A Moderation Framework for the Swift and Transparent Removal of Illicit Blockchain Content

Roman Matzutt, Vincent Ahlrichs, Jan Pennekamp, Roman Karwacik, Klaus Wehrle
Communication and Distributed Systems
RWTH Aachen University
Aachen, Germany
lastname@comsys.rwth-aachen.de

***Abstract*—Blockchains gained tremendous attention for their capability to provide immutable and decentralized event ledgers that can facilitate interactions between mutually distrusting parties. However, precisely this immutability and the openness of permissionless blockchains raised concerns about the consequences of illicit content being irreversibly stored on them. Related work coined the notion of redactable blockchains, which allow for removing illicit content from their history without affecting the blockchain's integrity. While honest users can safely prune identified content, current approaches either create trust issues by empowering fixed third parties to rewrite history, cannot react quickly to reported content due to using lengthy public votings, or create large per-redaction overheads.**

In this paper, we instead propose to outsource redactions to small and periodically exchanged juries, whose members can only jointly redact transactions using chameleon hash functions and threshold cryptography. Multiple juries are active at the same time to swiftly redact reported content. They oversee their activities via a global redaction log, which provides transparency and allows for appealing and reversing a rogue jury's decisions. Hence, our approach establishes a framework for the swift and transparent moderation of blockchain content. Our evaluation shows that our moderation scheme can be realized with feasible per-block and per-redaction overheads, i.e., the redaction capabilities do not impede the blockchain's normal operation.

***Index Terms*—redactable blockchain, illicit content, chameleon hash functions, threshold cryptography**

I. Introduction

Permissionless blockchains, such as Bitcoin [1], allow anybody to participate [2]: For instance, anyone can propose new transactions, run a full node to revalidate the blockchain, or attempt to mine new blocks and earn rewards. While this openness enables interactions among unknown or distrusting parties, it becomes problematic if users irreversibly store *illicit content* on the blockchain [3]. In 2019, for example, police investigated in the UK after someone stored child abuse imagery on the blockchain of Bitcoin SV [4], [5]. Such incidents show that blockchain networks must become capable of *swiftly* and *proactively* counteracting illicit blockchain content.

Redactable blockchains [6]–[11] emerged to enable retrospective blockchain modifications, e.g., to remove illicit content. This way, honest node operators can safely delete redacted data and remain capable of fully validating the (modified) blockchain. However, previously proposed redaction schemes have to trade off trust, flexibility, and performance, i.e., they either give the control over redactions to a fixed set of few (trusted) nodes [6], [7], they establish transparency by relying on on-chain voting with large delays of, e.g., one week [8]–[10], or they impose large per-redaction overheads [11].

In this work, we thus propose *RedactChain*, a *moderation framework* that enables the swift retrospective redaction of illicit content while ensuring transparency in the permissionless setting. RedactChain achieves this desirable combination by outsourcing redactions to multiple small redaction juries that are periodically elected at random. Furthermore, juries coordinate and *mutually oversee* each other via a separate redaction log, e.g., to prevent a jury from stalling the removal of content. Each redaction jury can swiftly decide to redact reported transactions. However, they have to cooperate to do so, and they can only modify a small portion of the blockchain for a limited time. To account for cases where illicit content remained undetected for a long time, RedactChain can still use a long-term voting scheme (e.g., [8]) as a fallback mechanism. In our design, we rely on chameleon hash functions (CHFs) [6], [12], [13], which enable anybody knowing a secret trapdoor key to modify individual blocks efficiently. In contrast to other CHF-based approaches, RedactChain does not rely on a fixed CHF controlled by a fixed set of redactors. Instead, newly elected juries establish new CHFs via distributed key generation (DKG) [14]. Further, we subject all redactions to rules that enable juries to redact content even from spendable transaction outputs without permitting arbitrary modifications, e.g., they prevent attempts to override previous payments. Finally, the redaction log provides a transparency ledger of all accepted modifications, which additionally supports the coordination between currently active redaction juries.

Contributions. We make the following main contributions to redacting illicit content from permissionless blockchains:

- We enable *swift-and-transparent* redactions in the permissionless setting, i.e., without relying on a trusted third party.
- We define a *moderation framework* that enables multiple juries to coordinate redactions and resolve disputes.
- We show how even theoretically spendable transactions can be redacted *globally* while retaining their spendability.
- We detail that short-term redaction schemes can be combined with slower long-term redaction schemes that, in turn, enable redactions also within very old blocks.

Author manuscript.

II. The Need for Content Moderation

We first reiterate content insertion practices found in Bitcoin before discussing related work and arguing for the need for swift-and-transparent redactions in the permissionless setting.

A. Blockchain Content Insertion in Bitcoin

Non-financial content has been inserted into Bitcoin's blockchain in both intended and unintended ways in the past.

Bitcoin offers two *intended* means for inserting small chunks of arbitrary data [3]. First, miners can add roughly 100 B to the coinbase field of their blocks [3]. Second, users can augment their transactions with up to 83 B of data by adding a single `OP_RETURN` output per transaction. This limited method is widely accepted for realizing Bitcoin-backed applications [15], [16]. However, other Bitcoin forks increase the allowed payload size of `OP_RETURN` outputs [17]. This way, `OP_RETURN` has been abused in the past to engrave child abuse imagery, prompting a police investigation in the UK [5].

Furthermore, Bitcoin users had already earlier abused *unintended* ways to engrave larger data volumes by replacing the mutable fields of standard transactions with their data chunks [3]. For instance, each pay-to-public-key-hash (P2PKH) output contains a 20 B-long hash value that defines the associated coins' owner. A content inserter might create a large transaction with thousands of manipulated P2PKH outputs to engrave kilobytes of data [3]. In contrast to intended insertion methods, this approach creates hypothetically spendable transactions and, thus, simply deleting manipulated outputs might impact the network's consensus. For instance, removing spent outputs would break the transaction graph.

B. Related Work

Previous research has dealt with unwanted blockchain content by either *preventing* its insertion, *locally erasing* content, or creating different brands of *redactable* blockchains.

Content Prevention. One approach to reducing the risks of blockchain content is preventing it from being recorded. Available strategies include: (a) detecting content in pending transactions, (b) financially disincentivizing the creation of large transactions, or (c) hardening the easily replaceable hash values of transaction outputs against manipulation [18]. However, while these strategies can limit the extent of content insertion, none of them can provide full protection [18].

Local Erasure. Node operators can further erase transactions from their local blockchain copy [19]. However, the node operator then depends on other nodes to verify transactions spending erased outputs. This dependency can be mitigated by obfuscating transaction outputs instead of fully erasing them, e.g., by hashing blockchain identifiers again [19]. However, node operators have to actively maintain their local blockchain copy, which implies further administrative overhead for them.

Trust-based Redactions. Another branch of research thus creates redactable blockchains, which allow for retrospective blockchain modifications. Initial solutions [6], [7] use a system-wide and fixed *chameleon hash function (CHF)* [12], [13] to chain blocks. Computing the hash value via a CHF involves a public key, and anyone knowing the corresponding secret *trapdoor key* can compute collisions efficiently. Blockchain designs can thus determine a redactor (or a small group) who is in charge of redacting illicit content. However, the redactor needs to be trusted to execute redactions faithfully.

Voting-based Redactions. To mitigate strong trust assumptions, other redactable blockchains rely on public voting instead of fixed redactors. On the one hand, miners can use their blocks to vote on user-proposed redactions [8]–[10]. Previously proposed voting periods should last for 1024 Bitcoin blocks [8], [10], which corresponds to roughly one week of real-world time. On the other hand, the redaction capabilities can be outsourced to smaller consortia, who only have temporal redaction capabilities; however, as of now, this delegation of control comes with considerable per-redaction overheads [11].

Policy-based Redactions. Finally, policy-based redaction schemes [20]–[24] enable transaction owners to specify redaction policies, e.g., to account for privacy concerns or achieve GDPR compliance. However, these approaches cannot protect against users who insert illicit content, as the transaction owner has to cooperate to remove such content.

C. Missing Swift-and-Transparent Redactions

Previous reports warned that illicit blockchain content has the potential to jeopardize permissionless blockchain systems such as Bitcoin [3], [6], [19]. In 2019, we further observed a police investigation triggered by child abuse imagery engraved on the blockchain of a Bitcoin fork [5]. Since such content is distributed to all full nodes in the permissionless setting, the network has to react swiftly and provide transparency to remove identified content without reducing trust in the blockchain's overall immutability.

Unfortunately, our analysis shows that previous proposals do not provide swift *and* transparent redactions that also cover unintended insertion methods without complicating the validation process. Content prevention [18] mitigates blockchain content but only provides heuristics, i.e., full prevention cannot be guaranteed. Local erasure [19] allows node operators to react quickly but burdens them with removing the illicit content themselves. While redactable blockchains promise to unburden node operators, trust-based redactions [6], [7] create a point of centralization and do not provide transparency. Contrarily, voting-based redactions either react only slowly [8]–[10] or create considerable per-redaction overheads [11]. Finally, policy-based redactions [20]–[24] require the transaction creator to cooperate, which renders them inapplicable in our scenario.

Furthermore, previous approaches focused on the technical enablers for redactable blockchains. Other crucial aspects tend to remain unaddressed: Potentially spendable transaction outputs can hold content, but they are either explicitly declared non-redactable [8], [10] or not considered. However, redacting spendable outputs is required, e.g., to counteract P2PKH manipulation, but such modifications may affect the validation of future transactions and thus need special treatment.

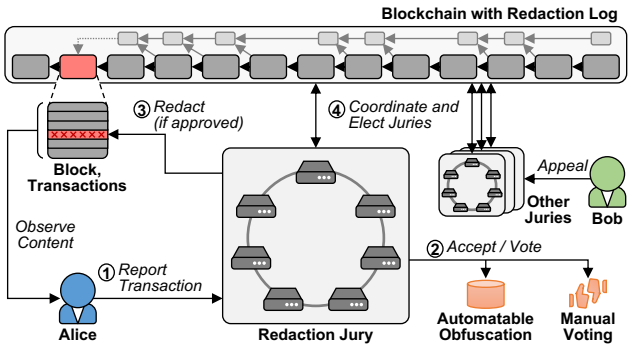

Fig. 1: Overview of RedactChain's moderation framework.

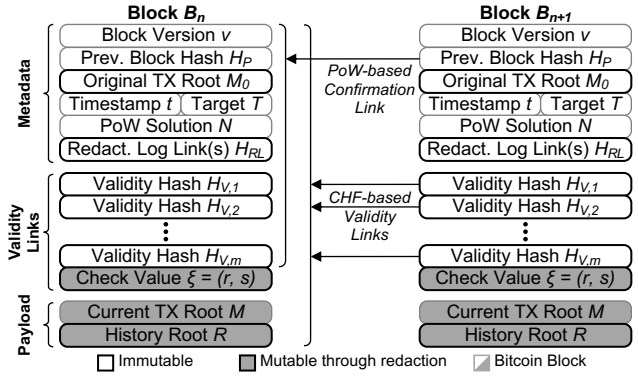

Fig. 2: Block structure of RedactChain compared to Bitcoin.

Further, any redaction also changes the identifiers of affected transactions. Nodes must account for changing identifiers if transaction outputs remain spendable after the redaction.

In the following, we address this research gap by presenting *RedactChain*, a swift-and-transparent moderation framework for redacting illicit content in the permissionless setting.

III. RedactChain Overview

We first give an overview of RedactChain's components, moderation process, and block structure before presenting the individual redaction steps in detail in the following sections.

Moderation Process. Figure 1 shows how RedactChain's components interact to moderate and execute redactions in a swift-and-transparent manner in four steps (Steps ①–④):

Any user can ① report a transaction for illicit content, providing a claim arguing for redacting the transaction to one of m simultaneously active *juries* J_i with n members each.

The jury then ② votes off-chain whether or not to redact the reported transaction and ③ executes the redaction depending on the outcome of the voting process. Similar to previous approaches [6], [7], RedactChain uses chameleon hash functions (CHFs) [12], [13] to grant juries the ability to redact blocks. However, RedactChain limits the juries' influence in multiple ways to prevent misuse of power. First, we rely on distributed key generation (DKG) [14] to distribute the control over a CHF's secret trapdoor key across the jury members. RedactChain further defines strict rules for valid modifications. For instance, each jury may modify any transaction at most once, and other nodes will reject further modifications once the transaction has not been modified for Δ_R blocks to ensure that disputes are ultimately settled. New juries are elected every Δ_D blocks from the recently successful miners, i.e., replaced juries can no longer modify new blocks. To prevent older content from becoming non-redactable, RedactChain can fall back onto slower public voting processes (e.g., [8]). Furthermore, in contrast to previous solutions, RedactChain also supports the global redaction of theoretically spendable outputs (e.g., manipulated P2PKH transactions) by *obfuscating* instead of deleting these outputs. Since the spendability of such transactions is not affected by the obfuscation step, juries can execute such redactions without having to inspect content inserted via unintended means manually.

Juries ④ coordinate via a separate *redaction log*. The redaction log serves three functions. First, juries record *update entries* on the redaction log that document individual modifications, i.e., redactions or restorations. Second, the redaction log is coupled to the main blockchain to establish consensus about an approximate timing of all events. Nodes maintain per-transaction *redaction timers* $\Delta_{R,t}$ based on this approximate timing. The redaction timer $\Delta_{R,t}$ of transaction t gets reset whenever a valid modification of t is recorded on the redaction log, and it expires when it was not reset for Δ_R blocks. Nodes reject any modification proposed after $\Delta_{R,t}$ expires. This way, RedactChain prevents juries from overstepping their competencies and establishes transparency. Further, users can now appeal to unwarranted redactions by presenting the original transaction to another jury. Finally, newly assembled juries announce their new CHFs' public values via the redaction log so that miners can start using those CHFs.

Blockchain Structure. Figure 2 shows RedactChain's block structure (compared to Bitcoin's) that facilitates its multi-jury approach. RedactChain uses one *validity link* $H_{V,i}$ per jury to assert the previous (potentially redacted) block's current state in addition to the common PoW-based *confirmation link* H_P. The confirmation link only covers the immutable fields of a block header. Each block holds an immutable copy of its *initial* Merkle tree root M_0 to be covered by the next block's confirmation link. Additionally, each block has m validity links corresponding to the m active juries' CHFs. By checking the validity links, any node can assert that the block is in its initial state or that a responsible jury has redacted it. Namely, jury J_i redacts a block by updating its check value $\xi = (r, s)$ via its CHF's trapdoor key (cf. Section V-A) to keep $H_{V,i}$ of the next block intact. Finally, the history root R accumulates all update entries corresponding to modifications of transactions within the current block (cf. Section V-C).

IV. Detecting Unwanted Content

We now detail how users ① report transactions and how juries ② handle these reports, i.e., vote whether to redact.

Filing Reports. Similar to previous solutions [8], [10], [11], RedactChain enables users to report transactions containing illicit content. Besides normal users, established organizations fighting illicit content, such as the Internet Watch Foundation

(IWF) [25], can monitor the blockchain and report unwanted and objectionable content in a timely manner. A user files their report by contacting a randomly selected jury based on the members' hostnames, ports, and public keys published as part of the jury election process (cf. Section VI-A). Each report consists of a transaction's identifier and a succinct claim of why the content should be redacted. RedactChain relies on a reliable-broadcast primitive [26] for filing reports to ensure that all jury members receive the same report.

Handling Reports. After receiving a report, the jury engages in an off-chain decision-making process to determine whether to redact the reported transaction. Here, we distinguish content inserted via unintended and intended methods as both means have distinctly different characteristics (cf. Section II-A). Namely, intended methods typically only allow for storing small data chunks (and past negative examples [4], [5] provide arguments for keeping this line), and unintended methods can circumvent these restrictions by interfering with the validation of financial transactions.

We first consider handling unintended insertion methods. Since the predominantly used methods rely on manipulating mutable identifiers in transaction outputs, RedactChain does not remove affected outputs during the redaction step but only obfuscates them (cf. Section V-B). Thus, juries can automate the decision-making process and always redact transactions reported for embedding content via unintended means because this form of redaction does not affect the outputs' spendability. However, the obfuscation inflicts overheads. Using further heuristics, e.g., requiring a minimum number of outputs [18] or explicitly checking for manipulations via content detectors [3], can help identify false reports and reduce these overheads.

Contrary to this identifier manipulation, intended insertion methods, by definition, are not abusing blockchain features. Hence, the jury members have to manually vote for or against redacting corresponding content. We expect only few instances of such manual voting since encoding objectionable content is harder due to these methods' limited capacity. In scenarios where the insertion of larger data chunks is desired (e.g., via the Bitcoin Data Protocol [27], which is used to insert up to 100 kB per `OP_RETURN` [27]), jury members need more support. For example, public access to fingerprint databases of known illicit material could vastly simplify the safe and automatable detection of such material [28]. Alternatively, stakeholders such as the IWF, which already works with such databases to fight child abuse on the Internet [25], could monitor the blockchain for such content.

V. Decentralized Redaction Process

After approving a report, a jury needs to ③ redact the corresponding transaction. We now detail this redaction process.

A. Threshold CHFs for Trusted Redactions

RedactChain relies on distributed CHFs to enable juries to modify the blockchain without preventing other nodes from efficiently verifying the updated blockchain's integrity. Whenever a new jury is elected (cf. Section VI-A), it first establishes and announces a new CHF and can subsequently start to jointly apply that CHF to execute redactions.

Establishment. RedactChain makes use of the same class of CHFs based on Nyberg-Rueppel signatures [13], [29] that was used in initial redactable blockchains [6]. This scheme uses a *check value* $\xi = (r, s)$ to enable efficient collisions, which is chosen randomly for the initial hashing. When computing a collision, ξ is then updated based on the new input message, the secret trapdoor key, and a randomly chosen *collision value*.

Despite providing fast redactability, the initial proposal for redactable blockchains [6] relies on one fixed CHF, i.e., the redacting nodes are fixed. Instead, RedactChain dynamically exchanges CHFs over time to mitigate trust issues. When a jury is elected (cf. Section VI-A), the jury's members first establish connections to each other and create a new CHF. The jury uses DKG [14] to generate a new distributed secret trapdoor key, i.e., each member only holds a share (according to Shamir's secret-sharing scheme [30]) of the secret key, but all members obtain the corresponding public key required to compute hash values. The jury then writes the public key to the redaction log as a *jury assembly block* so that miners can start mining blocks using the new CHF (cf. Section VI-A).

Secure Collision Computation. The jury can now redact transactions from blocks mined during their duty period by computing a collision for their CHF in a decentralized manner. The initial CHF-based proposal suggested recombining the secret trapdoor key before computing the collision [6]. However, this approach enables single jury members to issue arbitrary redactions from then on. RedactChain instead decentralizes the computation of a collision value k' and uses the homomorphic properties of Shamir's secret-sharing scheme [31] to locally compute the new check value $\xi' = (r', s')$, i.e., update the block without changing its hash value. The jury members draw a new random collision value k' also using DKG, as that value needs to remain secret. Furthermore, the nodes can update r' locally using the collision value's public value $g^{k'}$, but computing s' involves knowing k' [13, Section 4]; hence, the nodes first prepare shares of s' and then recombine those.

B. Updating Transactions and Blocks

Besides computing CHF collisions, the jury has to update affected transactions and blocks to execute redactions.

Replacing Transactions. Each jury member locally updates transactions to be redacted as an input for our decentralized CHF collision computation (cf. Section V-A). The removal technique depends on the applied insertion mode as derived from the user's report.

Data inserted via *intended methods*, i.e., `OP_RETURN` and coinbase, is not tied to other transactions. Such content can be removed without affecting the validation of other blocks.

Contrarily, *unintended methods* rely on manipulating spendable outputs (e.g., P2PKH), which hold coins. Simply removing such outputs can have side effects as (a) coins can be burned, and (b) some of the outputs might be spendable or already spent. We prevent breaking the transaction graph by only *obfuscating* content from manipulated transactions,

i.e., the content cannot be recovered after the redaction but any spendable output remains spendable. To this end, we cryptographically hash the mutable identifiers a second time instead of removing them, which is in line with strategies proposed for local content erasure [19] or obfuscating illicit content from Bitcoin's UTXO set [32]. When a user attempts to spend an obfuscated output, the node can still validate the transaction by executing additional hashing on the fly.

Finally, *other insertion methods*, e.g., using non-standard transactions or input scripts [3], need to be mitigated via strictly enforcing standardness tests [33] also on transactions mined into a block. For instance, redacting input scripts always breaks the transaction graph as they, by definition, satisfy a previous transaction's spending condition. Removing such content thus relates to local erasure (cf. Section II-B) since the removing node cannot fully validate the blockchain anymore.

Updating Blocks. To conclude the redaction, the jury J_i updates the corresponding block (cf. Figure 2). Each member computes an update entry describing the edit and appends it to the block's update history (cf. Section V-C). Each member updates the Merkle root M and the history root R accordingly. Then, the jury can jointly update the check value ξ using the secure collision computation (cf. Section V-A). This way, the confirmation link H_P and the validity link $H_{V,i}$ of the block's successor remain intact and assert that the modified block was previously confirmed and that it was altered by one of the responsible juries. Finally, the jury broadcasts the updated block, and other nodes accept the change after verifying it.

C. Ensuring the Transparency of Redactions

Besides preserving the blockchain's integrity regardless of redactions via CHFs, RedactChain maintains a per-block update history as well as a dedicated redaction log to provide transparency and to coordinate juries (cf. Section VI-B).

Update Entries. An *update entry* describes one blockchain modification, i.e., a redaction or restoration. Each entry is indexed by a block-level counter, which is incremented for each edit, and the index i of the redacting jury J_i. Furthermore, each entry summarizes the block's state before the edit as well as the reason for the edit based on the initial user report. The j-th edit of a block covers the block's previous state by stating the edited transaction's old identifier t_{j-1}, the old Merkle root M_{j-1} and history root R_{j-1}, and the old check value ξ_{j-1}. This way, other nodes can verify that the block was in a valid state before, reasons for redactions become public, and keeping track of old transaction identifiers helps nodes to validate future transactions referencing edited transactions.

Update History. Each block has an *update history*. This history contains all update entries affecting a block and is tied to its header via the history root R. The update history serves two purposes. First, nodes can comprehend and verify every past edit of the block. Second, even joining nodes learn about changes of transaction identifiers due to a redaction to validate transactions spending outputs of modified transactions.

Redaction Log. In addition to per-block update histories, RedactChain also keeps a global *redaction log* of all activities.

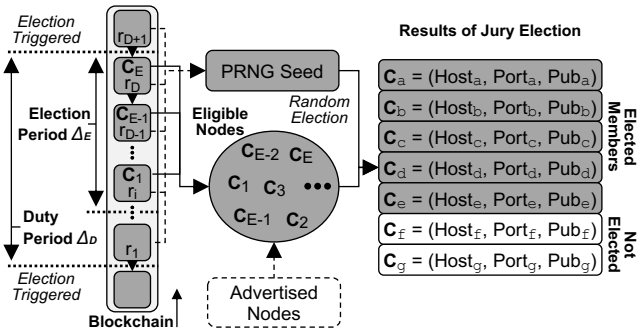

Fig. 3: Default jury election. Nodes locally draw n members of the last Δ_E successful miners from a pseudo-random shuffling.

The redaction log is a separate append-only ledger that is publicly readable but only extended by juries. Namely, juries publish their jury assembly block (cf. Section VI-A) as well as the update entries of their blockchain edits on the redaction log. Each update entry is additionally encapsulated with backlinks confirming prior redaction log entries, a timestamp of the entry's creation, and a signature jointly created by the jury. We use Nyberg-Rueppel signatures so that juries can obtain the required randomness by running an additional DKG instance in parallel during the collision computation for the redaction (cf. Section V-A). Furthermore, the redaction log helps to establish consensus about the *approximate timing* of events among all nodes by coupling the redaction log to the main blockchain. Namely, each block references the redaction log's current tip via H_{RL} (cf. Figure 2). Thereby, the block confirms that all entries on the redaction log existed at the time the block was mined. Even though dishonest miners may attempt to skew this timing by ignoring legitimate entries, honest miners will faithfully confirm the most recent entries to confirm all legitimate entries in a timely manner. Dishonest miners further cannot predate entries because the signature-based linking prevents inserting entries into an existing path. Finally, miners can merge redaction log forks and confirm the insertion time also of entries deliberately appended to old entries by confirming multiple tips simultaneously.

VI. Coordinating Redaction Juries

Juries have to ④ coordinate the handover of redaction capabilities and their activities. We further discuss our fallback mechanism to also redact older transactions that this coordination would otherwise make non-redactable.

A. Jury Election Process

Figure 3 shows the election process for each jury J_i consisting of n members. New juries are elected every Δ_D blocks from a pool of eligible nodes. By default, RedactChain considers all successful miners of the Δ_E most recent blocks eligible to be voted into a jury. Each miner includes its hostname, port, and a public key in its blocks so that juries can bootstrap and users can create an index of how to reach active jury members to file a report. For increased diversity, RedactChain can be extended to also consider a Sybil-resistant pool of non-miners eligible to become jury members [34].

After Δ_D blocks, each node locally creates an eligibility list from this data and shuffles this list based on a pseudo-random number generator (PRNG) that is seeded with randomness drawn from the last Δ_D blocks. RedactChain can be extended with dedicated randomness extractors [35] to further reduce bias from blockchain-sampled randomness. We elect all i juries in parallel by repeatedly hashing the initial PRNG seed and reshuffling the eligibility list accordingly. The nodes then consider the n topmost nodes of the shuffled eligibility list as the elected members of J_i. The elected members also learn their exact position in J_i and how to connect to the other jury members. The juries can then bootstrap and create their jury assembly block. The members sign the assembly block; other nodes only accept assembly blocks with at least $x \geq 2n/3$ valid signatures. If a jury cannot announce the assembly block within a short time (e.g., <10 blocks), the nodes elect a jury J_{m+1} in its place. Finally, the jury publishes the assembly block to the redaction log and starts accepting reports.

B. Coordination and Mutual Oversight

Simultaneously active juries ensure that RedactChain remains actionable and accountable even if single juries misbehave, but the juries need to coordinate. We now present how the redaction log and its approximate timing benefit coordination and enable a fair appeal process for users. Finally, we briefly discuss conflicts stemming from concurrent redactions.

Approximate Timing for Coordination. Nodes mainly use the mining process to coordinate, e.g., new juries are elected every Δ_D blocks. Coupling the redaction log to the main blockchain (cf. Section V-C) further enables an approximate timing for when edits happened. Nodes use this timing to keep track of per-transaction *redaction timers* $\Delta_{R,t}$, i.e., the number of blocks since the last edit of transaction t. Each node resets $\Delta_{R,t}$ whenever a valid edit of t is appended to the redaction log. Conversely, the nodes reject further jury actions once $\Delta_{R,t}$ exceeds a threshold Δ_R and expires. If $\Delta_{R,t}$ expires, t can only be redacted via a slower public voting (cf. Section VI-C).

Fair Appeal Process. Appeals are necessary to settle disputes stemming from manual jury votes or revert unwarranted redactions. Our multi-jury approach enables users to request reverting a past decision at another jury. Users can ask a jury to restore a redacted transaction t by presenting a cached copy of the original state of t. This way, the jury can revert the redaction of t while still allowing all but appealing nodes to immediately remove or obfuscate t locally. If the appeal is successful, the nodes will reset $\Delta_{R,t}$, but keep track of which juries already edited t to reject future modification attempts from those juries. Hence, even strongly disputed transactions ultimately reach a final state as their redaction timer will expire at some point due to no jury intending or being able to issue another edit. However, appeals can cause redaction timers to expire after the juries' duty ends. In these cases, juries may resolve pending disputes before dissolving.

Conflicting Redactions. As juries are active concurrently, their modifications can provoke *conflicts* when they attempt to edit two transactions of the same block simultaneously. Nodes arrange the edits based on their timestamps on the redaction log to prevent accidentally overwriting any edits. In the worst case, the jury losing this tiebreaker has to redo its modification. Since juries are intentionally small (e.g., tens of nodes), they can coordinate their intended edits even before the update entries are confirmed on the redaction log to lower the risk of collisions. Furthermore, the most resource-intensive step of a redaction is using DKG to obtain a collision value (cf. Section VII-C), which is independent of the transaction to be redacted. Hence, juries can delay applying a specific collision value until after the off-chain coordination.

C. Integration of Other Moderation Schemes

While RedactChain's redaction process enables swift redactions without relying on fixed redactors, older transactions become immutable again and could potentially contain overseen content. To mitigate this risk, RedactChain is compatible with long-term, voting-based redaction schemes [8]. Both approaches use an immutable confirmation link as a fallback to validate a block's initial state and a validity link that is invalidated as soon as the block is edited. Hence, RedactChain can be extended to offer a similar long-term voting process for older transactions. In this case, all miners vote on-chain, independent from juries and redaction timers, and all nodes can observe the voting process. This approach introduces significant delays over RedactChain's default redaction process but enables the removal of deeply engraved content. Similarly, content mitigation schemes, which limit the insertion of content (cf. Section II-B), can be deployed with RedactChain to reduce the number of expected redactions.

VII. Evaluation

Now, we evaluate RedactChain's redactions by discussing their effectiveness, security, and performance overheads.

A. Effectiveness and Non-Invasiveness

We first discuss that RedactChain provides a holistic rule set for redactions suitable for decentralized settings while protecting the blockchain's and transaction graph's integrity.

RedactChain can redact small chunks of content inserted via intended means (`OP_RETURN` or coinbase) by simply removing the respective values after a manual vote without side effects on the transaction graph since no spendable transaction outputs are affected. Contrarily, removing potentially spendable outputs, e.g., manipulated P2PKH outputs, can alter the transaction graph. RedactChain prevents breaking the transaction graph by only obfuscating instead of removing potentially spendable outputs during redaction (cf. Section V-B). Nodes can validate pending transactions even against obfuscated outputs by replaying that obfuscation on the fly. However, any modification of a transaction alters its identifier. The update entries attached to each block and the redaction log (cf. Section V-C) ensure that all nodes can keep track of changing transaction identifiers. Since input scripts, e.g., for P2SH outputs, are inherently tied to the transaction graph's integrity, RedactChain does not modify them, and their inclusion must rather be mitigated [18] or handled locally [19]

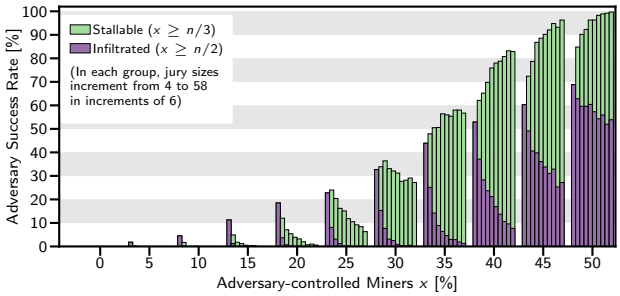

Fig. 4: Success probability of an increasingly powerful adversary to obstruct a single jury (bars are not stacked).

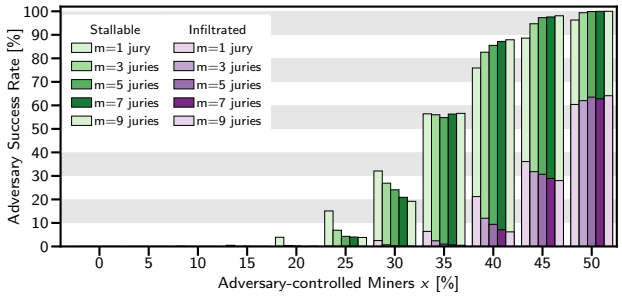

Fig. 5: Mutually overseeing juries (here: $n=28$ peers) increase the resilience against adversaries for $x \leq 30\,\%$.

by other means. Finally, RedactChain prohibits non-standard transactions to reduce the complexity of executing redactions. Overall, RedactChain extends upon the limited redactability achieved by related work in the permissionless setting, which only considers provably unspendable transaction outputs.

We enable swift operation through outsourcing redactions to small juries and deliberately limit their capabilities through our periodic election of new juries (cf. Section VI-A). While older transactions would become immutable this way, we extend RedactChain's coverage by remaining compatible with slower public votes as a fallback (cf. Section VI-C).

B. Security Discussion: Adversary Resilience

We assess RedactChain's security by considering a malicious adversary who intends to stall redactions or perform rogue modifications based on their share of nodes eligible for jury election and the other nodes' verifiability of redactions.

An adversary can modify a transaction t if (a) they know one of the responsible juries' secret trapdoor key *and* (b) the redaction timer $\Delta_{R,t}$ has not yet expired. Secret trapdoor keys are generated using DKG (cf. Section V-A); hence, the adversary can recombine a jury's key and then compute collisions for the jury's CHF to modify blocks if they control $x \geq n/2$ jury members [14]. In this case, we say that the adversary successfully *infiltrated* a jury. Even though the adversary could technically execute redactions at will, they must still adhere to the rule set for redactions (cf. Section V-B). Namely, the adversary can delete only `OP_RETURN` and coinbase fields, but they must correctly obfuscate potentially spendable outputs, i.e., the adversary cannot alter the transaction graph even after infiltrating a jury. The adversary must update the redaction log properly before other nodes accept the edit. Hence, the nodes detect attempts to modify a transaction more than once. If the adversary controls $n/3 \leq x < n/2$ jury members, they can only stall the final recombination step and prevent the honest members from executing redactions. While this strategy can delay a redaction, users can report the transaction to another, actionable jury (cf. Section VI-B) or ultimately fall back to a long-term public voting scheme if required (cf. Section VI-C). As long as the adversary controls only $x < n/3$ members, they cannot affect RedactChain's operability.

Figure 4 shows the probability that an increasingly powerful adversary can stall or infiltrate a single jury. We simulated an adversary who controls an increasing share of a total of 100 equal miners. We fix the duty period at $\Delta_D = 1000$ blocks ($\sim$one week for Bitcoin) and the election period at $\Delta_E = 150$ blocks ($\sim$one day). We then elect 1000 juries of $4, 10, \ldots, 58$ members from the randomly chosen miners of the last Δ_E blocks based on a seed derived from the block identifiers of a randomly chosen sequence of $\Delta_D + 1$ Bitcoin blocks. This simulation shows that, while especially large juries only face a low infiltration risk from an adversary controlling up to $35\,\%$ of the miners, there is a non-negligible risk that an adversary can stall a jury once controlling at least $25\,\%$ of the miners.

We can further reduce this risk by relying on multiple, mutually overseeing juries. An odd number of juries ensures that an honest jury will have control over the final decision for at most $y < m/2$ infiltrated juries. Figure 5 gives the success rate of the adversary of gaining control over $y > m/2$ juries for $m = 1, 3, 5, 7, 9$ and a fixed, moderate (cf. Section VII-C) jury size of $n = 28$. While the adversary has a success rate of $15.1\,\%$ to stall a single jury when controlling $25\,\%$ of all miners, they have a lowered chance of only $3.8\,\%$ of stalling the redaction process for $m = 9$ active juries. Notably, this risk is lower than relying on a single jury of $n = 58$ members and still provides feasible redactions. Even though the adversary can stall redactions temporarily in rare cases, the network can still resort to public voting as a fallback (cf. Section VI-C).

C. Redaction Time

We now discuss the redaction times achieved by RedactChain and its scalability to large network sizes.

Measurement Setup. We measure the redaction time for a single jury based on a Python prototype [36], which uses `aiohttp` for communication and an adapted version of `python-bitcoinlib` to handle redactable blocks. All jury members run on the same server (2× Intel Xeon Silver 4116, 196 GB RAM) and communicate over local TCP connections via RSA-signed messages. We use 2048-bit primes for our cryptographic primitives. We create a simulated blockchain of redactable blocks without mining difficulty by implementing a pre-calculated transaction graph using the adapted `python-bitcoinlib` module. This blockchain consists of a genesis block for distributing initial funds and 1000 blocks with 1000 transactions each. Each block contains two redact-worthy transactions, one with 50 P2PKH outputs and one with an `OP_RETURN` output. From this blockchain, one fixed redaction jury of increasing size redacts both redact-worthy

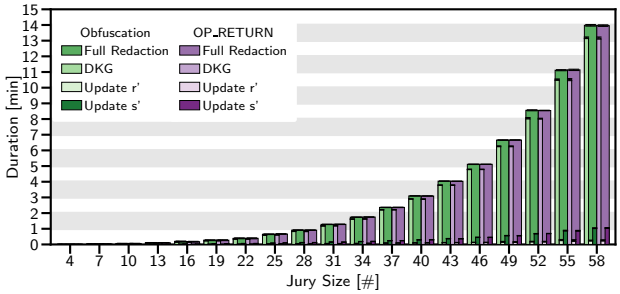

Fig. 6: Larger jury sizes affect the performance of our redaction process, mainly due to DKG.

transactions from the first 30 blocks. We give the duration of the measured redaction steps by averaging the difference between the earliest jury member entering the phase and the last jury member concluding it for each redaction over the 30 blocks. We further give 99 % confidence intervals. Since the phases can overlap between jury members, the sum of phases can be larger than the overall redaction time.

Measurement Results. Figure 6 shows the time it takes a single jury of sizes $4, 7, \ldots, 58$ to redact a transaction after accepting a user's report. We additionally highlight the main steps of the distributed collision computation, i.e., obtaining a collision value using DKG and computing the new check value components r' (local) and s' (involves Shamir recombination).

Our results indicate no substantial difference between obfuscating large transactions and redacting an `OP_RETURN` output. However, the redaction time increases superlinearly for larger jury sizes, mainly due to DKG. For instance, obfuscating a single large transaction takes a jury with $n = 58$ members 13.98 min, whereas a jury with $n = 28$ only requires 54.9 s, and a small jury with $n = 16$ takes only 11.0 s. Hence, relying on multiple but smaller juries can reduce the overhead of individual redactions without forfeiting RedactChain's resilience against adversaries (cf. Section VII-B). We have to create collision values via DKG since knowing a collision value allows retrieving the jury's secret trapdoor key [13]. However, juries can pre-compute an appropriate number of DKG values after their assembly [37]. While this approach does not reduce the overall performance overhead of a redaction, it can further improve the jury's reaction time to user reports. Finally, these redaction times are independent of the total network size.

D. Overhead of Additional Information

RedactChain tracks additional information to keep redactions easily and publicly verifiable. We now discuss the overhead due to this metadata, i.e., our block structure compared to Bitcoin's, the jury assembly blocks, and the redaction log.

Block Header. Bitcoin blocks have a fixed-length 80 B-long header [38]. In addition to this header's fields, RedactChain's block headers keep track of the original Merkle root M_0 (32 B), the $1 \leq i \leq m$ confirmations of the redaction log (1 B for the length, 32 B per link), the m CHF-based validity links (1 B for the length, 2048 bit = 256 B per link), the check value $\xi = (r, s)$ (2048 bit per component), and the history root R (32 B). Hence, the block header has a total size of 946 B for one jury and a worst-case size of 3250 B for nine juries confirming nine redaction log branches (i.e., one branch per jury). Since the average Bitcoin transaction has a size of 250 B [18], even using nine juries would reduce the block capacity by only 13 transactions on average.

Redaction Log and History. Each redaction has an update entry that is referenced in a block's update history and on the redaction log. As described in Section V-C, an update entry consists of the per-block redaction counter (4 B), the redacting jury's index (1 B), the block header's previous state as given by the old transaction identifier, Merkle tree root, and the history root (32 B each), as well as the old check value (2×256 B) and a reason for the redaction. In total, an update entry has a size of 854 B, assuming that the reason has a length of ≤ 240 B (plus a 1 B length field). Each update entry is encapsulated on the redaction log and holds a timestamp (4 B), a signature (256 B), and $1 \leq i \leq m$ confirmations of recent branches of the redaction log ($1 + i \cdot 32$ B). Hence, each entry has a total size between 1147 B and 1403 B. This overhead is significantly lower than recently achieved per-redaction overheads of 60 kB − 110 kB [11].

Jury Assembly Blocks. These blocks publish a new public mining key (256 B), and they hold signatures of $2n/3 < x \leq n$ jury members ($1 + x \cdot 256$ B). Jury assembly blocks are also part of the redaction log and thus hold i confirmation links ($1 + i \cdot 32$ B) as well. Hence, jury assembly blocks remain below 26 kB even for juries with 100 members, and they stay below 7.5 kB for moderate-sized juries of 28 members.

VIII. Conclusion

In this paper, we presented RedactChain as a moderation framework to counteract illicit blockchain content. RedactChain distributes redaction capabilities by operating multiple independent juries in parallel. Each jury consists of randomly selected nodes and is periodically replaced by a new jury. Our utilization of threshold cryptography during redactions and a global redaction log to coordinate the activities of different juries ensures that RedactChain remains actionable and resilient against adversaries. Our design further limits the damage an adversary-controlled jury can inflict, and our evaluation shows that RedactChain is robust against an adversary who controls up to $1/4$ of the total mining power. RedactChain further realizes its swift-and-transparent moderation process with feasible per-redaction and per-block overheads. Hence, illicit content can be redacted in under a minute in contrast to previous day-long or even week-long on-chain voting processes. Due to RedactChain's current focus on Bitcoin-like blockchains, we identify its generalization to other systems (e.g., Ethereum [39]) as promising future work.

ACKNOWLEDGMENTS. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under funding reference numbers 16KIS0443, 16DHLQ013, and Z31 BMBF Digital Campus. The funding under reference number Z31 BMBF Digital Campus has been provided by the German Academic Exchange Service (DAAD). The responsibility for the content of this publication lies with the authors. The authors further thank Eric Wagner, Jan Rüth, and Muhammad Hamad Alizai for the valuable discussions.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," White paper, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] K. Wüst and A. Gervais, "Do you Need a Blockchain?" in *Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 45–54.

[3] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, and K. Wehrle, "A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin," in *International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2018, pp. 420–438.

[4] Money Button. (2019) Against Illegal Content on the Blockchain. Archived on 2021-01-28. [Online]. Available: https://web.archive.org/web/20210128090213/https://blog.moneybutton.com/2019/01/31/against-illegal-content-on-the-blockchain/

[5] BBC News. (2019) Child abuse images hidden in cryptocurrency blockchain. Archived on 2021-06-25. [Online]. Available: https://web.archive.org/web/20210625141025/https://www.bbc.com/news/technology-47130268

[6] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable Blockchain – or – Rewriting History in Bitcoin and Friends," in *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 111–126.

[7] K. Ashritha, M. Sindhu, and K. Lakshmy, "Redactable Blockchain using Enhanced Chameleon Hash Function," in *International Conference on Advanced Computing & Communication Systems (ICACCS)*. IEEE, 2019, pp. 323–328.

[8] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable Blockchain in the Permissionless Setting," in *Symposium on Security and Privacy (S&P)*. IEEE, 2019, pp. 124–138.

[9] A. Marsalek and T. Zefferer, "A Correctable Public Blockchain," in *International Conference on Trust, Security and Privacy in Computing and Communications/International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*. IEEE, 2019, pp. 554–561.

[10] S. A. K. Thyagarajan, A. Bhat, B. Magri, D. Tschudi, and A. Kate, "Reparo: Publicly verifiable layer to repair blockchains," in *International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2021, pp. 37–56.

[11] X. Li, J. Xu, L. Yin, Y. Lu, Q. Tang, and Z. Zhang, "Escaping from Consensus: Instantly Redactable Blockchain Protocols in Permissionless Setting," Cryptology ePrint Archive, Report 2021/223, 2021, version 20211207:102627. [Online]. Available: https://ia.cr/2021/223

[12] H. Krawczyk and T. Rabin, "Chameleon Signatures," in *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2000.

[13] G. Ateniese and B. de Medeiros, "On the Key Exposure Problem in Chameleon Hashes," in *Security in Communication Networks*. Springer, 2005, pp. 165–179.

[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, pp. 51–83, 2007.

[15] M. Bartoletti and L. Pompianu, "An Analysis of Bitcoin OP_RETURN Metadata," in *International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2017, pp. 218–230.

[16] M. Bartoletti, B. Bellomy, and L. Pompianu, "A Journey into Bitcoin Metadata," *Journal of Grid Computing*, vol. 17, no. 1, pp. 3–22, 2019.

[17] Money Button. (2019) How We Added Support for Giant OP_RETURN Data in Money Button. Archived on 2021-01-21. [Online]. Available: https://web.archive.org/web/20210121180521if_/https://blog.moneybutton.com/2019/01/26/how-we-added-support-for-giant-op_return-data-in-money-button/

[18] R. Matzutt, M. Henze, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Thwarting Unwanted Blockchain Content Insertion," in *International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 364–370.

[19] M. Florian, S. Henningsen, S. Beaucamp, and B. Scheuermann, "Erasing Data from Blockchain Nodes," in *European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 367–376.

[20] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to Forget without Hard Forks," Cryptology ePrint Archive, Report 2017/106, 2017, version 20200602:075626. [Online]. Available: https://ia.cr/2017/106

[21] A. Dorri, S. S. Kanhere, and R. Jurdak, "MOF-BC: A memory optimized and flexible blockchain for large scale networks," *Future Generation Computer Systems*, vol. 92, pp. 357–373, 2019.

[22] N.-Y. Lee, J. Yang, M. M. H. Onik, and C.-S. Kim, "Modifiable Public Blockchains Using Truncated Hashing and Sidechains," *IEEE Access*, vol. 7, pp. 173 571–173 582, 2019.

[23] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based," in *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2019.

[24] Y. Tian, N. Li, Y. Li, P. Szalachowski, and J. Zhou, "Policy-Based Chameleon Hash for Blockchain Rewriting with Black-Box Accountability," in *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2020, pp. 813–828.

[25] Internet Watch Foundation. (2015) Hash List. Archived on 2022-03-13. [Online]. Available: https://web.archive.org/web/20220313190554id_/https://www.iwf.org.uk/our-technology/our-services/image-hash-list

[26] G. Bracha, "An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol," in *Symposium on Principles of Distributed Computing (PODC)*. ACM, 1984, pp. 154–162.

[27] "unwriter". (2019) B:// – Bitcoin Data Protocol. Archived on 2022-03-13. [Online]. Available: https://web.archive.org/web/20220313191035if_/https://github.com/unwriter/B

[28] K. Cremona, D. Tabone, and C. De Raffaele, "Cybersecurity and the Blockchain: Preventing the Insertion of Child Pornography Images," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE, 2019, pp. 197–204.

[29] K. Nyberg and R. A. Rueppel, "Message recovery for signature schemes based on the discrete logarithm problem," in *Advances in Cryptology — EUROCRYPT'94*. Springer, 1995, pp. 182–193.

[30] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[31] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," in *Symposium on Theory of Computing (STOC)*. ACM, 1988, pp. 1–10.

[32] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, "CoinPrune: Shrinking Bitcoin's Blockchain Retrospectively," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3064–3078, 2021.

[33] Bitcoin Project. (2011) Protocol rules – Bitcoin Wiki. Archived on 2021-07-14. [Online]. Available: https://web.archive.org/web/20210714145539id_/https://en.bitcoin.it/wiki/Protocol_rules

[34] R. Matzutt, J. Pennekamp, E. Buchholz, and K. Wehrle, "Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services," in *ASIA Conference on Computer and Communications Security (ASIACCS)*. ACM, 2020, pp. 531–542.

[35] J. Bonneau, J. Clark, and S. Goldfeder, "On Bitcoin as a public randomness source," Cryptology ePrint Archive, Report 2015/1015, 2015, version 20151019:205945. [Online]. Available: https://ia.cr/2015/1015

[36] R. Matzutt, V. Ahlrichs, J. Pennekamp, R. Karwacik, and K. Wehrle. (2022) RedactChain: Proof-of-Concept Prototype for the Swift and Transparent Removal of Illicit Blockchain Content. Accessed on 2022-03-13. [Online]. Available: https://github.com/COMSYS/redactchain

[37] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and anonymous decentralized Bitcoin mixing," *Future Generation Computer Systems*, vol. 80, pp. 448–466, 2018.

[38] Bitcoin Project. (2010) Block – Bitcoin Wiki. Archived on 2022-01-10. [Online]. Available: https://web.archive.org/web/20220110114832id_/https://en.bitcoin.it/wiki/Block

[39] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," White paper, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf