Consistency-or-Die: Consistency for Key Transparency

Joakim Brorsson
Hyker
Malmö, Sweden
joakim@hyker.io

Elena Pagnin
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
elenap@chalmers.se

Bernardo David
IT University of Copenhagen
Copenhagen, Denmark
bernardo@bmdavid.com

Paul Stankovski Wagner
Lund University
Lund, Sweden
paul.stankovski_wagner@eit.lth.se

ABSTRACT

In this paper we point out the problem of insufficient tools for protecting against split-view attacks in Key Transparency protocols, and propose a solution to fill the void. We discuss why current approaches are not suitable and then propose a novel notion, GOD-less broadcast, that solves the issue. Like conventional notions of broadcast, GOD-less broadcast guarantees consistency. However, it does not provide Guaranteed Output Delivery (GOD). We provide an efficient realization of this new notion using a hidden committee of randomly selected and initially undisclosed users which endorse the current view. We also introduce and analyze a new concept of a quorum which ensures consistency among all honest active users. Our GOD-less broadcast protocol is practical and applicable to existing large scale Key Transparency systems, e.g. the Key Transparency used in WhatsApp.

KEYWORDS

Key Transparency, Consistency, Transparency Logs, Broadcast

1 INTRODUCTION

A backbone to securing the web is the existence of a trustworthy Certificate Authority (CA) infrastructure. The main role of a web CA is to issue digital certificates to validate the authenticity of websites. These certificates help establish secure connections between a user's browser and a website, ensuring that sensitive information is encrypted and transmitted securely. Additionally, CAs are expected to verify the identity of website owners and help prevent fraudulent activities. Unfortunately, we have witnessed several incidents [42] where incorrect or malicious certificates have successfully been used. For example, Symantec was caught wrongfully issuing a certificate for google.com [38], and DigiNotar was fully compromised by an unknown attacker which during the attack issued over 500 fake certificates [25], which were then used for spying on Iranian citizens. These incidents are due to placing *too much trust* in the CA infrastructure.

To remedy this situation, transparency logs for the web, called Certificate Transparency logs (CT) [28], have been deployed to ensure correct serving of TLS certificates. Many browsers currently mandate their use and thus reduce the trust that needs to be placed in CAs by making them transparent.

Intuitively, the transparency in transparency logs comes from publicly recording all certificates issued by a CA. This allows for greater visibility and accountability in the issuance of certificates, helping to detect any unauthorized or fraudulent certificates. More specifically, transparency logs are label-value data structures which are publicly verifiable to be *append-only*, allowing no data to be deleted or altered, and *consistent*, *i.e.* , serving all users the same view of the current state of the data structure. These two properties ensure that the same value (certificate) is delivered in response to all queries for a specific label.

Recently, transparency logs have been applied to serving public keys for end-to-end encryption messaging apps [10, 26, 29, 30, 32], and deployed in mainstream apps such as WhatsApp [33], iMessage [3] and Zoom [6]. In this case, the technique is called Key Transparency (KT) logs.

Even though they have a similar aim in detecting potential attacks connected to cryptographic material, CT and KT have a few core differences. While CT logs are designed for detecting and preventing issuance of fraudulent TLS certificates, KT protocols focus on providing a secure and transparent way to manage and distribute cryptographic keys. We provide further background on transparency logs and their properties in Appendix A.

1.1 State-of-the-Art and Current Issues

Key Transparency research consists of protocols for ensuring *append-only*, and protocols for ensuring *consistency*. Protocols for ensuring *append-only* [10, 26, 29, 30, 32] constitute the largest body of works, and these protocols have in recent years reached production level maturity. However, protocols for *consistency* have received much less attention, and current methods [7, 22, 30, 31, 44] are unsatisfactory as they provide weak consistency guarantees.

In light of this, it is imperative to develop better consistency protocols. A KT log with weak consistency guarantees cannot protect against split-view attacks (which break the consistency property) and is thus of limited value for ensuring that the correct public keys are served. The need is urgent. KT logs with no or weak consistency guarantees have already been deployed in high profile systems (see Section 1.1.2). At the same time, there have been a number of detected split-view incidents in CT logs [39–41]. Such split-views risk going *undetected* in current proposals for KT.

1.1.1 Academic Proposals. There are a three known methods for achieving consistency in KT protocols; gossip protocols, blockchains and designated witnesses. We here give an overview of these approaches, and provide further details and an extensive literature review on existing KT consistency works in Appendix B.

Gossip protocols [12, 14, 35] achieve consistency by having users gossip over the state of the log using Out-Of-Band (OOB) channels, where the service provider cannot suppress messages. Gossiping essentially consists of exchanging messages that endorse (or oppose) a certain view. While this approach works for CT, it is problematic for KT systems, since OOB channels are unworkable in real-world KT scenarios (matching QR codes in person does not scale well to a user base of, say, a billion users).

Blockchains have been suggested [7, 20, 44] as an alternative to gossip protocols. By using a blockchain, consistency comes for free since each record in a blockchain is cryptographically linked to previous records, in a sequential and immutable manner. However, relying on this technology implicitly implies accepting the costs and assumptions that come with implementing it. We consider the costs, *e.g.* , end users running full blockchain nodes, unrealistic in most mainstream scenarios such as KT for large scale instant messaging apps. Indeed, none of the existing KT deployments have chosen to use blockchains for consistency.

External Committees of Consistency Auditors [18, 30] is the most recent alternative for achieving efficient and scalable consistency in KT. The idea is to appoint an external committee of auditors, where each committee member endorses its view, and where at last two thirds of the members are assumed to be honest. A user can be assured of consistency of its own view if there is a large enough set of committee members (other users) who agree with this view. Security demands that such a committee be large and untargetable, otherwise an adversary could simply corrupt all committee members. However, for efficiency and practical reasons, such a committee should at the same time be small. First of all, because users' CPU and network overhead grow with the committee size. Secondly, because finding a large number of trustworthy external parties can be difficult in practice. Finding one size that fits all needs is a delicate matter, and the numbers used in current proposals (10-50 members [30]) are highly inadequate when dealing with high profile deployments such as WhatsApp, iMessage or Zoom.

In summary, neither gossip, nor blockchains, nor external committees of consistency auditors are satisfying solutions to ensure consistency for key transparency in high profile secure instant messaging apps.

We note that concurrent work [22] explores another direction of ensuring consistency for KT. The work in [22] provides a protocol which does not use external parties for auditing consistency. This is achieved by weakening the consistency guarantees of KT, so that the protocol only ensures that split-views are detected by *either* the party who queries for a key, *or* the key owner.

1.1.2 Practical Deployments. The drawbacks of state-of-the-art academic proposals for Key Transparency consistency impact the security of existing KT systems. Indeed, none of the systems we consider currently provide a satisfactory consistency guarantee.

The consistency guarantees in *iMessage* come from gossip protocols [3], which due to the lack of proper OOB channels give weak security guarantees. *Zoom* state that they "will partner with independent external auditors" [6] for ensuring the consistency guarantee, *i.e.* , they envision a future use of designated witnesses but lack a solution at present. *WhatsApp*'s approach is to use an S3 bucket with a 5-year retention period [33]. This means that consistency *depends on a central third party*, which is clearly undesirable since KT is designed to avoid trust in third parties.

1.2 A Novel Approach to KT Consistency

As mentioned in Section 1.1.1, consistency comes trivially if one is willing to accept the costs and assumptions associated with blockchains [23, 34, 45], or more generally broadcast systems [24, 37].[1] This is because by design such systems enjoy, among others, two distinctive features:

(1) *Consistency*: each party can verify that they have a view of an ordered list of data that is identical to the view of all other parties,
(2) *Guaranteed Output Delivery (GOD)*: each party has guaranteed access to any update of the list at specific time intervals.

We notice that for KT feature (1) suffices, and (2) is superfluous. Hence one could attempt to dissect a blockchain as a broadcast system, dispose of any machinery related to GOD, and extract just what provides the consistency guarantees, in the hope of reducing costs and assumptions. Unfortunately, consistency and GOD are not easily separable – GOD enables consistency. Without guaranteed output delivery, there is no machinery left to ensure that the same value reaches all parties in the system, undermining consistency.

Since extracting a stand-alone mechanism for consistency is impossible, the next natural attempt is to design a novel mechanism that replaces GOD while supporting consistency. This is precisely the aim of gossip protocols and protocols relying on external committees, with the shortcomings for KT discussed in Section 1.1.1.

1.2.1 Contribution Summary. In this work, we follow a different path. We define GOD-less broadcast, which is a protocol in which each user can detect split view attacks by itself, and in such a case immediately abort the protocol (die). We refer to this property as "Consistency-or-Die" (CoD), and demonstrate that it can be implemented without assuming GOD, OOB channels, or a trusted committee of external reviewers. We further show that GOD-less broadcast is efficient in practice, even for large scale applications supporting a billion users. Our solution is both efficient and provides much better security guarantees over billions of users in both theory and practice compared to all previous works. Hence, GOD-less broadcast represents the most promising candidate for providing consistency for KT.

1.2.2 Overview of our Techniques. To guarantee consistency, we employ a mechanism similar to external committees where each member endorses its view, but with two significant novelties. The first novelty is that the committee is neither external nor fixed. Instead, for each epoch a fresh set of *random* and *initially undisclosed* users is selected from all users in the system. This allows rooting the distributed trust across the entire set of users rather than in a small external committee, which a powerful adversary could easily corrupt. The second novelty is that the data which constitutes the view, *e.g.* a commitment to the current state by the identity provider,

[1]Although blockchains are not formally broadcast systems, they do implicitly provide similar guarantees, and are often used as broadcast channels in the academic literature.

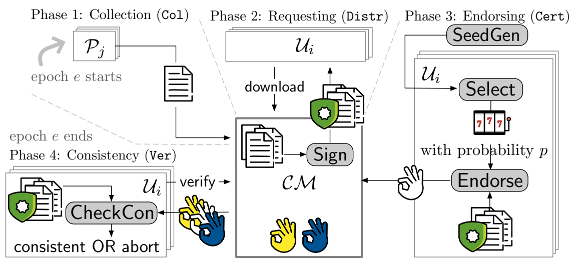


Figure 1: Illustration of core components and phases of GOD-less broadcast.

needs to be delivered *before the identities of the committee members become known*. This makes users tasked with endorsing a view *untargetable* for an adversary.

Figure 1 illustrates the core mechanisms of our approach. Time in the protocol proceeds in consecutive epochs. Each epoch is divided into four non-overlapping phases. In the first phase (Col), the channel maintainer $\mathcal{CM}$ collects data from publishing parties $\mathcal{P}_j$. In the second phase (Distr), $\mathcal{CM}$ organizes the collected data into what we call *the view* for the current epoch, and distributes the signed view as a reply to download queries from users $\mathcal{U}_i$. In the third phase (Cert), users independently run a mechanism for selecting a committee of endorsers that will certify the view of the epoch in a way that is cryptographically sound and does not reveal the identity of committee members until they publish an endorsement to the $\mathcal{CM}$ (using a hidden committee [15, 16, 23, 27], see Section 2). In our mechanism, users get a common seed and evaluate a Verifiable Random Function (VRF) (see Definition 2.1) on the seed – using their own secret VRF key. If the VRF output is below a fixed threshold value, they endorse the view they received in the previous phase by signing it and sending it to $\mathcal{CM}$ along with the VRF proof for public audit of their selection as the current epoch's endorsers. This mechanism is essential in our security analysis. In the fourth and final phase Ver, each user verifies the consistency of the view received in the second phase. This entails downloading a set of endorsements from the $\mathcal{CM}$, and verifying that it contains a *quorum* of legitimately selected endorsers for $\mathcal{U}_i$'s view. Setting the quorum value is another core part of our security analysis. If this consistency check fails, the user aborts and ceases to participate in the protocol for all future epochs, *i.e.* it "dies".

In contrast to gossip protocols, our approach enables immediate detection of split-view attacks, by the user itself, provides strong detection guarantees where *all* split-views are detected, and is compatible with the in-band communication pattern with an adversarial central party present in Key Transparency. Compared to external committee consistency protocols, it roots the consistency guarantee in the honest majority of the entire population of users (*i.e.* in billions of users in the case of WhatsApp) rather than a small set of targetable external auditors.

1.3 Contributions

Motivated by the lack of satisfactory solutions for consistency in KT, we propose a novel solution. Along the journey to our concrete construction, we make contributions at different levels.

(1) We introduce and formalize the notion of GOD-less broadcast (Section 3), i.e., a broadcast mechanism without the GOD property, but still complete and consistent.
(2) We provide a generic construction of a GOD-less broadcast protocol that makes black box use of standard building blocks: a PKI, a signature scheme and a hidden committee protocol (Section 4).
(3) We discuss how to efficiently instantiate our GOD-less broadcast protocol using well-known cryptographic schemes and provide estimate benchmarks for the heaviest procedures to argue feasibility of the overall construction.
(4) We provide an extensive security analysis of our GOD-less broadcast protocol. Notably existing security approaches in the field rely on the guaranteed output delivery property of broadcast channels and hence cannot be applied to our construction. Our analysis leverages results from combinatorics that have not been employed in previous work [4, 17] and reveals a concrete mechanism to identify optimal *quorum* values. A quorum is a concept introduced in this work related to preventing split view attacks.
(5) As a minor contribution, we also show how our novel security analysis can be applied to broadcast systems *with* guaranteed output delivery, and draw connections between split view attacks in the GOD-less setting and honest majority committees in broadcast systems with GOD.

2 PRELIMINARIES

VRFs A Verifiable Random Function [19] (VRF) allows a prover to, given a seed, output a value which is verifiably random to a verifyer. It is defined as in Definition 2.1.

Definition 2.1 (Verifiable Random Function). *A VRF consists of the following procedures:*
KeyGen$(\lambda) \rightarrow (\mathsf{sk}, \mathsf{pk})$*: This procedure on input a security parameter λ outputs a secret key* sk *and a public key* pk.
Prove$(\mathsf{sk}, s) \rightarrow (r, \pi_r)$*: This procedure on input a secret key* sk *and a seed s, outputs a value r and a proof π_r.*
Verify$(\mathsf{pk}, s, r, \pi_r) \rightarrow 0/1$*: This procedure, on input a public key* pk*, a seed s, a value r, and a proof π_r, outputs 1 if the proof verifies, and 0 otherwise.*

A VRF is secure if it has *uniqueness*, *i.e.* for each (pk, π_r) there is a single r which will verify, it has *provability*, *i.e.* a correctly produced r and π_r will always verify, and it has *pseudorandomness*, *i.e.* a computationally bounded adversary has a negligible advantage in distinguishing r from randomness. For formal definitions of these properties we refer to [19].

The above properties which are the common ones required for a VRF however do not account for malicious key generation, which if not accounted for can impact the output of the VRF to skew the distribution. In our scenario, we must account for this and therefore require a VRF with unpredictability under malicious key generation. Informally, this guarantees that regardless of how the public key was generated, the output of the VRF must be random. For formal definitions we refer to [16].

Hidden Committees Hidden committee protocols [15, 16, 23, 27] are protocols for selecting a subset of users from a universe of users in a way so that: (1) users are selected at random, (2) the identity of each selected user is anonymous to all parties except the user itself (including other selected users), and (3) any selected user can choose to reveal itself as selected in a verifiable way. Such protocols make it hard both for selected committee members to collude, and for adversaries to target committee members for corruption. Hidden committees have been used both in Proof-of-Stake [16, 23, 27] and in You Only Speak Once (YOSO) style protocols [4, 8, 9, 21]. In order to facilitate black box use of hidden committee schemes, we will use the following syntax.

Definition 2.2 (Hidden Committees). *A Hidden Committee selection protocol (*HC*) has the following procedures:*
KeyGen$(\lambda) \rightarrow (\mathsf{sk}, \mathsf{pk})$*: This procedure on input a security parameter* λ *outputs a secret key* sk *and a public key* pk *to the calling user.*
Select$(\mathsf{sk}, s, T) \rightarrow (\beta, y, \pi)$*: This procedure on input a secret key* sk*, the seed s, and a threshold for selection T, outputs values* β, y *and* π*, where* β *indicates whether* $y < T$*, and* π *is a proof that y was correctly computed from s and* sk*.*
Verify$(\mathsf{pk}, s, y, T, \pi) \rightarrow 0/1$*: This procedure on input a public key* pk*, a seed s, a value y, a threshold T, and a proof* π*, outputs 1 if* $y < T$ *and* π_i *is a valid proof with respect to s and* pk_i*. Else it outputs 0.*

Informally, a hidden committee selection scheme is correct if for all honestly generated $(\beta, y, \pi) \longleftarrow \mathsf{Select}(\mathsf{sk}, \mathsf{s}, T)$ it holds that $\mathsf{Verify}(\mathsf{pk}, \mathsf{s}, y, T, \pi) = \beta$ with probability 1.

Informally, we say that an HC scheme is secure if it is: (1) *anonymous*, *i.e.* that an adversary given a public key of an honest user and a seed has a negligible advantage in predicting whether the user will be selected, and (2) *unforgeable*, *i.e.* that an adversary cannot forge a proof of selection except with negligible probability.

3 INTRODUCING GOD-LESS BROADCAST

We introduce GOD-less broadcast: a lightweight version of a broadcast channel that can be realized in a concretely efficient way, and without relying on complex blockchain ecosystems. Efficiency comes by replacing the distributed system (typical in blockchains) with an *untrusted* central party (typical in transparency logs), and by relaxing some properties;

- GOD-less broadcast achieves the same *consistency* guarantees as broadcast,
- differently from broadcast, GOD-less broadcast makes no guarantees on message delivery, *i.e.* it provides neither censorship resistance nor guaranteed output delivery (as is the case in transparency logs),
- instead, GOD-less broadcast provides *abort* upon discovery of an inconsistent view (honest parties that discover an inconsistency will abort and become inactive).

3.1 Model

We now formally model GOD-less broadcast.

3.1.1 Entities, Roles, and States. A GOD-less broadcast system consists of the following entities (input/output processes):

- A channel maintainer $\mathcal{CM}$, which maintains a list of published messages, and answers queries for these messages.
- A set of $N \in \mathbb{N}$ users, each user denoted $\mathcal{U}_i$, which receives broadcast messages via $\mathcal{CM}$. The set of users is allowed to evolve over time.
- A set of publishers, each publisher denoted $\mathcal{P}_j$, which can broadcast messages via $\mathcal{CM}$. A publisher can be an external party, the $\mathcal{CM}$, or a user.

At specific points in time a number $n \ll N$ of users are assigned an e*ndorser* role. The role of an endorser is to ensure that $\mathcal{CM}$ presents a consistent view of published messages to all $\mathcal{U}_i$. At any point in time honest users are in one of *two possible states*; consistent, meaning that their view equals the one of all other users in the consistent state, or abort, meaning that the user discovered a discrepancy in their view (which may happen due to the GOD-less broadcast property) and should cease to participate in the protocol.

3.1.2 Architecture. GOD-less broadcast relies on a central maintainer. Publishers have interfaces to *send* to-be-broadcast messages to the maintainer. Users have interfaces to *receive* (query) the current view and endorsements from the maintainer, and to *endorse* a view (when acting as an endorser).

3.1.3 Infrastructure. Our protocol implementing a GOD-less broadcast system relies upon a public key infrastructure (PKI) that binds identities to keys. This means that the PKI stores a publicly accessible directory containing label-value records of the form (username, public key) for every party in the system. In our case, we will have a channel maintainer record $(\mathcal{CM}, pk_{\mathcal{CM}})$, and user records of the form $(\mathcal{U}_i, \mathsf{pk}_i)$. It is assumed that each user has a single identity key registered in the PKI. Keys are allowed to be updated.

3.1.4 Time, Epochs and Phases. GOD-less broadcast does not include any notion of time *per se*, as natively it is an asynchronous system. However, we do need a notion of time to decide whether a message is simply delayed or completely suppressed. We do so by using *epochs* as an abstract time-unit, and collect and distribute messages in epoch-batches.

Epochs progress in an incremental way via a nextep() mechanism. Each epoch is divided into four phases: *data collection, data distribution, data certification*, and *data verification.* The syntax of a GOD-less broadcast protocol follows these four phases. First, during the data collection phase Col, publishers submit data to the channel maintainer $\mathcal{CM}$ which collects these messages into a list M_e. Second, during the data distribution phase Distr, $\mathcal{CM}$ distributes M_e to users. Third, during the data certification phase Cert, a special subset of users act as endorsers by sending an endorsement of their view, which $\mathcal{CM}$ collects into E_e, Fourth and finally, during the data verification phase Ver, all users query for E_e and verify their view against the endorsements in E_e, so that all honest parties in the consistent state have the same view.

The system can tolerate a certain fraction of users which are offline in each epoch (in fact, there is no way of telling whether a user is offline or is targeted by adversarial message suppression). Users are thus not expected to maintain any state across epochs except for their own key pairs and some fixed public parameters. That is, any party can securely participate in epoch e without having obtained any of the messages sent during epoch $e - 1$.

3.1.5 Adversary. We consider a malicious adversary $\mathcal{A}$ which controls the communication network. In particular, $\mathcal{A}$ intercepts all protocol messages and can arbitrarily suppress messages to and from $\mathcal{CM}$. In addition, the adversary can corrupt up to a fraction $f < \frac{1}{3}$ of active users in the system in an adaptive way. Corruption needs to happen within the first two phases of each epoch.

3.2 Syntax

Definition 3.1 provides the syntax of a GOD-less broadcast protocol. To reflect the nature of most centralized systems, all interactive procedures are initiated by a user or publisher. If one wishes, it is trivial to change the model so that the central party initiates procedures. GOD-less broadcast relies on a few external resources;

- a **public key infrastructure** (PKI) that collects the public keys needed in the system,
- a **mechanism to advance epochs** nextep(), e.g., one can think that epochs are advanced every 20 seconds,
- a **mechanism to advance phases within an epoch** nextph(), e.g., phases last 5 seconds,
- and a **hidden committee protocol** HC.

Definition 3.1 (GBC). *A GOD-less broadcast protocol (GBC) is defined for a channel maintainer $\mathcal{CM}$, a set of users $\mathcal{U}_1, \dots, \mathcal{U}_N$, and a set of publishers $\mathcal{P}_1, \dots, \mathcal{P}_m$. In each epoch $\mathcal{CM}$ has an initially empty list of published messages M_e, and an initially empty endorsement list E_e. GBC is defined by the following procedures.*

Setup$(1^\lambda) \to (\mathsf{pp})$*: The setup procedure takes as input a security parameter λ and the PKI information. It outputs public parameters* pp *that are implicit input to all subsequent algorithms. The public parameters include at least; the description of all public parameters of the external resources; a starting epoch counter $e = 0$; an initial phase* `Col`*; a threshold T for selection of endorsers (by* HC*); and a set of user identifiers $\mathcal{U}_i$ and publisher identifiers $\mathcal{P}_i$.*

Send $= \langle \mathcal{P}_j(data, e); \mathcal{CM}(\mathsf{M}_{e'}, e') \rangle$*: This interactive procedure can be run multiple times during phase* `Col`*, and is initiated by a publisher $\mathcal{P}_j$, who sends its epoch value e and data for publication to $\mathcal{CM}$. The $\mathcal{CM}$ compares the user's epoch counter e with its own e'. If equal, $\mathcal{CM}$ accepts, stores the data for publication in $\mathsf{M}_{e'}$ and returns* "ok" *to $\mathcal{P}_j$.*

Get $= \langle \mathcal{U}_i(e); \mathcal{CM}(\mathsf{sk}_{\mathcal{CM}}, \mathsf{M}_{e'}, e') \rangle$*: This interactive procedure is initiated by every active user in the system, who during* `Distr` *sends its epoch e to $\mathcal{CM}$. If $e = e'$, $\mathcal{CM}$ computes a token $\sigma_{e'}$ (e.g. a signature) which binds it to $\mathsf{M}_{e'}$ and sends $(\mathsf{M}_{e'}, \sigma_{e'})$ to $\mathcal{U}_i$.*

Endorse $= \langle \mathcal{U}_i(\mathsf{sk}_i, \mathsf{M}_e, \sigma_e, e); \mathcal{CM}(\mathsf{E}_{e'}, \mathsf{M}_{e'}, e') \rangle$*: In phase* `Cert`*, this interactive procedure is initiated once by every active user, who holds its secret key sk_i, its channel view M_e, a binding token σ_e (obtained during the* `Distr` *phase), and its epoch counter. The user first checks if it is an endorser in epoch e (via* HC*), and if so sends an endorsement of its view to $\mathcal{CM}$. $\mathcal{CM}$ verifies if $e = e'$ and if the user is an endorser. If true it stores the endorsement in $\mathsf{E}_{e'}$.*

CheckCon $= \langle \mathcal{U}_\mathsf{i}(\mathsf{M_e}, \sigma_\mathsf{e}, \mathsf{e}); \mathcal{CM}(\mathsf{E}_{\mathsf{e}'}, \mathsf{e}') \rangle$*: In phase* `Ver`*, this interactive procedure is initiated by every user, who holds its channel view M_e and token σ_e (obtained during the* `Distr` *phase) for epoch e. $\mathcal{U}_i$ requests the endorsements for epoch e from $\mathcal{CM}$. If $e = e'$, $\mathcal{CM}$ replies with the list of endorsements $\mathsf{E}_{e'}$. Upon receiving $\mathsf{E}_{e'}$, $\mathcal{U}_i$ checks its validity. If the check fails $\mathcal{U}_i$ sets it state to abort and stops participating in the protocol, else $\mathcal{U}_i$ considers M_e valid.*

3.3 Properties

GOD-less broadcast enjoys *completeness* and *consistency.*

Completeness ensures that *if parties involved in the protocol behave honestly, the verification procedure will leave users in a consistent state.* This property holds for every epoch, under the assumption that each procedure is executed solely in the designated phase, and there is no temporal overlap between phases.

Consistency, on the other hand, guarantees that there exists a *unique* view which is shared among all honest users in the consistent state. We formalize this security property as follows:

Definition 3.2 (Consistency). *For any 2 distinct honest users $\mathcal{U}_i$ and $\mathcal{U}_j$, $i \neq j$, the event where $\mathcal{U}_i$ holds view M_e, $\mathcal{U}_j$ holds view M^*_e, $\mathsf{M}^*_e \neq \mathsf{M}_e$, and both users end up in the consistent state at the end of epoch e, has a negligible probability of happening.*

This property holds for every epoch, under the assumptions that each procedure is executed solely in the designated phase, there is no temporal overlap between phases, and that the building blocks used in GBC are secure.

4 CONSTRUCTING GOD-LESS BROADCAST

In this section we show how to construct a secure GBCfrom black box use of a few building blocks. Our security analysis is generic and relies on the properties of the different building blocks.

4.1 Building Blocks

We make black-box use of the standard building blocks of a PKI, a signature scheme, and a hidden committee protocol.

4.1.1 SeedGen. We also rely on black box use of a function $\mathsf{s}_e \leftarrow$ SeedGen(e) which takes an epoch counter e as input and outputs the seed s_e of the epoch. A SeedGen is correct if all honest parties in the consistent state receive an identical output, except with a negligible probability. It is secure if it is *unpredictable*, meaning that an adversary has a negligible advantage in predicting its output s_e before the time $e - \delta$ (e being the current epoch and δ being a number of epochs or phases). The period between $e - \delta$ and e is referred to as the leaky period. There are multiple ways to realize such a function, which we detail in Section 4.5.

4.2 Our GBC Protocol

Our GBC protocol is presented in Protocol 1. In addition to the security parameter, the protocol takes as input a threshold, T, that sets the probability for a user being selected as an endorser. The protocol also takes as input the quorum size k which is a positive integer cutoff value that sets the number of needed endorsements for a view to be accepted.

4.2.1 Assumptions. In this section, we will assume that T and k are set to secure values. How to securely set them is discussed in depth in Section 5.

To save on notation, Protocol 1 makes no explicit calls to obtain public keys from the PKI, and instead assumes that they are readily available from the PKI. It is also assumed that public keys which are updated in the PKI are not considered valid until after a time equivalent to the leaky period of SeedGen. Functionality for efficiently checking when a key in the PKI was updated is present in PKIs with append-only properties, which is the case in our intended use case of KT (see further details in Section 6.1).

We assume a one time setup in which all the necessary public parameters are generated, all public keys of the initial users are collected in a PKI, all lists are initialized to $\emptyset$ (at epoch $e = 0$), and every user $\mathcal{U}_i$ is set in a consistent state.

4.2.2 Protocol Description. To keep in line with the design of messaging apps, all procedures are initiated by users or publishers who send requests or upload data to the channel maintainer $\mathcal{CM}$. The channel maintainer, in turn, checks that each request is for the current epoch before acting on it (*e.g.* in line 2 in the Send procedure). Each time a user initiates a procedure it starts with checking whether it is still in a consistent state, or if it has detected a split view and entered the abort state and shall thus not participate in the protocol any further.

The Send procedure takes place during the `Col` phase and can be initiated by a publisher who wishes to upload data for publication. The $\mathcal{CM}$ stores any such data it receives.

The rest of the procedures must be initiated by all honest active users in each epoch.

The Get procedure takes place during the `Distr` phase. Here, all active honest users request the data published during the current epoch from $\mathcal{CM}$ (line 3). $\mathcal{CM}$ in turn constructs a token in the form of a signature (line 5), and responds each request with the data and the token (line 6). The purpose of the token is to allow users who later are selected as endorsers to prove that their view was actually given by the $\mathcal{CM}$ and not falsified by a corrupt endorser. It also allows non-selected users to efficiently compare their views with endorsed views. Note that at this point the consistency of the published data is not yet audited and thus it should not be trusted.

The Endorse procedure takes place during the `Cert` phase. Here all users first check if they are selected as endorsers (line 3 and 4). If not (line 5), they end the procedure for this epoch. If they are selected, they create an endorsement of their view using their secret key and the token obtained from $\mathcal{CM}$ in the Get procedure (line 8 and 9), and send their endorsement and proof of being selected as a witness to the $\mathcal{CM}$ (line 10). When the $\mathcal{CM}$ obtains the message, it checks if it comes from a valid endorser in the current epoch (line 12 and 13), and in that case stores the endorsement (line 14).

The CheckCon procedure takes place during the `Ver` phase. Here all users verify whether they have a consistent view of the data sent over GBC for the current epoch. Each user requests the endorsements for this epoch from $\mathcal{CM}$ (line 3), who responds with all endorsements it has received (line 5). Each user then filters out any endorsements which are not from a valid endorser of this epoch (lines 8-10). It then checks if the set of endorsements from valid endorsers is sufficiently large (line 12), and if all the endorsements agree with the view of the user (13). If true, the user sets its state to consistent. Otherwise it has detected a split view or a failure in obtaining sufficiently many valid endorsements. In either of those cases, the user sets its state to abort and ceases to participate for all future epochs.

4.3 Analysis of Protocol 1

4.3.1 Completeness. An honest $\mathcal{CM}$ distributes the same view $\mathsf{M}_{e'}$ with the same signature $\sigma_{e'}$ to all parties. Hence, all honest

Protocol 1 – Our GOD-less broadcast Protocol (GBC) from: SeedGen, PKI, Sig, HC, $k \in \mathbb{N}$.

Entities: $\mathcal{CM}$ channel maintainer; $\mathcal{P}_j$ publishers; $\mathcal{U}_i$ users.
Notation: $\mathsf{sk}.X_i$ is $\mathcal{U}_i$'s secret key for protocol X (sk_i contains all $\mathsf{sk}.X_i$'s).

```
Send = ⟨$\mathcal{P}_j(\mathsf{data}, e); \mathcal{CM}(\mathsf{M}_{e'}, e')$⟩                    Col
1: $\mathcal{P}_j$ sends $(\mathsf{send}, \mathsf{data}, e)$ to $\mathcal{CM}$.
2: if $e = e'$ then:
3:     $\mathcal{CM}$ appends data to $\mathsf{M}_{e'}$ and sends "ok" to $\mathcal{P}_j$.

Get = ⟨$\mathcal{U}_i(e); \mathcal{CM}(\mathsf{sk}_{\mathcal{CM}}, \mathsf{M}_{e'}, e')$⟩                    Distr
1: if $\mathcal{U}_i$ is in the abort state then
2:     end procedure here.
3: $\mathcal{U}_i$ sends $(\mathsf{download}, e)$ to $\mathcal{CM}$.
4: if $e = e'$ then
5:     $\mathcal{CM}$ computes $\sigma_{e'} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk.Sig}_{\mathcal{CM}}, \mathsf{M}_{e'}, e')$
6:     $\mathcal{CM}$ sends $(\mathsf{view}, \mathsf{M}_{e'}, \sigma_{e'})$ to $\mathcal{U}_i$.
7: Upon receiving $(\mathsf{view}, \mathsf{M}_{e'}, \sigma_{e'})$ $\mathcal{U}_i$ checks:
8: if $0 = \mathsf{Sig.Verify}(\mathsf{pk}_{\mathcal{CM}}, (\mathsf{M}_{e'}, e), \sigma_{e'})$ then
9:     $\mathcal{U}_i$ sets its state to abort.

Endorse = ⟨$\mathcal{U}_i(\mathsf{sk}_i, \mathsf{M}_e, \sigma_e, e); \mathcal{CM}(\mathsf{E}_{e'}, \mathsf{M}_{e'}, e')$⟩:                    Cert
1: if $\mathcal{U}_i$ is in abort state then
2:     $\mathcal{U}_i$ ends here.
3: $\mathcal{U}_i$ obtains $\mathsf{s}_e \leftarrow \mathsf{SeedGen}(e)$
4: $\mathcal{U}_i$ computes $(\beta, y_i, \pi_i) \leftarrow \mathsf{HC.Select}(\mathsf{sk.HC}_i, \mathsf{s}_e)$
5: if $\beta = 0$ then
6:     $\mathcal{U}_i$ ends here.
7: else                    ($\mathcal{U}_i$ has endorser role in epoch $e$)
8:     $\mathcal{U}_i$ sets $\mathsf{view}_i \leftarrow (e, \mathsf{M}_e, \sigma_e)$
9:     $\mathcal{U}_i$ lets $\epsilon_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk.Sig}_i, \mathsf{view}_i)$
10:    $\mathcal{U}_i$ sends $(\mathsf{endorse}, w_i = (y_i, \pi_i, \epsilon_i, e))$ to $\mathcal{CM}$.
11: On input $(\mathsf{endorse}, w_i)$, $\mathcal{CM}$ parses $w_i$ as $(y_i, \pi_i, \epsilon_i, e)$
12: $\mathcal{CM}$ obtains $\mathsf{s}_{e'} \leftarrow \mathsf{SeedGen}(e')$
13: if $e = e' \wedge 1 = \mathsf{HC.Verify}(\mathsf{pk.HC}_i, \mathsf{s}_{e'}, y_i, T, \pi_i)$ then
14:     $\mathcal{CM}$ lets $\mathsf{E}_{e'} = \mathsf{E}_{e'} \cup \{(y_i, \pi_i, \epsilon_i)\}$.

CheckCon = ⟨$\mathcal{U}_i(\mathsf{M}_e, \sigma_e, e); \mathcal{CM}(\mathsf{E}_{e'}, e')$⟩:                    Ver
1: if $\mathcal{U}_i$ is in abort state then
2:     $\mathcal{U}_i$ ends here.
3: $\mathcal{U}_i$ sends $(\mathsf{verify}, e)$ to $\mathcal{CM}$.
4: if $e = e'$ then
5:     $\mathcal{CM}$ sends $(\mathsf{verify}, \mathsf{E}_{e'})$ to $\mathcal{U}_i$.
6: Upon receiving $(\mathsf{verify}, \mathsf{E}_{e'})$, $\mathcal{U}_i$ lets $V \leftarrow \emptyset$
7: $\mathcal{U}_i$ obtains $\mathsf{s}_e \leftarrow \mathsf{SeedGen}(e)$
8: for $w_j = (y_j, \pi_j, \epsilon_j) \in E_{e'}$ do
9:     if $1 = \mathsf{HC.Verify}(\mathsf{pk.HC}_j, \mathsf{s}_e, y_j, T, \pi_j)$ then
10:        $\mathcal{U}_i$ lets $V \leftarrow V \cup \{\mathsf{pk.Sig}_j, \epsilon_j\}$
11: $\mathcal{U}_i$ sets $\mathsf{view} \leftarrow (e, \mathsf{M}_e, \sigma_e)$
12: if $(|V| \geq k)$ then                    (Quorum Check)
13:     if $\forall\ (\mathsf{pk}_i, \epsilon_i) \in V$, $1 = \mathsf{Sig.Verify}(\mathsf{pk}_i, \mathsf{view}, \epsilon_i)$ then
14:        $\mathcal{U}_i$ sets its state as consistent.
15: else
16:     $\mathcal{U}_i$ sets its state to abort.
```

parties share the same, consistent view. Since SeedGen and HC are correct, an honest $\mathcal{CM}$ will accept all endorsements from honest endorsers during the `Cert` phase. Recall that the quorum parameter k is set so that $\mathcal{CM}$ will collect at least k endorsements of the same view (more details on this in Section 5). Moreover, when the set of endorsements is sent to a user in the `Ver` phase, each endorsing signature verifies (due to Sig being correct, and the endorsements coming from honest parties). This means that, at each epoch, all honest users that start in a consistent state are left in a consistent state, proving the protocol complete.

4.3.2 Consistency: Recall that by definition a GOD-less broadcast protocol is consistent if at each epoch, there is a negligible probability that two honest users –who start in a consistent state– end up in the state consistent after the `Cert` phase, while holding two distinct views $\mathsf{M}_e^* \neq \mathsf{M}_e$. For conciseness we refer to this condition as *the split view attack*. This property should hold under the presence of an adaptive adversary $\mathcal{A}$ which can drop and inject messages, and can corrupt a fraction of the parties, including the channel maintainer $\mathcal{CM}$. In what follows we argue that the split view attack can only occur with negligible probability.

Assume the setting of the split view attack and denote the two honest users as $\mathcal{U}$ and $\widehat{\mathcal{U}}$, and their views obtained during the `Distr` phase as (M_e, σ_e) and $(\widehat{\mathsf{M}}_e, \widehat{\sigma}_e)$ respectively. After the completion of the `Ver` phase, $\mathcal{U}$ and $\widehat{\mathcal{U}}$ are both in a consistent state and $\mathsf{M}_e \neq \widehat{\mathsf{M}}_e$

We distinguish two cases depending on whether or not $\mathcal{A}$ corrupts the channel maintainer.

The case of an *honest* $\mathcal{CM}$: If $\mathcal{CM}$ is honest, at each epoch, it distributes one consistent signed view (M_e, σ_e). Hence the split view attack happens only if the adversary drops the messages from the channel maintainer and injects one or more alternative views. However, since $\mathcal{CM}$ is honest, $\mathcal{A}$ would need to successfully forge a valid signature $\widehat{\sigma}_e$ for an alternative view $\widehat{\mathsf{M}}_e$, which contradicts the security assumption on the signature scheme Sig.

The case of a *corrupt* $\mathcal{CM}$: This case is more delicate, since now $\mathcal{A}$ can produce several views. We consider here an adversary that corrupts parties *before* the `Cert` phase of each epoch (we discuss how to remove this simplifying assumption in 4.3.3).

First, we observe that $\mathcal{A}$ cannot learn the identity of honest endorsers and thus cannot target them for corruption. This is due to the fact that $\mathcal{A}$ needs to corrupt before it sees endorse messages (line 10 in Endorse), and to the fact that given only users' public keys and the seed, $\mathcal{A}$ cannot guess who will have the e*ndorser* role in the current epoch, since otherwise this will break the anonymity property of the hidden committee scheme. Second, we observe that for an unpredictable SeedGen, and since HC selection is unpredictable (a predictable selection would contradict the anonymity property), the adversary has no advantage in adversarially generating the public keys to be selected by HC. These observations guarantee that any one corrupt party has the same probability of being selected as endorser as any one honest party.

Since $\mathcal{CM}$ is corrupt, it has the power to split the universe of users into multiple non-intersecting subsets, and distribute a different view to each subset. Notice that while honest endorsers in each subset will only endorse the signed view they received, each *corrupt* endorser can send endorsements of *multiple* views to $\mathcal{CM}$. A corrupt $\mathcal{CM}$ can then choose which of the multiple endorsements for different views from a corrupt endorser it should forward to different subsets of users to fulfill the verification checks in the CheckCon procedure. To prevent such an attack from succeeding, *i.e.* preventing the situation where two honest users accepts different views, in the `Ver` phase, users perform a *quorum check* (lines 12, 13, and 9 in the CheckCon procedure). There are three essential ingredients in this check; (1) the cutoff value k, (2) valid signatures/endorsements, and (3) legitimate endorsers. Legitimate selection of endorsers is guaranteed by the properties of the HC scheme, since unforgeability ensures that no corrupt user can fake being selected as an endorser, and anonymity ensures $\mathcal{A}$ cannot target honest endorsers for corruption. Since endorsements are signatures, valid endorsements are guaranteed by the unforgeability of the signature scheme. Security from split view attacks is then guaranteed when the cutoff parameter k is set so that, given the fraction of corrupt users, the probability that the number of honest endorsers in a subset (honest endorsers with the same view as a "victim") plus the number of corrupt endorsers selected by the HC.Select functionality does not reach k (except with extremely small probability).

Clearly the most successful way to implement the split view attack is to divide the users into two sets and dispatch (M_e, σ_e) to the first bucket and $(\widehat{\mathsf{M}}_e, \widehat{\sigma}_e)$ to the second. Let M denote the adversary's corruption budget, i.e., $\mathcal{A}$ can corrupt M users (in addition to $\mathcal{CM}$), and let H denote the number of total honest active users in the system. We provide a detailed mathematical analysis of how to set k in Section 5. The outcome of the analysis is that for any $M < H$, it is possible to find a quorum value k for which the probability of a split view attack is negligible, and at the same time, the probability of liveness (to have at least k endorsers of one single view) is overwhelming. This forces even a corrupt $\mathcal{CM}$ to dispatch a single consistent view across all users.

4.3.3 Dealing With a Rushing Adversary. In Protocol 1, the $\mathcal{CM}$ gets to see endorsements before passing them on to other users (line 10 of Endorse). If $\mathcal{CM}$ is corrupt, this reveals the identities of endorsers to $\mathcal{A}$. An adaptive adversary could then *rush* to corrupt the honest endorsers before passing on the endorsements, and instead have the (now corrupt) endorsers sign any view using their signing keys. A small quorum value $k \ll M$ renders the scheme vulnerable to an attack where $\mathcal{A}$ can corrupt sufficiently many "hidden" endorsers at multiple epochs without violating any assumptions of the corruption ratio. In order to solve this issue, we adopt the strategy of Ouroboros Praos [16], where forward secure signatures are employed to thwart this exact attack. Employ a *key evolving signature scheme* Sig in Protocol 1 and assume *secure erasures.* In an evolving signature scheme, each signature is generated with a signing key corresponding to a given time slot, which can be "evolved" to obtain the signing key for the next time slot. Signatures are also verified with respect to a specific time slot. In this setting, parties sign their view using the key evolving signature scheme instead of a standard EUF-CMA secure signature scheme. Before sending their message, the parties evolve their signing key and securely erase the previous signing key, achieving forward security. Parties who are not in the current committee simply evolve their signing keys and securely erase the previous signing key. Even if the $\mathcal{A}$ immediately corrupts all parties in a committee when they make themselves known (with the endorse message in line 10 of Endorse), it cannot sign a different view using the signing key corresponding to the previous time slot, since it has been securely erased. Hence, a rushing adaptive adversary cannot affect the security of our protocol.

4.4 Realizing HC

We present an efficient and scalable instantiation of a hidden committee protocol that can be used in our GOD-less broadcast construction. Our approach is inspired by the cryptographic sortition techniques in *e.g.* [4, 16, 23], and describes a generic way to build HC from any strong VRF. Next to the generic construction, we present a concrete and efficient realization that employs the VRF with unpredictability under malicious key generation from [16] (see Protocol 2). Since the VRF is publicly verifiable, committee members can convince anyone of their role at any later point in time (in particular *after* endorsing a view).

Protocol 2 works as follows. Let T be a public parameter of the scheme denoting a public integer threshold value (that will influence the probability of being selected as endorser). Given as public input the seed s, and as private input the user's secret key sk_i, our selection process requires user $\mathcal{U}_i$ to compute $\mathsf{VRF.Prove}(\mathsf{sk}_i, \mathsf{s}) \to (y, \pi)$. Interpret y as an integer, if this value is below the public threshold T, the user $\mathcal{U}_i$ is in the committee of endorsers.

Note that π is a publicly verifiable proof that y is the output of the VRF evaluated by $\mathcal{U}_i$ (the holder of sk_i) on the public input s. Hence a user can keep private the fact that it is selected as part of the committee, but prove that they were selected when it is time to act according with a task given to the committee.

Protocol 2 – HC Protocol (instantiation with [16])

```
public parameters: threshold $T$, seed s
        (two hash functions $\mathsf{H} : \{0,1\}^* \to \mathbb{G}, \mathsf{H}' : \{0,1\}^* \to \mathbb{G}$)

HC.KeyGen($\lambda$):
1: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VRF.KeyGen}(\lambda)$        $(\mathsf{sk} \leftarrow \mathbb{Z}_q, \mathsf{pk} = g^{\mathsf{sk}})$

HC.Select(sk, s):
1: $\mathsf{VRF.Prove}_{\mathsf{sk}}(\mathsf{s}) \to (y, \pi)$
        $( y = (r, u)$ where $u = \mathsf{H}'(\mathsf{s})^{\mathsf{sk}}, r = \mathsf{H}(\mathsf{s}, u); \pi = (u, \pi')$
        where $\pi' = \mathsf{DLEQ.Prove}(log_{\mathsf{H}'(\mathsf{s})}(u) = log_g(\mathsf{pk})))$
2: if $y < T$ then        $(Map2int(r) < T, Map2int : \mathbb{G} \to \mathbb{Z}_q)$
3:     $\beta = 1$ (and set current role to $endorser$)
4: else $\beta = 0$
5: return $(\beta, y, \pi)$

HC.Verify(pk, s, $y$, $\pi$):
1: Compute $\mathsf{VRF.Verify}_{\mathsf{pk}}(\mathsf{s}, y, \pi) \to \beta$
   $( y \stackrel{?}{=} (\mathsf{H}(\mathsf{s}, u), u) \land \mathsf{DLEQ.Verify}(\mathsf{pk}, \mathsf{s}, \pi) \stackrel{?}{=} 1 \land Map2int(r) \stackrel{?}{<} T )$
2: return $\beta$.
```

4.5 Realizing SeedGen

As defined in Section 4.1.1, a secure SeedGen needs to be unpredictable. Without an unpredictable seed, an adversary could target a (any) given future epoch and use the time until that epoch arrives to pre-compute key pairs that are selected by the HC for this seed. This would undermine the quorum check of Protocol 1, since the adversary could then compute sufficiently many adversarial keys to form a quorum. Seed unpredictability prevents such adversarial key generation attacks by making sure that the adversary gets no (zero) time for pre-computation. Thus, keys in the PKI can only be trusted for HC selection based on the seed s_e, if the keys were registered in the PKI before the leaky period for SeedGen's computation of s_e began. That is, in a secure GBC *the seed that selects an endorser must not be older than an endorser's public key* (which is assumed in Section 4.2.1). With this in mind, we now give realizations of SeedGen under different assumptions, first by assuming GOD in a pre-existing external system, then internally realized without GOD but with simplifying assumptions, and finally the general case assuming neither GOD nor other simplifications.

4.5.1 Using an External Randomness Beacon with GOD. One way of realizing SeedGen is to use an external randomness beacon [11] which provides a fresh random nonce in each epoch. Such protocols allow for any parties to publicly verify that each output has been generated correctly, ensuring that the output is unbiased and unpredictable. In order to achieve such strong properties, randomness beacons generally require standard broadcast with GOD, which we want to avoid. However, notice that randomness beacons are readily available in a number of Proof-of-Stake based blockchain protocols that intrinsically execute them as sub-protocols, *e.g.* [27]. Hence, instead of executing a randomness beacon protocol, parties can rely on an existing beacon and leverage its public verifiability properties to validate its random outputs relative to each epoch.

4.5.2 Using an Internal Beacon Without GOD (With Simplifying Assumptions). Let us now describe how to realize SeedGen via the bounded bias beacon of Ouroboros Praos [16], without relying on GOD for security. We refer to this realization as $\mathsf{SeedGen}_{\mathsf{Praos}}(e)$. In the model of [16], the adversary is allowed to reset the beacon (resample randomness) a bounded number of times during the leaky period which is a specific time interval before epoch e. Such a beacon can be realized internally in GBC by having every user in the GBC protocol provide not only an endorsement signature ϵ_i and a VRF output/proof pair (y_i, π_i), but also an extra VRF output/proof pair (s_i, π'_i), obtained by each party in the endorser committee by evaluating the VRF on the seed s_{e-1} from epoch $e-1$. When computing $\mathsf{SeedGen}_{\mathsf{Praos}}(e)$, the seed s_e is derived by verifying all pairs (s_i, π'_i) from epoch $e-1$, and computing $\mathsf{s}_e = H(s_1|\ldots|s_n)$ from valid VRF outputs $s_1, \ldots, s_n$, where H is modeled as a random oracle.

Provided that the VRF key pairs are fixed in the PKI before the randomness generation starts, we can follow the analysis from [16]. If we make the simplifying assumption that keys are fixed in the PKI (no updated or added keys throughout the protocol), then the adversary can only introduce a bounded amount of bias in s_e by selectively adding or removing pairs s_i, π'_i from the view of honest users (*i.e.* refusing to deliver pairs generated by honest parties or refusing to generate pairs that should have been generated by corrupted parties). In other words, the committee election for epoch e uses randomness s_e generated from VRF outputs from epoch $e-1$ and elects committee members whose keys were already registered in the PKI before epoch $e - 1$ started. Thus s_e is unpredictable if *at least one* of the s_i's used to compute the SeedGen was delivered by an honest party, which can be statistically guaranteed with high probability by choosing an appropriate committee size.

4.5.3 General Construction (Without Simplifying Assumptions). We now provide a general realization $\text{SeedGen}_{\text{Gen}}$ that neither relies on GOD nor assumes fixed keys in the PKI.

In the above realization, $\text{SeedGen}_{\text{Praos}}$, the seed s_e is computed from the VRF outputs $s_1, \ldots, s_n$ from epoch $e - 1$, with proofs $\pi'_1, \ldots, \pi'_n$ that are verified against the seed s_{e-1}. The seed s_{e-1} has, in turn, been constructed from VRF outputs from epoch $e - 2$, with proofs that are verified against the seed s_{e-2}, and so on. When keys are allowed to be updated in the PKI, this chaining of seeds becomes a problem for users which resume after being offline. Resuming users can not rely on the unpredictability of such a chain of seeds to guard against adversarial key generation, because they were asleep when the seeds were fresh. Thus, resuming users do not know how old the seeds in the chain are – they can even be older than when these users were last online. Securely evaluating $\text{SeedGen}_{\text{Praos}}(e)$ requires that keys used in the evaluation are fixed when randomness generation starts. Randomness generation in $\text{SeedGen}_{\text{Praos}}(e)$ can start as soon as the seed is known. However, without any guarantees for seed freshness, there is no way to set a point in time for when to fix keys.

A heuristic for meeting the key fixation requirement in our case is for a resuming user evaluating $\text{SeedGen}_{\text{Praos}}(e)$ to make sure that the seed used for evaluating the s_i's is not older than when the user was last online (and ignore any keys updated after that time). In order to solve this, we can define $\text{SeedGen}_{\text{Gen}}(e)$ as $H(\mathsf{s}_{e-1}, \text{SeedGen}_{\text{Praos}}(e))$ and have a user compute s_{e-1} as follows. The user obtains s_k and $\text{SeedGen}_{\text{Praos}}(k)$ from the $\mathcal{CM}$ for all epochs $e - j < k < e$ for which the user was sleeping (two hashes per offline epoch). Then, it reconstructs the hash chain defined by $\text{SeedGen}_{\text{Gen}}(e)$ from s_{e-j} and $\text{SeedGen}_{\text{Praos}}(e - j)$ (when it was last online) up to s_{e-1}. Since the user knows that s_{e-j} was unpredictable for epoch $e - j$, and due to the preimage resistance of H, then s_{e-1} is by construction more recent than epoch $e - j$. No keys registered in the PKI before epoch $e - j$ could thus have been adversarially generated to be selected from s_{e-1}.

4.6 Implementation Efficiency Estimates

Protocol 1 employs only lightweight cryptographic tools. The heaviest computational tasks for users is in GBC.CheckCon, where they have to verify selection proofs (line 9) and verify endorsements (line 13), since for these tasks the CPU overhead scales linearly with the size of the quorum. We here provide an efficiency estimation of these parts when concretely instantiating selection proofs from a VRF, as in Protocol 2, and endorsements as ECDSA signatures.

Our main use case for GOD-less broadcast is consistency in key transparency, and thus we evaluate the performance on mobile phones. We have implemented the VRF of [16] in C using OpenSSL 1.1.1w. Internally this VRF depends on a hash, which we instantiate as SHA256, and on group operations, which we instantiate over the P256 curve. We compiled for iOS, and executed tests of VRF and ECDSA on the mid-tier phone iPhone 12 with iOS 17.3.1 installed. CPU time was measured using the mach_absolute_time function in <mach/mach_time.h>. This timer provides high resolution measurements (nanoseconds) of CPU ticks. The code of the implementation and tests is available at [1]. The results are available in Table 1.

To obtain the overhead of these operations in different scenarios, one can multiply the values in the sec/eval column in Table 1 with the quorum sizes from Figure 3b. Practical execution times are in the range of roughly a second to a minute, depending on system parameters. A refined discussion with concrete examples of execution times for different system parameters is given in Section 5.3.

If a forward secure signature scheme is chosen instead of ECDSA (for reasons explained in Section 4.3.3), the performance will be slightly worse for signature verification. This performance penalty is evaluated in [13], where the verification time of ECDSA based forward-secure signatures is roughly double to plain ECDSA.

	sec/eval	Time for 1,000 endorsements
ECDSA	0.000549738	0.55s
VRF	0.001450149	1.45s

Table 1: CPU speed for VRF and ECDSA verification.

5 HOW TO SET THE QUORUM SIZE

We now provide a new and improved statistical analysis, which is motivated by the fact that the stochastic nature of the sortition process makes it impossible for a system user to know the size of the endorsement committee. That is, in any epoch, the actual number of users that are selected as endorsers in that epoch is not a number that is revealed or can be made available to the system users. This makes it more involved to define security guarantees for an honest majority of endorsers or split view defense, further motivating our quorum concept. Compared to the analyses in, e.g., [4, 17], our statistical analysis is also more flexible regarding how to model passive users, which may be considered actively malicious or simply inactive depending on the situation.

The security analysis of our GOD-less broadcast protocol (Protocol 1) relies on having an opportune quorum size k that identifies the minimal number of endorsements – signatures for the same view generated by distinct users – which together guarantee, with high probability, the existence of a unique view across all users who are in a consistent state.

Our goal here is to develop a well-defined mechanism to compute k for any given GOD-less broadcast system given the following three parameters; (1) the probability of being selected as an endorser by a hidden committee protocol HC (denoted by p); (2) the total number of honest active users (denoted by H); and (3) the total number of malicious active users (denoted by M).

Let $0 < p < 1$ denote the probability that any one user is selected as an endorser. That is, p is the probability that any given user is selected for jury duty on the hidden committee of endorsers. Note that all users in the system are selected cryptographically at random (with probability p), in a publicly verifiable way, and independently from other users. This is done using the function HC.Select (employed for Endorse in GOD-less broadcast), which returns as part of its output a selection bit $\beta \in \{0, 1\}$ to indicate if a user is selected as an endorser. Intuitively this selection process simulates a coin flip that returns "heads" (output 1) with probability p. The selection bit β output by HC.Select is therefore accurately modeled as a Bernoulli random variable with parameter p. Since the user selection trials are independent of one another, the overall sortition process runs across the whole population of N users is distributed as a Binomial random variable $B(N, p)$.[2]

For the novelty in our analysis, we adapt a partition–and–cut methodology in which we first partition users into different classes, and then we exploit statistical properties of these classes to find a cutoff point that in some sense maximizes the differences between those classes of users.

Let us first partition the N users in the system into three subsets;

- H honest users,
- M malicious users,
- S silent (non-responsive) users,

so that we have $N = H + M + S$ with $H, M, S \geq 0$. Note that for the S silent users we do not care *why* they are silent. They can be actively suppressed by an adversary or they may simply be off-line and temporarily not interacting in the system.

This partitioning approach enables us to (conceptually) separate actively malicious users from users that are inactive in a given epoch (which can typically be a non-negligible fraction of the population when considering large scale, world-wide adopted systems) to provide more accurate security estimates.

At each epoch, we expect to have $H + M$ active users running HC.Select. We recall a known fact about Binomial distributions, which follows directly from Chu–Vandermonde identity: $B(H + M, p) = B(H, p) + B(M, p)$. This means that we can analyse the selection of active malicious users and of active honest users independently. In particular, $B(M, p)$ identifies a well-defined discrete probability distribution that is different, in shape, from $B(H, p)$ if $M \neq H$. In short, our analysis is based on this shape difference.

For the sake of understanding our analysis, it is easier to first consider the simplified case of selecting a committee that, with high probability, has an honest majority. This is what more intuitively corresponds to a quorum in the traditional setting where we have a channel *with* GOD. We present this in Section 5.1, and the reader may note that this part in itself also serves as a new and more accurate analysis of the setting in [4].

In Section 5.2 we will then make our analysis more involved for the GOD-less channel quorum case to provide high probability guarantees that the system is additionally protected against split view attacks, which is harder to accomplish since it requires more than just an honest majority in the endorsement committee.

5.1 Warmup: Quorum in Broadcast *with* GOD – Protection Relying on an Honest Majority

In GOD channels, everyone that speaks will he heard, so a quorum in this setting is a committee that, with high probability, has an honest majority.

Let us denote $Z \sim B(M, p)$ and $Y \sim B(H, p)$, so that Z and Y respectively correspond to the number of malicious and honest users that have been selected as endorsers. While the endorsement committee additionally consists of a number $W \sim B(S, p)$ of silent users, they are not active in the endorsement process and need not be considered further in this analysis.

The reader may note that Z and Y are not available[3] to the system users, so they do not know the size of the actual endorsement committee for the (or any) given epoch. Our analysis therefore focuses on finding a value $k \in \{1, \ldots, M\}$ such that both $\Pr(Z \geq k)$ and $\Pr(Y \leq k)$ are negligible. In other words, in our practical application we need a cutoff value which guarantees that, with high probability, the following two bad events do *not* occur:

A) there are k or more malicious endorsers, and
B) there are k or fewer honest endorsers on the committee.

Intuitively, we are thus comparing the right tail area of $B(M, p)$ with that of the left tail area of $B(H, p)$, and require both of them to be small. If such a cutoff value k exists, then the act of verifying k valid signatures on one and the same view provides a provable guarantee that more honest than malicious users endorse the view.

To see that such a k always exists, first note that the probability p is a parameter that directly determines the expected size of the endorsement committee. Given any fixed probability p, let the value $k' \in \{1, \ldots, M\}$ denote the smallest value i for which $\Pr(X \geq i) \leq \Pr(Y \leq i)$. It is clear that k' exists and is uniquely defined since $M < H$. Note that for k' we have $\Pr(X \geq k') \approx \Pr(Y \leq k')$, and $b' = -\log(\Pr(Y \leq k'))$ can be interpreted as the bit security level of the probability p, as $2^{-b'}$ is the probability of endorsement committee failure in terms of the bad events A or B above.

In order to find a suitable cutoff value k that is optimal in this metric[4] for any given bit security level b, we take the cutoff value k' corresponding to the minimal probability p (minimal expected committee size) with bit security level $b' \geq b$.

These computations are indeed practical, even for our extreme case target of systems with $N = 1,000,000,000$ users. We provide concrete values for realistic scenarios in Section 5.2.1, and we also provide code that can be used to compute these values [2].

5.2 Quorum in GOD-less broadcast – Protecting Against Split View Attacks

While our description above focused on explaining the statistical mechanisms of obtaining a committee with an honest majority, we now proceed to our target quorum case, which requires stronger guarantees. As discussed in Section 4.3, for the case of a malicious $\mathcal{CM}$, the adversary $\mathcal{A}$'s highest chance to succeed in a split view attack is to divide the honest users into two disjoint sets of equal size. This translates to having half of the honest endorsers endorsing the same view, regardless of which view the malicious users choose to endorse. In statistical terms, we thus need to compare $X \sim B(M + \frac{H}{2}, p)$ to $Y \sim B(H, p)$. The intuitive reasoning is identical to the explanation above, except that the peaks of the probability distributions are now closer together to reflect that safe-guarding against split view attacks is harder than ensuring an honest majority. However, the procedure for finding an optimal cutoff value k for a

[2]Recall that the Binomial random returns the number of successes in a sequence of N independent experiments, each with its own Boolean-valued outcome, and the probability of a successful output is p. The probability mass function of obtaining exactly k success is $\Pr(B(N, p) = k) = \binom{N}{k} p^k (1 - p)^{N-k}$.

[3]In the case where the system users have additional knowledge of the committee size, it is possible to further improve the analysis, but this is left for future work.

[4]Our analysis permits many alternative metrics to be used, in particular, our method is quite flexible in regard to the employed metric.

given bit security b is precisely the same, and the act of verifying k valid signatures on one and the same view now provides a provable guarantee of the consistency property of the KT log. And this k value is precisely our quorum size that we set out to determine, which completes our main goal for this part.

The probability distributions we are discussing closely resemble the ones plotted in Figure 2. The optimal cutoff value k' corresponds to a vertical line that separates the two bumps. The bit security level b intuitively corresponds to the log of the tail areas beyond the cutoff value k'. Figure 2 further illustrates how the probability distributions change shape as we modify the probability p (expected endorsement committee size).

From our statistical analysis it follows that the limit of efficiency is approached as the number of malicious users M approaches $\frac{H}{2}$. When $M \geq \frac{H}{2}$, there exists no hope of efficiently protecting the system against split view attacks, but for reasonable $M < \frac{H}{2}$, we can actually compute concrete values for relevant parameter sets.

5.2.1 *Some Optimal Quorum Size Computations.*

For some realistic parameter sets, consider a large scale application with one billion (1,000,000,000) users, where we allowing for a 20% portion of the users to be silent/non-responsive. In Figure 3 we present optimal quorum sizes for parameter sets with varying degrees of honesty and maliciousness in the remaining population.

5.3 About Grinding

We handle adversarial seed tampering for the cryptographic sortition similarly to [4]. This approach gives the adversary power to choose the seed. However, since epochs have time limitation, and the adversary is computationally bounded, only up to $T = 2^t$ seed resets are allowed. In our analysis, this corresponds to having T samplings from X. Since each sampling (new seed) corresponds to an independent event, the overall probability of succeeding in finding a seed that yields a larger committee is $\sum_{i=1}^{T} \Pr(X \geq k) = T \cdot \Pr(X \geq k)$. To make this latter probability negligible, it is enough to select a k for which $\Pr(X \geq k) = 2^{-(b+t)}$. In other words, using our methodology, it is possible to explicitly account for grinding by adjusting the metric to weight the respective tail areas of $B(M + \frac{H}{2}, p)$ and $B(H, p)$ accordingly.

5.3.1 *A Practical Perspective on Grinding vs. Epoch Duration.*

It is practically possible to eliminate the grinding attack vector by setting a bit security level of, say, 256 bits as per Figure 3b. However, this gives slightly larger quorum sizes than what an optimized approach would give.

Higher grinding tolerance implies larger quorums, which, in turn, implies longer verification times. However, let us put these things into perspective with a concrete imaginable use-case.

If we consider the worst case highly malicious population as given in Figure 3, setting cryptographic level strength b to 256 bits does avoid the grinding problem entirely, but the resulting quorum size implies about 80 seconds of processing time for verification. While this can be considered an upper bound estimate on the amount of processing time that is needed for view verifications on a device, it is still very reasonable in practical applications that use, say, one-day epochs. And do recall that the scale that we consider here is a population size of 1,000,000,000 users/devices.

Another way of looking at it is to shorten the epoch times to, say, 1 minute. Then grinding is much less of an issue and b can be set lower, e.g. $b = 80$. The revised worst case estimate for the required processing time for per-device view verification is then reduced to 24 seconds – clearly below a target epoch duration of 1 minute.

While these crude estimates can be considered very conservative, they illustrate that our techniques are not limiting in practice, not even for large scale applications. And furthermore, more optimized approaches can give even better performance guarantees in practice, for example, as we suggested in Section 5.3, by recomputing the optimal quorum sizes using different metrics that explicitly include a grinding factor that is suitable for the target use-case application.

6 DISCUSSION & CONCLUSION

6.1 Key Transparency from GOD-less broadcast

We here give an overview for how a KT protocol which provides both append-only and consistency guarantees can be straightforwardly constructed from black-box use of a Verifiable Key Directory [10, 30] (VKD) and our GOD-less broadcast protocol GBC. Such a KT protocol works a follows.

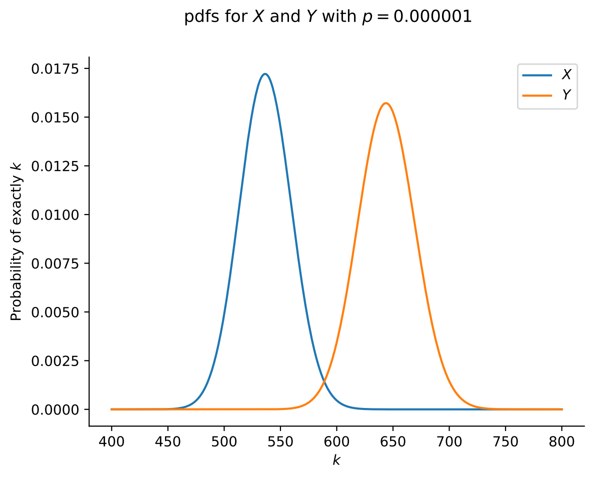
A single maintainer $\mathcal{M}$ maintains: (1) a directory Dir, of label-value pairs binding each user to a public key, (2) a VKD instance for auditing the append-only of Dir, and (3) a GBC channel instance for auditing the consistency of the append-only log provided by the VKD. To be clear, the single maintainer $\mathcal{M}$ will act as *both* IdP for the VKD and $\mathcal{CM}$ for GBC. The PKI which GBC is defined to take as input during setup is defined as Dir.

Time must proceed in non-overlapping phases as described in Section 3.1.4. Throughout the execution of the *KT* protocol, the maintainer collects registrations and updates of public keys from users for inclusion in Dir. Once per epoch, in the `Col` phase, $\mathcal{M}$ will include these new keys in the label-value directory Dir, commit to its new state via the VKD, and submit it to the GBC instance. (for future distribution in the `Distr` phase). Thus, the only publishing party on the GBC instance is the maintainer itself, and the only published data on GBC is the append-only commitment of the VKD instance maintained by $\mathcal{M}$ itself.

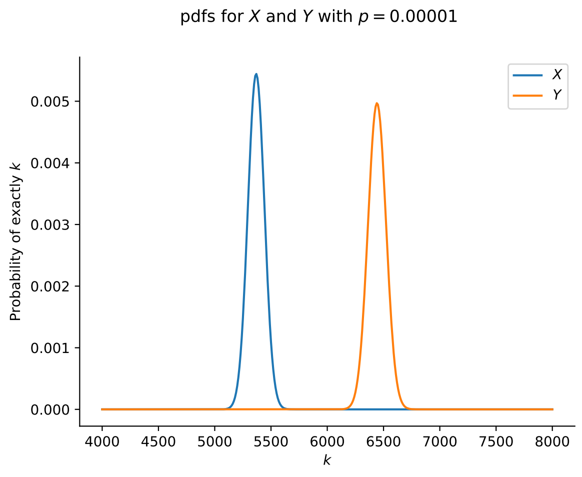
In the rest of the phases of an epoch, the maintainer follows the protocol specifications of VKD and GBC in a straightforward manner. There is no interaction between the protocols in these phases. Each user must participate in auditing the consistency of the *KT* protocol according to how auditing works in GBC and VKD.

6.2 Bootstrapping

In the first epoch of Protocol 1, there are neither any keys registered in the PKI nor any seed or endorsements of a previous epoch. To get around this issue, one can introduce a special bootstrapping epoch, $e = -1$, to be executed before anything else in the protocol. In this epoch, all users who exist at that time should first enroll their keys in the PKI. After that, a seed trusted to not have been precomputed (*e.g.* by using a hash of a current stock market valuation, or an MPC computed value) can be published by the $\mathcal{CM}$ to be used for selecting the endorsers of this special epoch.

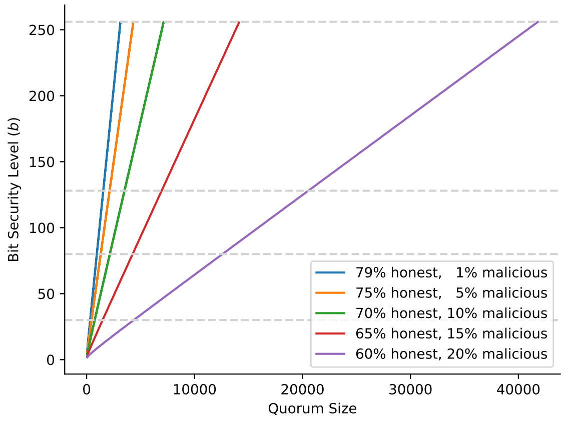


(a) Smaller Committee Size



(b) Larger Committee Size

Figure 2: Probability distribution function (pdf) visualization for population with 60% honest, 20% malicious and 20% silent users.



(a) Plotted

b	Quorum Size					
	79%	75%	70%	65%	60%	Honest
	1%	5%	10%	15%	20%	Malicious
256	3,133	4,329	7,141	14,145	41,816	
128	1,541	2,127	3,509	6,951	20,537	
80	944	1,306	2,152	4,260	12,584	
30	329	454	750	1,481	4,365	

(b) Tabulated

Figure 3: Optimal quorum sizes for split view defense in a 1,000,000,000 population in which 20% of the users are silent, with a varying degrees of honesty level in the remaining population.

6.3 Privacy

Some protocols for the append-only guarantee, *e.g.* CONIKS [32], SEEMless [10] and Parakeet [30] (but not all *e.g.* Merkle^2 [26]) provide a notion of privacy which limits information leaks such as the number of registered keys, when keys where updated and information about other keys than the one which was queried for during lookups. However, these protocols do leak, *and must leak*, information about what keys are registered in the system. This is since their main purpose is to provide append-only for key lookup. The identities of users and their public keys are *public information* in key lookup systems. Any privacy regarding the identities of registered users is thus necessarily weak. While Protocol 1 does not directly undermine many of the privacy guarantees of a VKD, GBC relies on an certain fraction of honesty among the users. Thus, to boost confidence in this assumption, it is natural for the identities in the PKI to be public. Since this is the normal case for a PKI, and since privacy guarantees of VKD are necessarily weak in the above sense, we argue that publishing the list of identities of users is sensible since membership of a system is in fact already public information.

6.4 Silent User Churn

While new users joining the system and old users explicitly leaving the system is not a problem in Protocol 1, some users might silently stop participating in the protocol without notification. For this to not affect the liveliness of the protocol (by overrunning the capacity for offline parties), such users need to be removed from the system.

6.5 Conclusions

We have provided the first consistency protocol for key transparency which – contrary to all previous work – simultaneously avoids to rely on small external committees of auditors, on OOB channels, and does not resort to using full broadcast systems or blockchains. We have further shown that our construction is practical for large scale applications such as WhatsApp and iMessage, capable of supporting billions of users.

ACKNOWLEDGMENTS

Joakim Brorsson was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Bernardo David was supported by the Independent Research Fund Denmark (IRFD) grant numbers 9040-00399B and 0165-00079B. Paul Stankovski Wagner was supported by the Swedish Foundation for Strategic Research grant RIT17-0035, and the Swedish Research Council grant 2019-04166.

REFERENCES

[1] Anonymous. 2024. *Code for implementation estimates.* https://anonymous.4open.science/r/SpeedTestCoD-0286/README.md

[2] Anonymous. 2024. *Code for statistical analysis.* https://anonymous.4open.science/r/quorum-stats-1742/README.md

[3] Apple. 2023. Advancing iMessage security: iMessage Contact Key Verification. https://security.apple.com/blog/imessage-contact-key-verification/. (2023). [Accessed 12-02-2024].

[4] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a Public Blockchain Keep a Secret?. In *TCC 2020, Part I (LNCS, Vol. 12550)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, Heidelberg, 260–290. https://doi.org/10.1007/978-3-030-64375-1_10

[5] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. 2022. On the (in)Security of ROS. *Journal of Cryptology* 35, 4 (Oct. 2022), 25. https://doi.org/10.1007/s00145-022-09436-0

[6] Josh Blum, Simon Booth, Brian Chen, Oded Gal, Maxwell Krohn, Julia Len, Karan Lyons, Antonio Marcedone, Mike Maxim, Merry Ember Mou, et al. 2022. Zoom Cryptography Whitepaper. https://raw.githubusercontent.com/zoom/zoom-e2e-whitepaper/master/zoom_e2e.pdf. (2022).

[7] Joseph Bonneau. 2016. EthIKS: Using Ethereum to audit a CONIKS key transparency log. In *International Conference on Financial Cryptography and Data Security*. Springer, 95–105.

[8] Joakim Brorsson, Bernardo David, Lorenzo Gentile, Elena Pagnin, and Paul Stankovski Wagner. 2023. PAPR: Publicly auditable privacy revocation for anonymous credentials. In *Cryptographers' Track at the RSA Conference*. Springer, 163–190.

[9] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. 2022. YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model. In *ASIACRYPT 2022, Part I (LNCS, Vol. 13791)*, Shweta Agrawal and Dongdai Lin (Eds.). Springer, Heidelberg, 651–680. https://doi.org/10.1007/978-3-031-22963-3_22

[10] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. 2019. SEEMless: Secure End-to-End Encrypted Messaging with less Trust. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 1639–1656. https://doi.org/10.1145/3319535.3363202

[11] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 75–92. https://doi.org/10.1109/SP46215.2023.10179419

[12] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. 2015. Efficient gossip protocols for verifying the consistency of certificate logs. In *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 415–423.

[13] Eric Cronin, Sugih Jamin, Tal Malkin, and Patrick Drew McDaniel. 2003. On the Performance, Feasibility, and Use of Forward-Secure Signatures. In *ACM CCS 2003*, Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger (Eds.). ACM Press, 131–144. https://doi.org/10.1145/948109.948130

[14] Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler. 2018. Aggregation-based certificate transparency gossip. *arXiv preprint arXiv:1806.08817* (2018).

[15] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In *FC 2019 (LNCS, Vol. 11598)*, Ian Goldberg and Tyler Moore (Eds.). Springer, Heidelberg, 23–41. https://doi.org/10.1007/978-3-030-32101-7_2

[16] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018, Part II (LNCS, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 66–98. https://doi.org/10.1007/978-3-319-78375-8_3

[17] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. 2022. GearBox: Optimal-size Shard Committees by Leveraging the Safety-Liveness Dichotomy. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 683–696. https://doi.org/10.1145/3548606.3559375

[18] Alexandra Dirksen, David Klein, Robert Michael, Tilman Stehr, Konrad Rieck, and Martin Johns. 2021. LogPicker: Strengthening Certificate Transparency Against Covert Adversaries. *Proc. Priv. Enhancing Technol.* 2021, 4 (2021), 184–202.

[19] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function with Short Proofs and Keys. In *PKC 2005 (LNCS, Vol. 3386)*, Serge Vaudenay (Ed.). Springer, Heidelberg, 416–431. https://doi.org/10.1007/978-3-540-30580-4_28

[20] Lukasz Dykcik, Laurent Chuat, Pawel Szalachowski, and Adrian Perrig. 2018. BlockPKI: An automated, resilient, and transparent public-key infrastructure. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 105–114.

[21] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. 2021. Random-Index PIR and Applications. In *TCC 2021, Part III (LNCS, Vol. 13044)*, Kobbi Nissim and Brent Waters (Eds.). Springer, Heidelberg, 32–61. https://doi.org/10.1007/978-3-030-90456-2_2

[22] Esha Ghosh and Melissa Chase. 2024. Weak Consistency mode in Key Transparency: OPTIKS. *Cryptology ePrint Archive* (2024).

[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*. 51–68.

[24] Martin Hirt and Vassilis Zikas. 2010. Adaptively Secure Broadcast. In *EUROCRYPT 2010 (LNCS, Vol. 6110)*, Henri Gilbert (Ed.). Springer, Heidelberg, 466–485. https://doi.org/10.1007/978-3-642-13190-5_24

[25] Team Hans Hoogstraaten, Daniël Niggebrugge, Danny Heppener, Frank Groenewegen, Janna Wettinck, Kevin Strooy, Pascal Arends, Paul Pols, Robbert Kouprie, Steffen Moorrees, et al. 2012. *Black Tulip.* Technical Report. Tech. Rep.(Fox-IT BV, 2012).

[26] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. 2021. Merkle^2: A Low-Latency Transparency Log System. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 285–303. https://doi.org/10.1109/SP40001.2021.00088

[27] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO 2017, Part I (LNCS, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[28] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. 2021. Certificate Transparency Version 2.0. RFC 9162. https://doi.org/10.17487/RFC9162

[29] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. 2023. OPTIKS: An Optimized Key Transparency System. Cryptology ePrint Archive, Paper 2023/1515. https://eprint.iacr.org/2023/1515 https://eprint.iacr.org/2023/1515.

[30] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. 2023. Parakeet: Practical key transparency for end-to-end encrypted messaging. *Cryptology ePrint Archive* (2023).

[31] Brendan McMillion. 2024. *Key Transparency Architecture.* Internet-Draft draft-ietf-keytrans-architecture-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-keytrans-architecture/00/ Work in Progress.

[32] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In *USENIX Security 2015*, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, 383–398.

[33] Meta. 2023. WhatsApp Key Transparency Overview. https://www.whatsapp.com/security/WhatsApp-Key-Transparency-Whitepaper.pdf. (2023). [Accessed 12-02-2024].

[34] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review* (2008).

[35] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. 2018. *Gossiping in CT.* Internet-Draft draft-ietf-trans-gossip-05. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-trans-gossip/05/ Work in Progress.

[36] Michael Oxford, David Parker, and Mark Ryan. 2020. Quantitative verification of certificate transparency gossip protocols. In *2020 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–9.
[37] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)* 27, 2 (1980), 228–234.
[38] Blog Post. 2015. Improved Digital Certificate Security. https://security.googleblog.com/2015/09/improved-digital-certificate-security.html. [Accessed 12-02-2024].
[39] Forum Post. 2016. Upcoming CT Log Removal: Izenpe — groups.google.com. https://groups.google.com/a/chromium.org/g/ct-policy/c/qOorKuhL1vA. [Accessed 01-02-2024].
[40] Forum Post. 2017. Upcoming Log Removal: Venafi CT Log Server — groups.google.com. https://groups.google.com/a/chromium.org/g/ct-policy/c/KMAcNT3asTQ. [Accessed 01-02-2024].
[41] Forum Post. 2020. Trust Asia 2021 has produced inconsistent STHs — groups.google.com. https://groups.google.com/a/chromium.org/g/ct-policy/c/VJaSg717m9g. [Accessed 01-02-2024].
[42] sslmate. 2024. Timeline of Certificate Authority Failures. https://sslmate.com/resources/certificate_authority_failures. [Accessed 12-02-2024].
[43] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping authorities" honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*. Ieee, 526–545.
[44] Alin Tomescu and Srinivas Devadas. 2017. Catena: Efficient Non-equivocation via Bitcoin. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 393–409. https://doi.org/10.1109/SP.2017.19
[45] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

A EXTENDED BACKGROUND ON TRANSPARENCY LOGS

Transparency logs are publicly verifiable label-value data structures maintained by a central party called the Identity Provider (IdP).

The most widely deployed transparency log system is Certificate Transparency [28] (CT) which ensures correct distibution of TLS-certificates. For example Google, Cloudflare and Let's Encrypt each host their own Certificate Transparency log, and inclusion in a log is mandatory for Chrome and Safari to accept a TLS certificate. There also exist major deployments [3, 6, 33] of Key Transparency (KT), which ensures correct distribution of public keys. Significant works on key transparency include CONIKS [32], Merkle^2 [26], SEEMless [10], Parakeet [30] and OPTIKS [29]. These build on the same core principles as CT, but are tailored for the use case of key transparency by using different underlying data structures, *e.g.* prefix trees rather than chronological trees, and assuming a different communication pattern with all communication going via a central party.

A.1 Verifiability of Transparency Logs

Transparency logs are verifiable to be *append-only*, *i.e.* nothing inserted into the log can be deleted or altered, and *consistent*, *i.e.* the same view of the monitored data structure is given to all users.

A.1.1 Verifiability of Append-Only. Intuitively, append-only means that a value, once assigned to a label, should never change. This prevents an Identity Provider from *e.g.* temporarily changing a public key associated with a specific identity during an attack, and then change it back later to avoid detection.

To prove that a data structure is append-only, the maintainer publishes a signed digest, sometimes called a commitment, of the log state for each epoch in the system. Each commitment binds the current state of a label-value data structure (exactly what labels exist and what are their values). The small commitment can then be distributed to all users. Users can verify whether a given value for a label is consistent with a commitment, and whether the updates to the data structure was append-only for two consecutive commitments. This is formalized as Verifiable Key Directories (VKD) [10] and Private Authenticated History Dictionaries (PAHD) [29].

A.1.2 Split-View Attacks. As describe above, VKDs and PAHDs provide append-only guarantees in relation to a sequence of commitments. They do however not provide any guarantees for the consistency of these commitments. Thus, when relying solely on a VKD/PAHD, there is no guarantee that the IdP does not keep separate versions of the VKD/PAHD for different users, serving different commitments to each user. If that is the case, even though a VKD/PAHD guarantees that values cannot be changed over time for each user, different users might have different views of the correct VKD/PAHD, and thus get different values for the same label. Such an attacks where the IdP keeps separate versions of the VKD/PAHD, are called *split-view attacks*, and allow the IdP to *e.g.* serve malicious public keys to targeted users, while serving the correct key to all other users to avoid detection.

Since transparency logs require both append-only and consistency properties, and VKDs and PAHDs only provide append-only guarantees, VKDs and PAHDs cannot be used by themselves to construct a transparency log. Instead they need to be paired with a protocol for verifible consistency.

B EXTENDED DISCUSSION AND LITERATURE REVIEW OF CONSISTENCY PROTOCOLS

There are different ways suggested in the literature to achieve consistency for transparency logs:

B.1 Consistency Via Blockchains

One suggested way is to rely on blockchains. Blockchains [23, 34, 45] solve the problem of securely handling financial transactions without involving any central party, and have broadcast-like properties. Thus, one can store data such as commitments for a VKD/PAHD on a blockchain to obtain consistency guarantees. Since blockchains come with extended security assumptions and performance penalties, one can argue that adding them only to use them for consistency is unnecessarily expensive.

Dykcik et al. [20], presents a straightforward scheme which incorporates a transparency log into a smart contract on a blockchain. Tomescu and Devadas in their Cathena system [44] design a scheme which leverages Bitcoin to provide consistency. Performance improvements over a straightforward blockchain for the entire log is presented. Bonneau in EthIKS [7] proposes to put CONIKS inside an Ethereum contract, allowing to piggyback on Ethereum's consensus protocol for security guarantees.

B.2 Consistency Via Gossip Protocols

Gossiping over OOB channels are the suggested method of achieving consistency by Certificate Transparency [28] and in a recent internet draft for Key Transparency [31]. In a gossip protocol, consistency guarantees are achieved by the log maintainer signing a commitment to the log state at recurring intervals, and then providing these signatures to users. Users of a log exchange these signatures and compare them. Inconsistencies between these signed commitments can thus be detected during comparison, and constitutes cryptographic proof of misbehaviour which can be be reported. Comparing signed commitments by the log maintainer is on a voluntary basis and occurs over OOB channels.

This approach, while avoiding blockchains and its drawbacks, only give statistical probabilities for detecting split-views (which are quite low as discussed below) rather than formal guarantees about consistency. Further, gossiping is done retroactively, meaning that split-view attacks will only be detected after they have occurred. Gossiping proactively would not be practical due to the "soft" guarantees and long waiting times. While this makes undetected split-view attacks harder, it does *not* rule them out, and users thus do not have guarantees that a public key is safe to use.

A further drawback of gossip protocols in the context of key transparency, is that they rely upon using OOB channels. If OOB channels are not used, an adversary controlling the network can suppress messages from targeted users to evade detection of an attack (or even alter them through Meddler-in-the-Middle attacks as we discuss below). This is problematic since OOB channels are not available at scale where KT protocols are deployed.

Chuat et al. [12] propose the first gossip protocols for Certificate Transparency and present results for how signed views are distributed using a simulation based on real Internet traffic traces at a 0.1% gossiping rate (the fraction of parties volunteering to gossip). In their results, after 24 hours, 11% of the users holds a signed view signed during this period. No results for the probability and speed of detection of log inconsistencies are presented.

The Chuat et al. gossip protocols are also evaluated in [36] by Oxford et al. Assuming a gossiping rate of 100%, the measure data dissemination (as also done by Chuat et al.), probability of split-view detection and rate of such detections. In their analysis, after 20 gossiping rounds the latest view is disseminated to all users, and a split-view attack is detected with 40% probability. They also show that if one makes the extra assumption of server-to-server gossip, these numbers can be improved.

Dahlberg et al. [14] explore how the network infrastructure such as routers and switched can provide gossiping as a service.

Nordberg et al. [35] proposes an Internet-Draft where gossiping takes place between a server and a client, so that if a meddler-in-the-middle attack occurs, it will be detected once it ends and the client established contact directly with the correct server.

We note that Apple's approach to gossiping in iMessage [3], where they piggyback gossip data on end-to-end encrypted messages over in-band channels (*i.e.* via Apple), does not solve this issue of in-band gossiping. Even though the gossip data is now encrypted, the IdP (Apple) can execute a split-view attack where it replaces public keys given to the victim, and act as a Meddler-in-the-Middle which alters gossiping data. Such an attack can be sustained indefinitely (assuming it does not break append-only guarantees). To the best of the authors knowledge, Apple has not provided an analysis for the security of this approach.

B.3 Consistency Via External Committees of Consistency Auditors

To avoid both the cost of blockchains and weak guarantees of gossip protocol, one can instead opt to use a set of external consistency auditors. These external auditors are centrally selected as a static set of parties. During the protocol execution, the IdP sends the signed commitment to all consistency auditors in each epoch. The auditors then sign these. So, instead of comparing commitments directly as in gossiping, users can compare their view with the signed views of the consistency auditors to see if it corresponds with a majority of the auditors. This approach provides both pro-active (immediate) detection of misbehaviour as well as formal guarantees, as long as the set of auditors has a sufficiently large fraction of honest parties.

The drawback however, is that it sacrifices the distributed nature of using blockchains or gossiping. Consistency guarantees in blockchains and gossiping is rooted in the honest of a very large set of users (all miners/stakers or all KT users). It is unlikely that even a powerful adversary can corrupt the majority of such large sets of users. The same can not be said about a small and well known set of consistency auditors. Current proposals suggest committee sizes of roughly 50 [30] auditors. A powerful adversary (say *e.g.* a state sponsored adversary) can realistically corrupt most, if not all, of a set of auditors of this size.

Parakeet by Malvai et al. [30], presents a simple protocol for a fixed set of external auditors which all sign their view, (up to 50 auditors are simulated in the paper). The identity provider needs to obtain a threshold of two thirds of such certifications agreeing on a single view to present to users in order to prove that no split-view attack is ongoing.

Dirksen et al. [18] lets a set of Certificate Transparency logs act as consistency auditors for each other by pitting them against each other, where for each cert, it is included in one log only and the other logs are expected to audit this log.

While increasing the number of external auditors would somewhat increase the resilience against split-view attacks, it would still not achieve the distribution of blockchains or gossip protocols, and thus not live up to the same level of resilience. Further, scaling up the number of auditors is non-trivial. Firstly, it is a problem in practice to find a large set of auditors which all users actually trust to have an honest majority. Second, it poses an efficiency problem for users which have to validate authenticity and correctness of the auditors' statements. For example, the overhead of parakeet's consistency protocol scales linearly with the number of auditors.

We note that Syta et al. [43] proposed a protocol for auditor cosigning with sublinear verification overhead, where auditors interact with each other to combine their signatures into a single multisignature. This protocol has however since been broken [5], and even if it were secure it would only scale to thousands of auditors, compared to millions or billions in blockchains or gossiping.

B.4 Weaker Versions of Consistency

[22] explores another direction of ensuring consistency for KT. The work in [22] provides a protocol which does not use external parties for auditing consistency. This is achieved by weakening the consistency guarantees of KT, so that the protocol only ensures that split-views are detected by *either* the party who queries for a key, *or* the key owner.