to commit a value while hiding the value to others. Aside from the property of hiding, the binding property guarantees the party cannot modify the underlying value after commitment. Consider a simple commitment scheme: $\mathsf{Com}(m) = g^m h^r$, where r is randomness provided by the committed party. The property of homomorphism is desirable in our usage scenario. Specifically, given two cryptographic commitments for two messages m_1 and m_2, we can compute $\mathsf{Com}(m_1 + m_2)$ by $\mathsf{Com}(m_1) \cdot \mathsf{Com}(m_2) = g^{m_1+m_2} h^{r_1+r_2}$.

Zero knowledge proof of knowledge. A protocol [27] that enables a party to convince another party about knowledge of the witness for a computational relation without revealing information of the witness. In particular, we use a non-interactive version (NIZK) [19], [28] to prove a series of commitments are well-formed without disclosing the underlying values. A NIZK protocol consists of three stages: (Setup, Prove, Verify). Our Setup procedures are public in the sense that it outputs a publicly accessible Common Reference String (CRS).

III. System Overview

Our system is meant to suit the enormous demands for privacy-assured and competent blocklist services. The blockchain paradigm comes in handy for the needs of incentives and public auditability. That is, we leverage an open blockchain platform with smart contract [2] to establish trust as well as to re-allocate cryptocurrencies among participants.

A. System architecture

As a starting point, we are after a simplified but also common and useful scenario, where we only consider two parties, namely the users and the cryptocurrency blocklist providers. We then have two basic assumptions: 1) service providers publicize high-quality blocklists freely; 2) users can conveniently find competent blocklist providers. It is fairly reasonable to consider a curious blocklist provider hereby, as it is

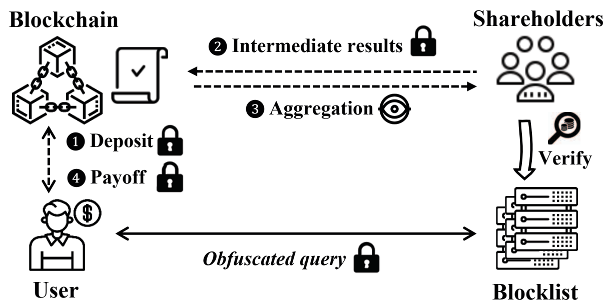

Fig. 1: System architecture with highlighted security guarantees.

incentivized to infer valuable information from collected user queries, e.g., w.r.t. large-amount cryptocurrency transactions.

Hence, as displayed with the solid line in Fig. 1, our basic solution is to protect the confidentiality of user queries. In addition, we also intend to realize a lightweight design with imperceivable user overhead. Next, we consider the validity of the above assumptions. In practice, publicly available blocklists are often far from high-quality blocklists. The alternative way is to trust major platforms that provide paid premium services. Nonetheless, it is still difficult to audit whether these platforms actually provide reliable services as promised.

To this end, we propose a framework to resolve the trust issues and auditability concerns at once. We simply allow committed shareholders to evaluate the blocklist quality and validate the service. The complete system architecture is also depicted in Fig. 1. One notable effort of our design is to anonymize the interaction between the blockchain and users/shareholders, thus increasing the difficulty of manipulative attempts. In addition, the system should also be resistant to manipulation, e.g., bribing, collusion, which is largely overlooked in the literature. We provide a cryptoeconomic analysis of the countermeasures to discourage manipulation from a game-theoretic view.

B. Threat model

We now state our concrete threat assumptions for both our basic query design and the quality evaluation design. Firstly, the blocklist service provider is modeled as an honest but curious party, who is primarily interested in the users' query content, e.g., data reselling for maximized profits. On the other hand, we also consider that some users may try to exhaust the blocklist services, to either learn as much of the blocklist contents as possible or maliciously launch availability attacks.

For decentralized evaluation, we use the blockchain as the underlying platform trusted for integrity and availability but not for confidentiality. Expressly, we assume that the blockchain guarantees that any computation running atop functions exactly as specified. But the computation itself and used data are all publicly visible. In addition, all cryptocurrency payments stored on the blockchain can be publicly traced. We stress that the inherent vulnerabilities of existing blockchain implementation, such as 51% attack against the Proof of Work consensus and eclipse attacks against synchronization of network broadcasting, are out of the scope of this paper.

As demonstrated in Fig. 1, tasks running atop the blockchain are meant to be publicly visible. A group of coercers are fully incentivized to fix the results to maximize their economic interests, thus compromising the integrity of the system. One of our main design goals is to thwart any such attempt and create a robust framework (in the sense of economic rationality) for decentralized evaluation of blocklist quality and validation of blocklist services.

C. Goals

According to the threat assumptions above, we summarize the core security goals we wish to achieve. In addition, we also list the performance and economic goals we strive to achieve in pursuit of the most feasible implementation.

- **Query confidentiality**: The target to be queried by a user is never disclosed to the blocklist service provider, but the user can effectively learn if the target is in the blocklist.
- **Service scalability**: A blocklist service provider should be capable of serving a considerable amount of users simultaneously with relatively low maintenance cost.
- **Evaluation robustness**: The quality of blocklist is evaluated in a fully decentralized manner, yet also immune to undesired tampering to some extent.
- **Minimal costs**: Ordinary users should be able to become shareholders to evaluate a blocklist with minimized monetary cost induced by cryptocurrency deposits.

IV. Basic Design of Secure Blocklist Query

We first consider the unsatisfying demand for effectively blocking scam addresses without the massive user information leakage to service providers. Assuming the existence of curious-but-honest blocklist service providers, we propose a lightweight and scalable solution for interested users to query any blocklist service provider in a privacy-preserving manner.

A. Design rationale

The underlying problem we study here is closely connected to numerous comprehensively studied cryptographic primitives, such as Private Information Retrieval [29] and Private Set Intersection [30]. Despite strong provable security guarantees, these protocols are not suitable for our usage case that requires frequent interaction in practice, as they inevitably introduce massive undesirable overhead.

As a result, it is preferable to have a privacy-preserving blocklist query design that balances the performance goals and security requirements. In terms of concrete efficiency, there are two intuitive guidelines to be met: 1) the interaction between the user and the service provider should be minimized for the scalability of the system; 2) the computation required for hiding user queries from the service provider should be restrained as much as possible for user convenience.

We observe that these requirements can be simultaneously fulfilled by integrating a bucketization scheme [31] with a conventional message hiding protocol based on oblivious pseudorandom function (OPRF). Intuitively, the OPRF serves as a core pillar to enable joint two-party computation obliviously

```
Key procedures and interaction between client $\mathcal{C}$ and server $\mathcal{S}$

Denote an oracle $H : \{0,1\}^* \rightarrow \mathbb{G}$.
$\mathcal{S}$ performs the setup with raw data $\mathbf{D} = \{q_1, \cdots, q_n\}$:
1. $R \leftarrow\$ \mathbb{F}$          // Used for the oblivious evaluation of user queries
2. for $q_i \in D$ do
3.     $b \leftarrow H(q_i)^R$
4.     $p \leftarrow \mathsf{prefix}(H(q_i), \lambda)$          // $\lambda$ is pre-defined prefix bit length
5.     $\mathsf{bucketPartition}(p, b)$          // $2^\lambda$ buckets are partitioned
$\mathcal{C}$ performs the query procedures with query $u$:
1. $r \leftarrow\$ \mathbb{F}$          // Used for blinding the query
2. $m \leftarrow H(u)^r$          // Masked query with the blinding factor
3. $p \leftarrow \mathsf{prefix}(H(u), \lambda)$
$\mathcal{C} \longrightarrow \mathcal{S} : (p, m)$
$\mathcal{S}$ generates the response:
1. $s_p \leftarrow \mathsf{bucketLookup}(p)$          // Results with same prefix
2. $\psi \leftarrow m^R$
$\mathcal{S} \longrightarrow \mathcal{C} : (\psi, s_p)$
$\mathcal{C}$ checks:
1. verdict $\leftarrow \psi^{1/r} \in s_p$ ? TRUE : FALSE
```

Fig. 2: Protocols for highly efficient privacy-preserving membership query.

such that our users only learn the output, whereas the target service provider infers nothing throughout the interaction. Based on that, a bucketization scheme that leverages the divide-and-conquer strategy to deal with the bucketized data pieces instead of the entire repository can greatly boost the performance of the system. By combining these two primitives, our design provides security guarantees of *k-anonymity* [32], where k is determined by the bucket size. Later in Section VI-B, we will show how to flexibly adjust the bucket size for the balance of security and performance in practice.

B. Protocol details

To achieve *k-anonymity* for each of the derived bucket, we assign each of the buckets with data in a way that the access distribution is flattened without any bias. Inspired by the practices in the paradigm of safe browsing [33] based on similar blocklisting mechanisms, we adopt a simple bucketization scheme based on prefix filtering of cryptographic hash results. In other words, all blocklist contents that share the same hash prefix are assigned to the same bucket. As demonstrated in a previous work [34] that defines a formal security framework, the security level of this scheme is affected by two crucial factors, namely the prefix bit length λ and the blocklist size $\tilde{S}$. To enforce *k-anonymity* given the feasible range of k, our composite security framework also relies on the security guarantees of OPRF constructions.

Figure 2 describes the computational procedures for a client and a server, as well as the required interaction between the two parties. We divide our protocol into four primary stages.

1) *Data preprocessing*: Before the target server $\mathcal{S}$ provides the blocklist query service, an initial setup should be completed first to bind the raw data with a mask input k randomly sampled from the finite field $\mathbb{F}$ by $\mathcal{S}$. In addition, $\mathcal{S}$ also needs to partition the blinded data into buckets in accord with the required prefix bit length λ. $\mathcal{S}$ can run this protocol in rotation whenever there is a demand for adjusting k.
2) *Secure query*: User $\mathcal{C}$ initiates a valid query by hiding its query content with a random coefficient r uniformly sampled from the same field $\mathbb{F}$. $\mathcal{C}$ then attaches the plaintext of hash prefix (w.r.t λ) of the query content. Notice a first-time user should synchronize on the value of λ with the server in order to get the latest bucket size. Optional steps: When $\mathcal{S}$ agrees to reveal the knowledge of the prefix list of the raw data, most of the queries requested by $\mathcal{C}$ can be completed locally by checking the stored prefix list distributed by $\mathcal{S}$. Specifically, $\mathcal{C}$ looks up the same hash prefix in the small-sized prefix list. If no match exists, it implies the query content is not in the blocklist; otherwise, further interaction is required.
3) *Online evaluation*: To process the user query, $\mathcal{S}$ returns all the results within the same bucket, denoted as s_p and blinds the user query with the same mask input R. Note that within the lifecycle of unaltered R, the results s_p can be locally cached to circumvent online interaction when consequent queries preserve the same prefix.
4) *Response recover*: Lastly, user $\mathcal{C}$ needs to recover the data with its stored coefficient r, such that the result can be used to determine if the query content matches with the returned results s_p.

Security guarantees. Assuming the Diffie-Hellman assumption, the obliviousness property of the underlying OPRF primitive plus the hash method we use enables us to prove in the random oracle model that the service provider learns nothing of the user query throughout the interaction. Likewise, the pseudorandomness property helps to achieve bounded leakage for blocklist contents not queried by the user.

Support for metadata query. Our protocol currently does not include metadata information such as the categories and reporting time. In practice, we believe this is the best strategy as it is hard to unify the structure of different blocklists. Besides, it would also be more challenging to evaluate the quality of a target blocklist. Nonetheless, our protocol can be extended to include any type of metadata, by leveraging the primitive of Private Keyword Search [35].

Remarks on denial of service (DoS) attacks. A common defense strategy is to adopt rate limiting through the mechanism of authorized keys. In addition, in our design, we ensure that server responses should not require a significant amount of computation compared to requests made by clients (including bogus requests). This is achieved with an inefficient oracle H.

V. Enhancement with Decentralized Evaluation

The basic design we have illustrated achieves backward compatibility with existing blocklist service providers at a low cost. However, the implicit assumption that reliable service providers are competent does not naturally hold in our motivating scenarios. How to distinguish high-quality services is often a crucial problem, as centralized platforms like Google [10] and Phishtank [36], which act as trust anchors, are missing in the decentralized paradigm. Hereby, we start with a conventional decentralized framework based on *commit-vote-payoff* procedures, where ordinary peers become valid shareholder

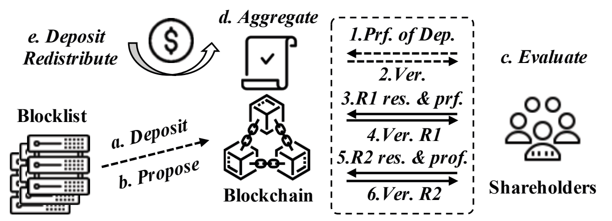

Fig. 3: Publicly committed workflow of decentralized evaluation.

voters and a list of "evaluated" blocklists are generated and the corresponding services are also periodically validated.

A. Commit reveal schemes

Since the earliest design of partial lock commit reveals protocol [16] implemented on Ethereum [2], many decentralized applications have attempted to leverage the commit-and-reveal approach to achieve consensus in a decentralized yet efficient manner. Closely related to the inherent incentive mechanism of blockchain, these schemes are typically built with assumptions that all players are strategic (fully rational).

One widely adopted instantiation in recent industrial practices is the token curated registries [15], [37]. It proves useful in settings where the ground truth is not available or cannot be easily discovered. Concretely, consider a service of unknown quality Q, and for simplicity, with an idealized binary choice $\hat{Q}$ indicating whether the service is reliable among the consensus of n shareholders. In our system, n is a small pre-determined number agreed upon among blocklist users. Each shareholder voter has a vote v_i as well as a corresponding weight τ_i. As a result, $\hat{Q}$ can be derived as follows for a conventional quorum-based vote setting:

$$\hat{Q} = \begin{cases} 1, & \sum_{i=0}^{n-1} \tau_i v_i > \frac{1}{2}\sum_{i=0}^{n-1} \tau_i \\ 0, & \sum_{i=0}^{n-1} \tau_i v_i \leq \frac{1}{2}\sum_{i=0}^{n-1} \tau_i \end{cases} \tag{1}$$

Consequently, a deterministic payoff function calculates the rewards/penalties associated with each shareholder's voting shares τ_i, after comparing the outcome and the vote $\langle \hat{Q}, v_i \rangle$. Note that the binary result derived among shareholders is directly applicable in our case, as these shareholders could represent the blocklist preference of all users.

In spite of its commonly acknowledged utilities in a decentralized context, numerous security vulnerabilities have been discovered [38]–[40]. One fundamental problem summarized from these attacks is that internal procedure followers and external adversaries are incentivized to abuse the system and influence the final outcome.

B. Overview of our workflow

To help users curate high-quality blocklist services, we adopt a design akin to the above-mentioned commit-and-reveal approach. Likewise, our evaluation procedures also involve two rounds of interaction for "pre-commitment" and "post-aggregation", but with enhanced overall security to discourage coercion in our game-theoretic framework.

Difference from prior works. The literature [41] of secure voting in centralized settings mostly considers these two sides by formalizing a property known as *perfect ballot secrecy*. Combining anonymization techniques with publicly verifiable results, one latest work [19] claims design based on this principle could be used to fairly derive evaluation results among shareholders. However, we observe that in our decentralzied context where sybil identities could be a big problem, coercers can hind behind anonymized identities and commit resources to become shareholders themselves. This viewpoint of coercers is largely overlooked and unaddressed in prior works.

To this end, we consider our security countermeasures from the perspectives of both sides: 1) implement an effective protection mechanism with affordable costs; 2) reduce the chance and thus the incentive for coercive attempts. The former aspect is achieved by only publicizing auditing trails for verification whereas anonymizing intermediate results, demonstrated in Fig. 3. This largely prevents individual parties from inferring the final outcome and attempting to bias it. To mitigate coercion, we go to great lengths to increase the cost of coercing one shareholder voter.

Workflow walkthrough. As demonstrated in Fig. 3, any user can join the decentralized committee as a shareholder voter for quality approval of the target blocklist service provider, as long as a deposit is locked on the blockchain. On the other hand, a blocklist service provider can propose for evaluation after deposits, too. From the perspective of security, we decouple the vulnerabilities of payment procedures and the evaluation procedures in our design such that we can employ existing building blocks proven to be secure.

To obtain unbiased and publicly verifiable evaluation results, our security mechanism needs to fulfill these core security requirements: 1) the final outcome can be correctly aggregated without any trusted third parties; 2) it is publicly verifiable that all shareholder voters faithfully follow the computation procedures; 3) any shareholder's vote is only recoverable by the coalition of the remaining players, in order to rigorously discourage the revealing of intermediate voting shares and thus prevent potential collusion and subversion behaviors.

On the other hand, to discourage incentives-driven coercion, we rely on the following observations: 1) it is indispensable to anonymize the identities of shareholders throughout the workflow; 2) it is also of great necessity to hide transaction volumes and transaction addresses while paying off the shareholders; 3) To prevent disguise as a shareholder, we have to obfuscate shareholders among a large number of candidates such that it is costly for coercers to coerce one shareholder.

Technically, we leverage the primitives of homomorphic cryptographic, non-interactive zero knowledge proof (NIZK), and verifiable random function. We stress that assembling building blocks based on these primitives into a design with affordable costs is non-trivial work.

C. Procedures of "evaluation"

Akin to the basic commit-and-reveal scheme, our procedures consist of two rounds of interaction with the blockchain.

```
Registration:

Setup $(thresh, N)$        // Define a threshold number for open registration
    set $cnt := 0$;

VoteCommit $(\pi_{deposit}, pk)$ from $\mathcal{M}$
    assert $\mathsf{NIZK}_{\mathsf{verify}}(\pi_{deposit}, \phi_{pub}) = 1$;
    receive $(comm_{secret}, comm_{vote}, \pi_A)$ from $\mathcal{M}_i$
    assert $\mathsf{NIZK}_{\mathsf{verify}}(\pi_A, \phi_A, comm_{secret}, comm_{vote}) = 1$;
    $cnt := cnt + 1$; broadcast "intention fixed";

On receive signal $(cnt = thresh)$, output a random number $\nu$
    On receive $(y_i, prf_i)$, check $\mathsf{VRF.Verify}_{pk}(\nu, y_i, prf_i) \overset{?}{=} true$
        set $selected.\mathsf{insert}(i)$; otherwise set $unselected.\mathsf{insert}(i)$
    unlock $\$deposit$, $\forall i \in unselected$; broadcast "voters fixed";
```

```
Auto-tally:

Setup $(\{comm_i\}_{i=0}^{N-1})$
    set $V := 1$; $cnt := 0$; broadcast "setup completed";

Vote $(comm_{secret}, comm_{vote}, \psi, \pi_B)$ from $\mathcal{M}_i$
    assert $\mathsf{NIZK}_{\mathsf{verify}}(\pi_B, \phi_B, comm_{secret}, comm_{vote}, \psi) = 1$;
    set $V := V \cdot \psi$; $cnt := cnt + 1$;
    broadcast "vote fixed";

On receive signal $(cnt = N)$
    set $tally := \mathsf{solveDLP}(g, V)$ // Max value for brute force search is $N$
    if $tally > N/2$:    set $outcome := 1$;
    else :               set $outcome := 0$;
    call payoff$(outcome, comm_i)$; broadcast "outcome released";
```

Fig. 4: The blockchain-related activities in our secure decentralized framework that "evaluates" high-quality blocklist services.

The underlying reason is that we need each shareholder voter to commit its vote during the first round, which can be aggregated in the second round. Figure 4 displays the concrete on-chain procedures of the two phases, namely the registration and auto-tally phases. Below we elaborate on procedure details of both on/off-chain operations. For ease of presentation, we omit the voting weight τ in our presented procedures.

Registration phase: Initially, all parties need to agree on a protocol to generate common reference string (CRS), which includes public materials such as group generators g, h for commitment. Upon a blocklist service proposal, the blockchain is used to record all necessary information. It first initiates two constant numbers $thresh$ and N, indicating the number of stakeholders it allows to deposit and the number it will select by the end of the registration, respectively.

Next, any interested player generates a key pair (sk, pk) for verfiable random function [42] later and sends pk to the blockchain during the deposits registration through private transactions. With verifiable deposit proof asserted on the blockchain, the shareholder is allowed to send a commitment to the intended vote. Let v be a binary voting option, and x be a secret only known to the shareholder. The commitment can be obtained as shown below:

$$comm_{secret} = g^x, \quad comm_{vote} = g^v h^x \tag{2}$$

After the shareholder submits the above commitment as well as NIZK proof, the blockchain can publicly verify the correctness of computation and count the shareholder in if the verification passes. For all shareholders that have deposited and passed verification, the blockchain generates a challenge ν for each player to compute a result y through $\mathsf{VRF.Eval}_{\mathsf{sk}}(\nu)$ attested with a proof prf through $\mathsf{VRF.Prove}_{\mathsf{sk}}(\nu)$ to determine whether it is granted with the voting privilege. The shareholder members of the decentralized committee are hence fixed after the blockchain checks y and prf.

Auto-tally phase: For all selected shareholder voters, they also have to complete the procedures of the second-round vote, in order for their votes to be incorporated into the final result. More concretely, a target shareholder voter p first assembles all the commitments provided by others in the following manner:

$$Y = \prod_{i=0}^{p-1} comm_{i,secret} / \prod_{i=p+1}^{N-1} comm_{i,secret} \tag{3}$$

Then, it also aggregates its vote with a commitment $\psi = g^v Y^x$. Intuitively, each individually aggregated result can be used during the on-chain auto-tally procedures to cancel out randomness (to hide votes) provided by each shareholder. This is a very useful technique [41] formally proved secure in the context of voting without trusted third parties. At the end of this step, the shareholder voter is also required to attest to the correctness of computation with NIZK proof.

The on-chain procedures begin to aggregate the intermediate results after all the shareholder voters have submitted the required materials. Otherwise, the voting procedures would be deemed unsuccessful and the deposited tokens will be redistributed. Assuming everything runs smoothly, the blockchain simply multiplies the results computed by each shareholder voter step by step and eventually derives a final outcome V. Now the blockchain needs to run a brute force search algorithm to find out the underlying solution to a discrete logarithm problem. Despite its asymptotic difficulty, the solution can be efficiently discovered in practice, as the domain of N is constrained to a small scale for feasible consensus building. This publicly found solution is equal to the number of shareholder voters in support of the target blocklist service.

Bridging secure payoff. As we have mentioned earlier, it requires a compatible design for secure payoffs to match with the final outcome released by the blockchain, without disclosing the votes of each individual shareholder. Essentially, the deposit commitment of the corresponding shareholder should be updated accordingly by multiplying with a new "helper" commitment derived from the *vote commitment* as well as a binary result calculated from the comparison between the *individual vote* and the *final outcome*. A useful trick here to form the "helper" commitment from the comparison result without revealing *individual vote* is to multiply the arithmetized expression of the boolean equality operation, i.e., $b_1 = b_2 \Leftrightarrow 1 - b_1 - b_2 + 2b_1b_2$. To put all pieces together, consider a deposit commitment $g^{amt}h^{secret'}$, if a shareholder wins the vote and is awarded one unit token, the updated deposit commitment would become $g^{amt+1}h^{secret+secret'}$.

Verifiable blocklist service. After the mutual decision among the shareholders, the blocklist service provider obtains the evaluation result attested by the corresponding publicly verifiable proof. However, the provider could still alter (portions of) the entries in the blocklist, but this is out of our scope as it is not incentivized to do so. The more likely situation would be that the provider fails to clear up obsolete entries and sort out valid blocklist entries.

In light of this, we specify that the blocklist service provider has to repeat the above procedures periodically, according to the pre-determined rules recorded on the blockchain after the proposal. Additionally, we also allow any off-chain party to challenge the blocklist service provider at any time, whose required deposits should be no less than the blocklist service provider. Note that shareholders need to evaluate the following instead of merely blocklist quality: 1) verify that blocklist entries are indeed included in the blocklist through random membership inference. 2) verify that the prefixes and blocklist entries are correctly mapped.

Security reduction. The desired quality "evaluation" procedures rely on the correctness of auto-tally procedures, and more specifically, the homomorphic property of our used commitment. In fact, the following equation holds, as Y provided by each shareholder in Equation (3) cancels out:

$$V = \prod_{i=0}^{N-1} g^{v_i} Y_i^{x_i} = g^{\sum_{i=0}^{N-1} v_i}$$

The entire design also hinges upon the security guarantees of the primitives we have leveraged. Concretely, with the standard Decisional Diffie-Hellman (DDH) assumption, our design can straightforwardly fit in with the security framework elaborated in [19]. Specifically, in order to support our feasible construction of NIZK (explained below), we need to place more public parameters in the CRS model, which can be instantiated in practice with distributed setup procedures. With CRS, we can prove the security of our design in the non-programmable random oracle model (ROM).

D. Feasible instantiation of NIZK

Both rounds of the voting require the shareholder to obtain valid NIZK proof to ensure the correctness of computation and thus circumvent any potential dispute. For the algebra-friendly commitment schemes used in our design, we can obtain efficient NIZK constructions based on the sigma protocols and the Fiat-Shamir heuristic [28]. The basic idea is to employ the OR composition techniques [19] to show either the NIZK statement is valid or the CRS contains a DDH tuple.

Figure 5 provides the detailed steps for a shareholder voter to provide a NIZK proof attesting to the computed results during the first round. Concretely, the public CRS should be set up beforehand. It consists of six group generators, where h_1 and h_2 are indispensably required to rigorously prove the security. We also need a random oracle to make the zero knowledge proof non-interactive.

As demonstrated earlier, $\mathcal{M}$ first determines its voting option v, generates a random secret x and corresponding commitment. For ease of presentation, we use c_0 and C to represent $comm_{secret}$ and $comm_{vote}$ in Equation (2), respectively. In addition, $\mathcal{M}$ also needs to generate other two commitments c_1, c_2 from x, which are used to form a DDH triple $(g^\alpha, h_1^\alpha, h_2^\alpha)$ for the consequent NIZK computation.

```
*Computation performed by shareholder M and the blockchain B*
Let $(g, h, h_1, h_2, \hat{g}, \hat{h}) \in \mathbb{G}^6$ be public parameters.
Let $R : \{0,1\}^* \to \mathbb{F}$ be a random oracle.
$\mathcal{M}$ computes:
  1. $x \leftarrow\$ \mathbb{F}$                                  // $x$ is the voter's secret
  2. $v \leftarrow \mathsf{setVote}()$, where $v \in \{0,1\}$        // $v$ is the voter's vote
  3. $(c_0, c_1, c_2, C) \leftarrow (g^x, h_1^x, h_2^x, g^v h^x)$
   // NIZK proof $\pi_A$ for relation $\phi_A((c_0, c_1, c_2), x) = 1$
  4. $\alpha, \beta_0, \beta_1 \leftarrow\$ \mathbb{F}$
  5. $\sigma_0, \sigma_1, \sigma_2 \leftarrow g^\alpha, h_1^\alpha, h_2^\alpha,\ \ \gamma_0, \gamma_1 \leftarrow \hat{g}^{\beta_0} g^{\beta_1}, \hat{h}^{\beta_0} h^{\beta_1}$
  6. $\mu \leftarrow R(c_0, c_1, c_2, \sigma_0, \sigma_1, \sigma_2, \gamma_0, \gamma_1)$
  7. $a \leftarrow -\beta_0,\ \ b \leftarrow \beta_1,\ \ \omega \leftarrow \alpha + (\mu + a)x$
$\mathcal{M} \longrightarrow \mathcal{B} : (c_0, c_1, c_2, C, \pi_A = (\sigma_0, \sigma_1, \sigma_2, \gamma_0, \gamma_1, a, b, \omega))$
$\mathcal{B}$ computes $\mu$ from $R$ and verifies $\pi_A$:
  1. $\mu \leftarrow R(c_0, c_1, c_2, \sigma_0, \sigma_1, \sigma_2, \gamma_0, \gamma_1)$
  2. $b_0 \leftarrow \sigma_0 c_0^{\mu+a} \overset{?}{=} g^\omega$
  3. $b_1 \leftarrow \sigma_1 c_1^{\mu+a} \overset{?}{=} h_1^\omega,\ \ b_2 \leftarrow \sigma_2 c_2^{\mu+a} \overset{?}{=} h_2^\omega$
  4. $b_3 \leftarrow \gamma_0 \hat{g}^a \overset{?}{=} g^b,\ \ b_4 \leftarrow \gamma_1 \hat{h}^a \overset{?}{=} h^b$
  5. return $b_0 \wedge b_1 \wedge b_2 \wedge b_3 \wedge b_4$
```

Fig. 5: Detailed computation procedures for the first round of interaction.

Concretely, the NIZK protocol is used to prove the obtained results c_0, c_1, c_2 are all well-formed in the sense they are derived from the same secret x. Notice we do not need to prove C is also well-formed at this stage, as it is not used by on-chain operations and we can leave it to the next phase. Formally, it is equivalent to prove the following relation:

$$\phi_A = \{((c_0, c_1, c_2), x) : c_0 = g^x \wedge c_1 = h_1^x \wedge c_2 = h_2^x\}$$

In order for $\mathcal{M}$ to generate a NIZK proof, three random values are first sampled. Then, $\mathcal{M}$ obtains a challenge μ by querying the random oracle. Later, the on-chain verification also relies on this same query. By setting up the coefficients a, b, ω in accord with μ, the proof π_A is now complete.

Similarly, by leveraging the OR composition technique, it is not hard to prove the following relation in the second round:

$$\phi_B = \{((c_0, C, \psi, Y), (x, v) : c_0 = g^x \wedge C = g^v h^x \wedge \psi = g^v Y^x)\}$$

Notice the value Y calculated in Equation (3) is regarded as public input because the blockchain also needs to re-compute Y using intermediate results provided by all shareholders in the first round. Due to the space limit, we omit the computation details to obtain a NIZK proof π_B for ϕ_B.

E. Game-theoretic analysis

Before we provide our design, we propose a game-theoretic framework to generalize the above attacks. The main intuition is that all the abusers (internally and externally) can be viewed as a group of coercers maximizing their profits. Hence, the basic idea is to model the decentralized evaluation procedures as a simple strategic game between the 'good' society representing public interests and coercer(s) in contrast. Specifically, we model such evaluation as a strategic game

TABLE I: Computational and communication overhead introduced by our design with k anonymity security guarantees.

Prefix len.	Sec. wrt. k	Resp. size (kB)	Orac. H	Preprocess time[†]	Qry. time (ms)	Eval. time (μs)	Rec. time (μs)
3 bit	4	0.13	Sha256	1.55 ± 0.02 *sec.*	$0.38 \pm 5 \times 10^{-3}$	41.33 ± 0.08	391.98 ± 0.93
4 bit	977	30.53	Argon2[*]	1.27 ± 0.03 *hour*	147.29 ± 4.26		

[*] To deliberately slow down user queries, we use sequential Argon2id with memory set to 4 MB, time cost (working factor) set to 3.
[†] This operation is evaluated with parallel computing (8 cores fully utilized) to simulate actual usage scenarios during server deployment.

$\langle \mathcal{M}, \mathcal{A}, \sum, (U_{\mathcal{M}}, U_{\mathcal{A}}) \rangle$, where $\mathcal{M}$ represents the society, $\mathcal{A}$ represents the coercers, $\sum$ is the set of strategy profiles, and $U_{\mathcal{M}}$ and $U_{\mathcal{A}}$ are utility functions of the corresponding player. Note that cryptocurrencies can be regarded as the payoffs of the above-defined utility functions. The strategy $\sum = \sum_{\mathcal{M}} \times \sum_{\mathcal{A}}$ comes with the ingredients defined below.

Strategies. $\mathcal{M}$'s strategy is to prevent bribing and colluding among shareholders. Denote a list of protection methods that could be adopted by $\mathcal{M}$ as $\sum_{\mathcal{M}} = \{\psi_0, \cdots, \psi_{max}\}$. In contrast, $\mathcal{A}$'s strategy is to coerce shareholders so as to alter the evaluation results. The strategy is thus given as $\sum_{\mathcal{A}} = \{0, \cdots, k^*, \cdots, k\}$, where the minimal number of shareholders required for the change the result of evaluation in favor of coercers is k^*, which we also refer to as $thresh$. Note we assume the value of k^* is public knowledge deducible from public information recorded on the blockchain.

Preferences. Because in our considered scenario, the strategies of $\mathcal{M}$ are pre-determined and recorded on the blockchain, we can simply assume only pure strategies are played, i.e., both parties will not randomize their strategy play. With that said, the preferences of these pure strategies can be represented by utility functions. Denote the utility of the society $\mathcal{M}$ as $U_{\mathcal{M}}(\psi_j, n) = V_{\mathcal{M}}(\mathsf{Oracle}(\psi_j, n)) - C_{\mathcal{M}}(\psi_j)$, where $U_{\mathcal{M}}$ is affected by both the protection mechanism ψ_j and the number of shareholders n that $\mathcal{A}$ attempts to coerce. This is because $U_{\mathcal{M}}$ is primarily comprised of the social value of the evaluation results $V_{\mathcal{M}}$, which naturally stems from Oracle, outputting the outcome of the decentralized evaluation result. As we also assume a binary result in Equation 1, we can denote the social value to $\mathcal{M}$ as $c_{\mathcal{M}}$, when the result is fairly derived. In contrast, the social value is $c_{\mathcal{M}} - \epsilon_{\mathcal{M}}$. Aside from social value, the utility should also deduct the implementation cost of the protection methods ψ, which we denote as $C_{\mathcal{M}}(\cdot)$.

Similarly, the utility function of the coercer $\mathcal{A}$ is then $U_{\mathcal{A}}(\psi_j, n) = V_{\mathcal{A}}(\mathsf{Oracle}(\psi_j, n)) - n \cdot C_{\mathcal{A}}(\psi_j)$. The social value to $\mathcal{A}$ is different from $\mathcal{M}$, which we use the function $V_{\mathcal{A}}$ to measure. Consequently, we denote the value to $\mathcal{A}$ as $c_{\mathcal{A}}$, given the favoured result; and $c_{\mathcal{A}} - \epsilon_{\mathcal{A}}$ otherwise. Note that this time the coercion costs grow with the number of shareholders that should be controlled by $\mathcal{A}$, as we neglect the constant cost to overcome the protection method ψ.

Implications. Before drawing a conclusion, we share some useful insights from the perspective of $\mathcal{A}$. Firstly, as $\mathcal{M}$ progressively changes the protection method from ψ_0 to ψ_{max}, it is desirable that it becomes much more costly for $\mathcal{A}$ to coerce the same number of shareholders. Namely, $C_{\mathcal{M}}(\psi_j)$ should be far smaller than $n \cdot C_{\mathcal{A}}(\psi_j)$ with a suitable value of j. We practice this by *anonymizing* the shareholders and intermediate results with cheap on-chain verification operations. Secondly, it is straightforward to see $\mathcal{A}$ has two undominated strategies, namely 0 and k^*, depending on how $\mathcal{M}$ may play. In other words, it only makes sense for rational $\mathcal{A}$ to coerce none or k^* shareholders. Formally, for a given ψ, if $c_{\mathcal{A}} - C_{\mathcal{A}}(\psi) * k^*$ is smaller than $c_{\mathcal{A}} - \epsilon_{\mathcal{A}}$, then $\mathcal{A}$ is incentivized not to coerce. Hence, we have incorporated *randomized* yet publicly *verifiable* shareholder selection procedures from a sufficiently large pool of candidates to increase k^* required for $\mathcal{A}$ to successfully coerce. Interestingly, the undominated strategy of coercing k^* shareholders can be seen as Stackelberg equilibrium, assuming the "leader" $\mathcal{M}$ chooses its strategy in advance and commits to it. In our application scenario, the commitment is partially guaranteed with the *on-chain* workflows that are publicly verifiable, under the majority good assumption.

VI. Implementation and Evaluation

A. Prototype implementation

We develop a core module for ordinary users to securely query whether a blockchain address resides within a target blocklist. The OPRF protocol described in Section IV-B is efficiently instantiated with groups over elliptic curves [43]. Due to Rust's near-native performance, rich security mechanisms and cross-compilation abilities, we implement the OPRF protocol with a Rust library [44] that operates upon the prime-order Ristretto group [45] over the curve 25519. We also rely on the slow hash function Argon2 [46] to instantiate an inefficient oracle to defend DoS attacks. This module can be directly deployed over any existing blockchain transaction system with open-sourced blocklists query services deployed.

To facilitate the use of high-quality blocklists, we simulate the API for actions with regard to quality evaluation through a decentralized committee. Conveniently, we use the same Rust library [44] to implement the commitment scheme and zero knowledge proofs depicted in Section V-C.

Dataset. In order to simulate the real-world deployment settings, it is crucial to understand how the scale of blocklists affects our design. In light of this, we collect public datasets for widely used cryptocurrency platforms, including Bitcoin, Ethereum, and Ripple. Notably, most of the entries are from Bitcoin Abuse Database Index [11] and CryptoScamDB [12]. After duplication, we derive around $243,000$ unique entries.

Experiment setup. With respect to the above prototypes, we evaluate the performance as well as the overall cost, and obtain average results out of 100 measurements. We set the size of the finite field $\mathbb{F}$ to 256 bit and use the GMP library [47] to implement big integer for efficiency. This will suit the Ristretto group we use, where a point on the curve 25519 yields 32 bytes after compression. As a result, a group element in both the commitment and the NIZK protocol is 32 bytes.

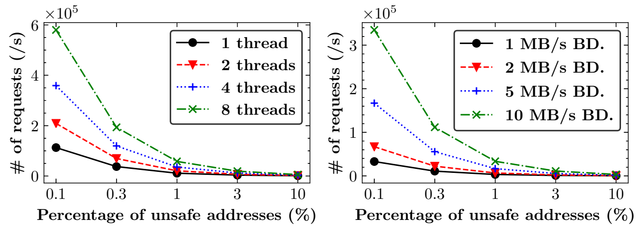

Fig. 6: Max concurrent requests allowed under various percentages. *Left:* CPU usage is a major constraint with a 3-bit prefix length. *Right:* Bandwidth is a more crucial factor with stronger security guarantees and 4-bit prefix length.

To simulate a real server, the deployment environment uses an eight-core Intel E-2174G (3.8 GHz) processor with 32 GB RAM. Compared to the desktop, the mobile end is more often a bottleneck. Hence, we carefully installed the application natively compiled and linked (Android NDK toolchains) from Rust, on the latest aarch64-linux-android platform with quad-core 2.35 GHz CPU and 6 GB RAM. Lastly, to enable efficient on-chain operations, we compile from reusable Rust modules to WebAssembly [48] binary. For its cross-platform interoperability, we deploy it to our private testnet with the support of Turing-complete Ethereum smart contract [49].

B. Evaluation on secure blocklist query

Efficient protection on users. Our evaluation results confirm the efficiency of our secure blocklist query protocol elaborated in Section IV-B. We hence prepare Table I to demonstrate the detailed communication and computational overhead under various useful settings. We first show two common security settings. For a 3-bit prefix match setting, a user query can be hidden among four indistinguishable queries, as guaranteed by $k = 4$. Because the response size accounts for most of the communication overhead, we only display its concrete figure in the table. We can see the communication overhead for this setting is almost negligible. We stress this figure grows linearly with the value of k. As a result, for a stronger security setting with a 4-bit prefix, the response size expands significantly.

Apart from query privacy, our design is also able to defend against DoS attacks, which commonly target valuable query services, with a bearable amount of preprocessing overhead. In Table I, we also show a full comparison between a normal oracle H and an inefficient oracle initialized with a customized slow hash function. Most of the perceivable overhead attributes to the blocklist preprocessing procedures conducted on the server end. Despite hours of computational efforts, it can be configured flexibly with the underlying slow function. Whereas on the user end, only the query time is deliberately slowed down for DoS defense. Notably, operations regarding ‘oblivious computation’ introduce little overhead even on the mobile end (at the magnitude of milliseconds).

Scalable service providers. Our lightweight design also enables a service provider to serve a large number of users at a relatively low cost. In our experiments, we test the number of concurrent requests allowed with a single server and further estimate the scalability of our design. Specifically, the test is conducted using the Jmeter [50] toolset. By distributing and

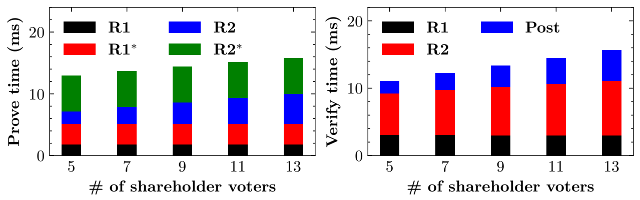

Fig. 7: Computational overhead w.r.t. number of voters. *Left:* Proving time for native operations of two rounds (R1, R2), and NIZK operations (R1*, R2*). *Right:* Verification time for two rounds and the post-aggregation procedures.

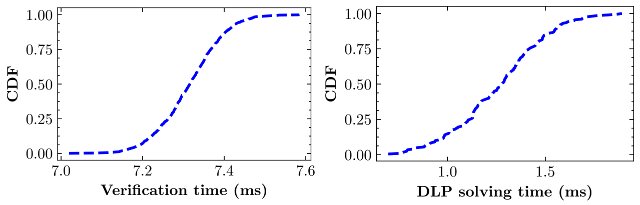

Fig. 8: Varying on-chain operation time. *Left:* Verification time for different shareholders due to computation of Y. *Right:* Recovery time of vote consensus by solving a small-exponent elliptic curve discrete logarithm problem.

synchronizing a prefix list from time to time, as we suggest in the design, a service provider can lift the vast majority of online interaction, thus greatly improving service scalability.

The remaining question thus becomes: *what's the percentage of the queries that eventually require online interaction*? As the answer depends on the quality and coverage of a given blocklist, we estimate it with five sets of candidate settings in Fig. 6. The center point is set to 1%, which is in accord with the ratio of the size of our collected blocklist to all existing Bitcoin addresses. Our general observation is that aside from the factor of computation bottleneck, the throughput is also affected by the server bandwidth. Notably, bandwidth is a more crucial constraint when the prefix length bit is set to four.

C. Performance of decentralized quality evaluation

We now measure computational and communication overhead incurred by homomorphic commitments and NIZK for quality evaluation. Based on that, we proceed to estimate concrete costs brought by blockchain-enabled verification.

Protocol efficiency. To understand what overhead a shareholder needs to take during the participation of the voting procedures, we estimate the concrete proving time for both two rounds on the mobile platform. On the other hand, the verification procedures are carried out in the aforementioned server environment. The results are presented in Fig. 7, where we vary the range of the voter number N and break down the detailed computational cost in terms of its serving purposes.

As shown, the NIZK proof generation procedures, which is used to attest to the correctness of computation, accounts for the majority of the overhead. In addition, as both the prover and the verifier need to compute Y to aggregate others' opinion in the second round, the required time grows linearly with the number of N. The other operation that also increases with the growth of N is the post-aggregation procedures after each of the proofs submitted by shareholder voters is verified.

Aside from the above averaged results, it is also noteworthy

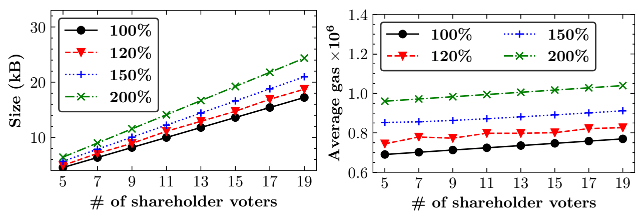

Fig. 9: On-chain cost grows with the number of voters. Label percentage is the ratio of $thresh$ over N (num. of voters) in the first round. *Left:* Compulsory proof size stored on blockchain. *Right:* Converted Ethereum gas cost.

TABLE II: Estimated on-chain cost undertaken by each shareholder

# of shareholder voters	5	7	9	11
Cost (USD)	16.02	16.28	16.54	16.80

that the verification time would also change, given N is fixed beforehand. This is because the computation of Y requires different runs of modulo inverse operations. The same also works for the time of solving a DLP problem to obtain the vote result during the post-aggregation procedures. In light of this, we plot Fig. 9 to demonstrate cumulative distribution function (CDF) for these two scenarios.

Decentralization costs. To accurately model the explicit cost of decentralized verification enabled by blockchain, we hereby analyze the detailed on-chain costs using the general-purpose Ethereum platform. We stress the costs could be further reduced in deployment through off-chain state channel designs but we regard them as orthogonal efforts. Our on-chain procedures implicitly require each shareholder to pay for both storage and computational costs. The primary portion of the cost stems from on-chain storage needs for submitted proofs. In the left subfigure of Fig. 9, we demonstrate the total amount of storage proofs to be stored on the blockchain. The growth of this total size is affected by both $thresh$ and N.

In addition to storage cost, we also estimate the amount of computation cost by converting the above-obtained verification time to a comparable baseline. Specifically, we apply the assumption that 1 gas equals 0.1 μs of CPU execution on pre-determined hardware, as suggested by the Ethereum WebAssembly proposal [49]. Because all the verification operations can be executed within 100 ms, they can easily be attached to a single block without any additional overhead. To aggregate the storage cost and the estimated computation cost, we obtain the right subfigure of Fig. 9.

Finally, according to Ethereum gas calculator [51] (using 11.8 Gwei gas price), we estimate the total explicit cost for each shareholder to complete full procedures of decentralized evaluation upon a proposal requested by a single blocklist service provider. The results are demonstrated in Table II. As a comparison, the average transaction amount once reached \$70 in early 2022, which makes the overall cost each shareholder needs to undertake quite reasonable.

VII. Related Work

To the best of our knowledge, all existing blocklists targeted for cryptocurrency provide very limited user privacy protection, if any. In addition, the sensitive query services they provide are essentially non-transparent and highly centralized. Nonetheless, we identify notable works closely related to ours.

Privacy-preserving data query. Private Information Retrieval (PIR) [29], [52] is a cryptographic protocol that eliminates the chance for a server to learn which items were retrieved by a user from a specified position in a database. To get rid of the assumption that users are knowledgeable about the indices of desired items, Private Keyword Search (PKS) [35] is later proposed. Typical ways to construct PKS include the use of polynomial oblivious evaluation [25] and multiparty computations [53]. However, they are highly inefficient for our type of usage scenarios [33]. In contrast, our work targets different security settings, resulting in much better performance.

Decentralized governance. Decentralized autonomous organizations (DAO) [17] have been widely adopted in the industry. It allows participants to collaboratively reach a consensus with verifiability. In academia, a few works [54]–[57] have also proposed similar blockchain-based e-voting designs over the years. However, most of the constructions partially protect the privacy and anonymity of voters. Additionally, they also do not consider coercion and cost-effective countermeasures.

Decentralized private computation. Numerous works have studied the privacy problems of blockchain-enabled applications. Earlier research efforts focus on providing generic frameworks [58]–[61] to realize private payments via anonymity sets. Notably, these designs and our decentralized evaluation procedures are highly complementary, as it is desirable for our solution to also enable privacy transactions for shareholders to deposit and withdraw money. Beyond private payments, follow-up works like Hawk [62] and Zether [18] extend the core idea behind Zerocash [58] to protect private data inputs of applications. Lately, ZEXE [63] further provides a means to hide computation. Compare to these generic frameworks built on zero knowledge succinct non-interactive argument of knowledge (zk-snark) [20]–[23], our customized NIZK protocols have clear performance advantages.

VIII. Conclusion

In light of the growing cybercrimes in the decentralized space, we present a suite of solutions to enhance the current paradigm of cryptocurrency blocklists. By a careful consolidation of the query procedures and quality evaluation procedures with customized designs, our proposed schemes improve both the security and effectiveness of cryptocurrency blocklist services with bearable on-chain costs. We hope this work will be beneficial to cryptocurrency users and to the wider adoption of secure and high-quality cryptocurrency blocklist services.

Acknowledgements

This work was fully supported by the Research Grants Council of Hong Kong under Grant CityU 11217819, 11217620, 11218521, N_CityU139/21, R6021-20F, RFS2122-1S04, and also by Shenzhen Municipality Science and Technology Innovation Commission (grant number SGDX20201103093004019, CityU), as well as the National Natural Science Foundation of China under Grants 61572412.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.” https://bitcoin.org/bitcoin.pdf, 2008.

[2] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger.” https://gavwood.com/paper.pdf, 2014.

[3] J. Wu, Q. Yuan, D. Lin, W. You, W. Chen, C. Chen, and Z. Zheng, “Who are the phishers? phishing scam detection on ethereum via network embedding,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

[4] M. Bartoletti, B. Pes, and S. Serusi, “Data mining for detecting bitcoin ponzi schemes,” in *Proc. of CVCBT*, 2018.

[5] W. Chen, Z. Zheng, E. C. Ngai, P. Zheng, and Y. Zhou, “Exploiting blockchain data to detect smart ponzi schemes on ethereum,” *IEEE Access*, vol. 7, pp. 37575–37586, 2019.

[6] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, “Dissecting ponzi schemes on ethereum: Identification, analysis, and impact,” *Future Gener. Comput. Syst.*, vol. 102, pp. 259–277, 2020.

[7] Chainalysis, “The 2021 crypto crime report.” https://go.chainalysis.com/rs/503-FAP-074/images/Chainalysis-Crypto-Crime-2021.pdf, 2021.

[8] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, “Phishnet: Predictive blacklisting to detect phishing attacks,” in *Proc. of INFOCOM*, 2010.

[9] A. Le, A. Markopoulou, and M. Faloutsos, “Phishdef: URL names say it all,” in *Proc. of INFOCOM*, 2011.

[10] “Google Safe Browsing.” https://safebrowsing.google.com, 2021.

[11] BitcoinAbuse, “Bitcoin abuse database index.” https://www.bitcoinabuse.com/reports, 2021.

[12] CryptoScamDB, “Cryptoscamdb: open-source dataset tracks malicious urls and their associated addresses.” https://api.cryptoscamdb.org/v1/addresses, 2021.

[13] Bellingcat, “Cryptosint.” https://www.bellingcat.com/?s=crypto, 2022.

[14] S. Ramanathan, J. Mirkovic, and M. Yu, “BLAG: improving the accuracy of blacklists,” in *Proc. of NDSS*, 2020.

[15] G. Tsoukalas and B. H. Falk, “Token-weighted crowdsourcing,” *Manag. Sci.*, vol. 66, no. 9, pp. 3843–3859, 2020.

[16] ConsenSys, “Partial lock commit reveal voting system that utilizes erc20 tokens.” https://github.com/ConsenSys/PLCRVoting, 2018.

[17] Ethereum.org, “Decentralized autonomous organisations (daos).” https://ethereum.org/en/dao/, 2018.

[18] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, “Zether: Towards privacy in a smart contract world,” in *Proc. of FC*, 2020.

[19] E. C. Crites, M. Maller, S. Meiklejohn, and R. Mercer, “Reputable list curation from decentralized voting,” *Proc. of PETS*, vol. 2020, no. 4, pp. 297–320, 2020.

[20] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: Nearly practical verifiable computation,” in *Proc. of IEEE S&P*, 2013.

[21] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, “Snarks for c: Verifying program executions succinctly and in zero knowledge,” in *Proc. of CRYPTO*, 2013.

[22] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, “Succinct non-interactive zero knowledge for a von neumann architecture,” in *Proc. of USENIX Security*, 2014.

[23] J. Groth, “On the size of pairing-based non-interactive arguments,” in *Proc. of EUROCRYPT*, 2016.

[24] M. Naor and O. Reingold, “Number-theoretic constructions of efficient pseudo-random functions,” in *Proc. of FOCS*, 1997.

[25] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, “Keyword search and oblivious pseudorandom functions,” in *Proc. of TCC*, 2005.

[26] M. Naor, “Bit commitment using pseudo-randomness,” in *Proc. of CRYPTO*, 1989.

[27] M. Bellare and O. Goldreich, “On defining proofs of knowledge,” in *Proc. of CRYPTO*, 1992.

[28] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Proc. of CRYPTO*, 1986.

[29] E. Kushilevitz and R. Ostrovsky, “Replication is NOT needed: SINGLE database, computationally-private information retrieval,” in *Proc. of FOCS*, 1997.

[30] B. Pinkas, T. Schneider, and M. Zohner, “Faster private set intersection based on OT extension,” in *Proc. of USENIX Security*, 2014.

[31] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart, “Protocols for checking compromised credentials,” in *Proc. of ACM CCS*, 2019.

[32] L. Sweeney, “k-anonymity: A model for protecting privacy,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[33] H. Cui, Y. Zhou, C. Wang, X. Wang, Y. Du, and Q. Wang, “PPSB: An open and flexible platform for privacy-preserving safe browsing,” *IEEE Trans. on Dependable and Secure Computing*, 2019.

[34] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein, “Protecting accounts from credential stuffing with password breach alerting,” in *Proc. of USENIX Security*, 2019.

[35] Y. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Proc. of ACNS*, 2005.

[36] “Phishtank.” https://www.phishtank.com, 2021.

[37] A. Asgaonkar and B. Krishnamachari, “Token curated registries - a game theoretic approach,” 2018.

[38] randao.org, “Randao: A dao working as rng of ethereum.” https://github.com/randao/randao, 2016.

[39] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts (sok),” in *Proc. of POST*, 2017.

[40] H. S. Galal and A. M. Youssef, “Verifiable sealed-bid auction on the ethereum blockchain,” in *Proc. of FC*, 2018.

[41] F. Hao, P. Y. A. Ryan, and P. Zielinski, “Anonymous voting by two-round public discussion,” *IET Inf. Secur.*, vol. 4, no. 2, pp. 62–67, 2010.

[42] Y. Dodis and A. Yampolskiy, “A verifiable random function with short proofs and keys,” in *Proc. of PKC*, 2005.

[43] J. Burns, D. Moore, K. Ray, R. Speers, and B. Vohaska, “Ec-oprf: Oblivious pseudorandom functions using elliptic curves.” Cryptology ePrint Archive, Report 2017/111, 2017.

[44] “Rust implementation of group operations on ristretto and curve25519.” https://github.com/dalek-cryptography/curve25519-dalek, 2021.

[45] M. Hamburg, “The ristretto group.” https://ristretto.group/, 2021.

[46] A. Biryukov, D. Dinu, and D. Khovratovich, “Argon2: New generation of memory-hard functions for password hashing and other applications,” in *Proc. of EuroS&P*, 2016.

[47] F. S. Foundation, “The gnu multiple precision arithmetic library.” https://gmplib.org/, 2021.

[48] “Webassembly (abbreviated wasm) is a binary instruction format for a stack-based virtual machine.” https://webassembly.org/, 2021.

[49] “Ethereum webassembly (ewasm).” https://ewasm.readthedocs.io/en/mkdocs/, 2021.

[50] “Apache jmeter.” https://jmeter.apache.org/, 2021.

[51] “Eth gas station.” https://ethgasstation.info/calculatorTxV.php, 2021.

[52] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, “Private information retrieval,” *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[53] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. M. Bellovin, “Blind seer: A scalable private DBMS,” in *Proc. of IEEE S&P*, 2014.

[54] K. Lee, J. I. James, T. G. Ejeta, and H. J. Kim, “Electronic voting service using block-chain,” *J. Digit. Forensics Secur. Law*, vol. 11, no. 2, pp. 123–136, 2016.

[55] S. Bistarelli, M. Mantilacci, P. Santancini, and F. Santini, “An end-to-end voting-system based on bitcoin,” in *Proc. of SAC*, 2017.

[56] M. A. Azad, S. Bag, and F. Hao, “Privbox: Verifiable decentralized reputation system for online marketplaces,” *Future Gener. Comput. Syst.*, vol. 89, pp. 44–57, 2018.

[57] B. Zhang, R. Oliynykov, and H. Balogun, “A treasury system for cryptocurrencies: Enabling better collaborative intelligence,” in *Proc. of NDSS*, 2019.

[58] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, “Zerocash: Decentralized anonymous payments from bitcoin,” in *Proc. of IEEE S&P*, 2014.

[59] E. Cecchetti, F. Zhang, Y. Ji, A. E. Kosba, A. Juels, and E. Shi, “Solidus: Confidential distributed ledger transactions via PVORM,” in *Proc. of ACM CCS*, 2017.

[60] N. Narula, W. Vasquez, and M. Virza, “zkledger: Privacy-preserving auditing for distributed ledgers,” in *Proc. of USENIX NSDI*, 2018.

[61] S. Noether and B. Goodell, “Triptych: Logarithmic-sized linkable ring signatures with applications,” in *Proc. of ESORICS*, 2020.

[62] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *Proc. of IEEE S&P*, 2016.

[63] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, “ZEXE: enabling decentralized private computation,” in *Proc. of IEEE S&P*, 2020.