HeadStart: Efficiently Verifiable and Low-Latency Participatory Randomness Generation at Scale

Hsun Lee*, Yuming Hsu*, Jing-Jie Wang*, Hao Cheng Yang*, Yu-Heng Chen*, Yih-Chun Hu† and Hsu-Chun Hsiao*‡
*National Taiwan University, Taiwan †University of Illinois at Urbana-Champaign, USA ‡Academia Sinica, Taiwan
*{133xun, d768092}@gmail.com, {b06902035, b06902097, b07902026, hchsiao}@csie.ntu.edu.tw, †yihchun@illinois.edu

***Abstract*—Generating randomness by public participation allows participants to directly contribute randomness and verify the result's security. Ideally, the difficulty of participating in such activities should be as low as possible to reduce the computational burden of being a contributor. However, existing randomness generation protocols are unsuitable for this scenario because of scalability or usability issues. Hence, this paper presents HeadStart, a participatory randomness protocol designed for public participation at scale. HeadStart allows contributors to verify the result on commodity devices efficiently and provides a parameter L that can make the result-publication latency L times lower. Additionally, we propose two implementation improvements to speed up the verification further and reduce the proof size. The verification complexity of HeadStart is only $O(L \times polylog(T) + \log C)$ for a contribution phase lasting for time T with C contributions.**

I. Introduction

Unpredictable and unbiased randomness is crucial for many applications concerning public interest, such as randomly drawing residents for limited vaccination [57], [40], ensuring the trustworthiness of prize-linked savings accounts [42], and maintaining fairness in VAT-receipt lotteries [23]. Today, these applications usually generate random values using a computer program or a live-streamed ball-drawing or die-throwing event. However, these methods provide no verifiable proof that their random values are indeed unpredictable and unbiased. Past incidents have also eroded public confidence in randomness generated by third parties [5], [3].

One way to improve public confidence is via *participatory randomness generation*, which allows people to directly contribute entropy to the output and be assured of the result's unpredictability and bias resistance. To ensure the fairness of participation, such protocols should be secure, usable, and scalable so that even millions of people, without powerful machines, could contribute easily. Although a number of randomness generation techniques have been proposed since the introduction of coin-flipping protocols [11], [7] and randomness beacons [51], participatory randomness generation has not been widely adopted because existing protocols do not take the security, scalability, and usability of public participation into consideration simultaneously.

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.24234
www.ndss-symposium.org

Some protocols apply verifiable random functions (VRFs) [31], [27] or extract randomness from Bitcoin [14], [9], [50] or public financial data [24] to compute the random result. Nonetheless, these approaches are insecure for participatory randomness generation because adversaries who know all the current contributions can bias the result by precomputing the best outcome before contributing. In order to defend against these kinds of biasing attacks, a secure protocol must bound the adversary's ability to predict the result before the end of the contribution phase. This unpredictability prevents adversaries from contributing a crafted contribution to bias the result.

However, existing secure approaches are inapplicable for large-scale participatory randomness generation due to scalability or usability issues. Protocols based on commitments [39], publicly-verifiable secret sharing (PVSS) [55], [19], [53], [43], and threshold signatures [18], [37] require multiple phases with high communication overhead to distribute keys, commit entropy or secret shares, reveal commitments, and recover the final results. In addition, these protocols assume that all or the majority of the predefined fixed set of participants are honest, which limits its usability in practice. On the other hand, protocols based on delay functions [17], [22], [44], [45], [32], [13] and verifiable delay functions (VDFs) [52], [12], [29], [36] can prevent adversaries from precomputing the result in time by delaying the result generation. However, their delay of the result publication after the contribution phase, what we called result-publication latency (RP latency), is relatively high because of the long result-generation time caused by delay functions and VDFs. Although existing randomness beacon protocols [17], [24], [29], [36], [12], [52] can reduce their RP latency by decreasing the beacon interval, the contribution phase for a specific output will also be shortened. This shortened contribution phase is ill-suited for public participation events, which often need to last for several days or weeks to ensure fairness and availability of public participation.

Accordingly, we propose HeadStart, a participatory randomness generation protocol designed for public participation at scale. HeadStart is scalable because its communication and verification complexity for each contributor are both only $O(\log C)$ with respect to the number of contributions C. The usability is from HeadStart's ease of direct participation, reasonable assumption on the honest contributors, and low RP latency. The contributors can contribute entropy and efficiently verify HeadStart's result via their commodity devices without assuming that other contributors are honest. HeadStart's contribution phase can be opened for a sufficiently long time while

achieving a considerably low RP latency that everyone can receive the result earlier. The result of HeadStart is generated by an organizer, such as a government or a large enterprise, who is responsible for generating a verifiable random result to convince the public.

Efficient verification and low RP latency are two important advantages of HeadStart that make it suitable for public participation. HeadStart achieves efficient verification by storing contributions in Merkle trees and adopting VDFs, so the verification complexity is only $O(L \times \mathit{polylog}(T) + \log C)$ for a contribution phase lasting for time T with C contributions. To overcome high RP latency caused by VDFs, HeadStart divides the contribution phase into L stages and gives its result-generation a "head start" of $L-1$ stages. Therefore, the RP latency of HeadStart can be reduced to only $\frac{T}{L}$, which is a huge improvement over T in existing approaches based on delay functions or VDFs [17], [22], [44], [45], [52]. Moreover, HeadStart can be extended to a randomness beacon that can alleviate the burden of frequent re-registration when the beacon interval is short, because every honest participant can ensure the security of the subsequent L stages. All these improvements are only at the cost of verifying L VDF proofs with a time complexity of $O(L \times \mathit{polylog}(T))$.

To further enhance HeadStart's verification performance in practice, we present an algorithm named ClHash in Section VII-B to bring VDF proof aggregation from theory into practice. Although VDF proof aggregation is a known technique in theory [56], no efficient implementation was available because implementing it requires solving a complex mathematical problem. To the best of our knowledge, our VDF implementation is the first to support proof aggregation in the class groups of imaginary quadratic fields, which can aggregate L VDF proofs into one. Besides, we also present a mechanism called DHPI in Section VII-C to speed up the initialization of verification by about five times.

To demonstrate HeadStart's practicality for people using commodity devices, we evaluate its verification costs on personal computers and mobile phones using both browsers and native applications with various parameter settings. This evaluation shows that when $L = 100$ and $C = 10^6$, an iPhone XR application took only 10.7 seconds for verification regardless of the length of the contribution phase. This is significantly faster than previous schemes [22], [45] that require contributors to compute longer than the contribution phase.

This paper makes the following contributions:

1) We propose HeadStart, an efficiently verifiable and low-latency participatory randomness generation protocol with high usability and scalability.
2) We present ClHash: the first efficient algorithm that hashes to a class group of an imaginary quadratic field, bringing class-group VDF proof aggregation from theory into practice.
3) We describe the HeadStart beacon to solve the issue of frequent re-registration mentioned above.
4) We evaluate HeadStart through analysis and experiments. The results confirm that HeadStart is scalable and can be efficiently verified on commodity devices with small RP latency.

II. Background

This section introduces the cryptographic primitives used in this paper: *participatory randomness generation*, *Merkle trees*, and *verifiable delay function (VDF)*.

A. Participatory Randomness Generation

Participatory randomness generation allows a group of *contributors* to collectively generate an unpredictable, bias-resistant and verifiable result. Typically, it consists of two phases: (1) a *contribution phase* to gather randomness from the public (i.e., the contributors), and (2) a *result-generation phase* to compute the random output. It can be represented as $RG(k, x_1, x_2, \cdots, x_n) \rightarrow y$, where k is a security parameter and $x_1, x_2, \cdots, x_n$ are the contributions used to generate a randomness result $y \in [0, 2^k)$.

A participatory randomness generation protocol should satisfy the following security properties:

1) **Unpredictability**. If an adversary performs a prediction $y' \in [0, 2^k)$ to guess the output y before the end of the contribution phase, the probability that $y' = y$ is negligible with respect to the security parameter k.
2) **Bias Resistance**. An adversary cannot actively manipulate the result in a meaningful (i.e., predictable) way.
3) **Verifiability**. Honest contributors can verify the result's unpredictability and bias-resistance. Namely, they can ensure that no adversary can predict or manipulate the result to its advantage with non-negligible probability.

Unpredictability defends against passive observers who do not participate in the protocol, and bias resistance defends against active adversaries who can contribute to the input. Related work [53], [52] also explicitly listed both in their security properties, so we followed this convention for ease of comparison.

B. Merkle Trees

A Merkle tree [48] is a hash tree that employs a cryptographic hash function to build its tree nodes. Each leaf node is labeled with the hash value of the data, and each non-leaf node is labeled with the hash value of its children. In this work, we consider a typical Merkle tree implemented in binary form, while alternative implementations exist for specific purposes such as Fast Merkle Trees [46] in Bitcoin and Dense Merkle Trees in the Certificate Transparency storage system [33].

The structure of a Merkle tree allows it to provide efficiently verifiable proof-of-inclusion for requested data included in the leaf nodes. Such proof-of-inclusion consists of an audit path, i.e., the set of nodes required to compute the hash value of the tree root from a leaf. If the root computed using the audit path matches the root in the proof, then the audit path proves that the leaf is in the tree. As a result, the time and space complexities to verify proof-of-inclusion are both $O(\log N)$, where N is the number of tree nodes.

C. Verifiable Delay Function (VDF)

VDF [12] is a type of moderately hard cryptographic function whose results can be efficiently verified. It is a trio of algorithms, which can be represented as $VDF = (\texttt{Setup}, \texttt{Eval}, \texttt{Verify})$. These three algorithms are defined as follows:

1) $\texttt{Setup}(k, T) \rightarrow \mathbf{pp} = (ek, vk)$ is a randomized algorithm that takes a security parameter k and a time parameter T as inputs. Its output $\mathbf{pp}$, comprises two public parameters: an evaluation key ek and a verification key vk. By convention, a $\mathbf{pp}$ specifies the input space $\mathcal{X}$ and the output space $\mathcal{Y}$.
2) $\texttt{Eval}(ek, x) \rightarrow (y, \pi)$ is a deterministic algorithm that takes an input $x \in \mathcal{X}$, outputs the result $y \in \mathcal{Y}$ along with a proof π and must run in parallel time T with $poly(\log(T), k)$ processors.
3) $\texttt{Verify}(vk, x, y, \pi) \rightarrow \{True, False\}$ is a deterministic algorithm that outputs whether $y \in \mathcal{Y}$ is the correct output of $\texttt{Eval}(ek, x)$. The `Verify` algorithm, which is much faster than `Eval`, must run in total time $poly(\log(T), k)$.

For a `Setup` that needs secret randomness, such VDF will require a trusted setup to ensure that even the one that initializes the protocol does not have a trapdoor to speed up the VDF computation. For this reason, our implementation adopts a VDF with a non-trusted setup—specifically, one constructed based on the class groups of imaginary quadratic fields proposed by Wesolowski [56].

Additionally, a VDF must satisfy the following properties:

1) **Correctness**: For all k, T, $\mathbf{pp}$ and $x \in \mathcal{X}$, if $(y, \pi) \leftarrow \texttt{Eval}(ek, x)$, then algorithm $\texttt{Verify}(vk, x, y, \pi)$ must be $True$.
2) **Soundness**: A VDF is sound if for all algorithms $\mathcal{A}$ that run in time $O(poly(T, k))$ and $(x, y, \pi) \leftarrow \mathcal{A}(k, \mathbf{pp}, T)$, the probability of $\texttt{Verify}(vk, x, y, \pi) = True$ but $y \neq \texttt{Eval}(ek, x)$ is $negl(k)$.
3) **σ-sequentiality**: A VDF is σ-sequential if (1) `Eval` can be computed in T to output (y, π), and (2) no randomized algorithm with $poly(T, k)$ parallel processors can distinguish y with non-negligible probability before time $\sigma(T)$, i.e., the guaranteed lower bound of `Eval` time with $O(poly(T, k))$ processors.

In the random oracle model, one can extract a uniformly random string by applying a hash function to the unpredictable VDF output. The VDF needs not be pseudorandom to achieve this goal [12].

In addition, σ-sequentiality implies that one can only obtain negligible information about y before computing `Eval` with time $\sigma(T)$. A perfect VDF would achieve $\sigma(T) = T$, but this requirement is unrealistic. In practice, it is sufficient to use a VDF ensuring $\sigma(T) = T - o(T)$, or even $\sigma(T) = T - \epsilon T$ with small ϵ. Recently, there have been several competitions [25], [10] and research [41] working on accelerating VDF computation. Their results indicate that it remains challenging for the state-of-the-art hardware and implementation to obtain $\sigma(T)$ significantly smaller than T. Therefore, in this paper, we assume $\sigma(T)$ to be less than but close to T, i.e., $\sigma(T) \lesssim T$.

III. Threat Model

An adversary's goal is to obtain a verifiable result from the protocol that is to his advantage. Specifically, an adversary may passively predict or actively bias the inputs of the protocol to produce such a result. He may also try to manipulate others' inputs or collude with the organizer and any set of contributors.

We define an honest contribution as one that is random and unpredictable, and a contributor honest if he validly contributes an honest contribution and verifies the protocol. The organizer is responsible for generating the result of the protocol. However, the organizer is not necessarily honest, and may collude with the adversary.

The organizer and the contributors can publish messages to a public bulletin board, which provides correct message timestamps. An adversary cannot prevent honest parties from posting and reading information on it, and the posted information cannot be modified or removed. In practice, the public bulletin board could be a public database, e-mail service, notification service, or blockchain.

We construct HeadStart to defend against polynomially-bounded adversaries in a synchronous network, where messages are reliably delivered without delay, assuming that at least one honest contributor exists. We model cryptographic hash functions as random oracles. As noted above, a Merkle tree is constructed using a cryptographic hash function, so a Merkle tree root can also be considered an output of a random oracle.

IV. The HeadStart Protocol

HeadStart consists of four phases: setup, contribution, result-generation, and verification. The result-publication (RP) latency is the time span between the end of contribution and the end of result generation. HeadStart divides the contribution phase into L stages and gives the result-generation phase a "head start" of $L - 1$ stages, reducing the RP latency to L-fold.

To explain the HeadStart protocol clearly, we begin with a special case in which $L = 1$ before proceeding to the general case, in which $L > 1$, where L is the number of contribution stages. We present the notations in Table I, and illustrate the special case in Fig. 1 and the general case in Fig. 2. We describe each phase in the detail below.

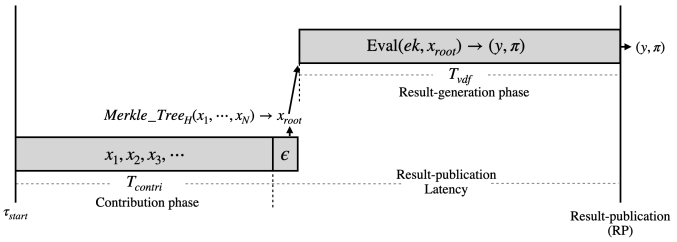

Fig. 1: The HeadStart special case in which $L = 1$

A. Special Case in Which $L = 1$

Setup phase. The setup phase consists of three sub-phases. These are: (1) parameter initialization, in which the organizer

chooses the security parameter k, a cryptographic hash function H, and a σ-sequential verifiable delay function $VDF = (\texttt{Setup}, \texttt{Eval}, \texttt{Verify})$ with k-bit security level, and also decides the time parameters: τ_{start}, t_{contri}, ϵ, and t_{vdf}, where $\sigma(t_{vdf}) > t_{contri} + \epsilon$, $t_{contri} = T_{contri}$ and $t_{vdf} = T_{vdf}$; (2) VDF setup, in which the organizer runs $\mathbf{pp} \leftarrow \texttt{Setup}(k, t_{vdf})$ to configure the VDF; and (3) parameter publication, in which the organizer publishes all the above parameters, including $k, H, VDF, \tau_{start}, t_{contri}, \epsilon, t_{vdf}, \mathbf{pp} = (ek, vk)$ to the bulletin board.

Contribution phase. The contributors save the published parameters and publish random values to the bulletin board as the contribution x_i, where x_i is in the range $[0, 2^k)$. After the end of this phase, the organizer gathers the contributions $(x_1, x_2, \cdots, x_C)$ from τ_{start} to $\tau_{start} + t_{contri}$.

Before entering the next phase, the organizer computes and publishes the Merkle tree root x_{root} and audit paths for each contributor to the bulletin board within time of length ϵ after the end of the contribution phase, which is subject to $t_{contri} + \epsilon < \sigma(t_{vdf})$, where $x_{root} \leftarrow Merkle\_Tree_H(x_1, x_2, \cdots, x_C)$.

Result-generation phase. Although the final result y is unknown at this moment, it actually has been determined because x_{root} has been published. The organizer then computes $(y, \pi) \leftarrow \texttt{Eval}(ek, x_{root})$ to obtain the result y and the VDF proof π. This computation takes t_{vdf} time. Lastly, the organizer publishes y and π to the bulletin board. As a result, the result publication occurs around $t_{vdf} + \epsilon$ after the contribution phase ends.

Verification phase. After result publication, contributors can verify the proof-of-inclusion and proof-of-VDF-correctness to assure that HeadStart's result is unpredictable and bias-resistant. To verify the proof-of-inclusion, each of the contributors acquires his Merkle tree audit path from the bulletin board, and then computes the root r with the audit path from his contribution x_i. If the computed root r matches the published root x_{root}, then proof-of-inclusion is deemed to be verified. To verify the proof-of-VDF-correctness, the contributor first confirms that the publication time of x_{root} is within time of length ϵ after the contribution phase, then computes $\texttt{Verify}\,(vk, x_{root}, y, \pi)$, and accepts the proof-of-VDF-correctness if $\texttt{Verify}$ returns $True$.

B. General Case in Which $L > 1$

The major advantage of HeadStart is that it provides a parameter L that can be configured to reduce the RP latency L-fold while preserving security. In the general case, we extend the special case by dividing the phases into L stages and thus give its result-generation a head start of $L - 1$ stages. Consequently, the RP latency of HeadStart can be reduced to around $\frac{T_{vdf}}{L}$.

Setup phase. The setup phase is made up of three subphases: (1) parameter initialization, in which the organizer chooses a parameter L, representing the number of contribution stages, along with the parameters as described in the special case, making $t_{contri} = \frac{T_{contri}}{L}$ and $t_{vdf} = \frac{T_{vdf}}{L}$. (2) VDF setup, in which the organizer computes $\mathbf{pp} \leftarrow \texttt{Setup}(k, t_{vdf})$ to setup the VDF ; and (3) parameter publication, in which the organizer publishes L and other previously described parameters to the bulletin board.

TABLE I: Notations.

Notation	Description
L	The number of divided contribution stages
C	The number of contributions
k	The number of bits of security level
τ_{start}	The start time of the contribution phase
T_{contri}	The duration of the contribution phase
t_{contri}	The duration of a contribution stage
ϵ	The duration of computing the Merkle tree root
T_{vdf}	The duration of the result-generation phase
t_{vdf}	The duration of the VDF
$\sigma(t_{vdf})$	The lower bound of $\texttt{Eval}$ with $O(poly(T, k))$ processors
x_i	The value of i^{th} contribution
$x_{i,j}$	The value of i^{th} contribution in stage j
x_{root}	The root value of the Merkle tree
$x_{root,j}$	The root value of the Merkle tree in the j^{th} stage
y_j	The randomness result in the j^{th} stage
π_j	The VDF proof in the j^{th} stage
$\|\|$	The string concatenation operation

Contribution phase. The contribution phase is divided into L stages, each spans t_{contri} and acts as a shorter contribution phase in the general case. After the j^{th} contribution stage ends, the organizer computes and publishes the j^{th} Merkle tree root $x_{root,j}$, and the audit paths for each contributor, within time of length ϵ.

Result-generation phase. This phase is also divided into L stages. The organizer computes $(y_1, \pi_1) \leftarrow \texttt{Eval}(ek, x_{root,1})$ for the first stage ($j = 1$), and computes $(y_j, \pi_j) \leftarrow \texttt{Eval}(ek, H(x_{root,j}||y_{j-1}))$ for all the other stages ($j > 1$). In the last result-generation stage ($j = L$), the result y_L is used as the result of the randomness generation. After y_L is computed, the organizer publishes all the values $y_1, \cdots, y_L, \pi_1, \cdots, \pi_L$ to the bulletin board around the time of length $L \times t_{vdf} + \epsilon - (L-1) \times t_{contri}$ after the contribution phase.

Verification phase. Contributors can verify the proof-of-inclusion and the proof-of-VDF-correctness after the result-publication. To verify the former, they acquire their audit paths from the bulletin board and follow the same process as in the special case. For the latter, they need to verify that all $x_{root,j}$ is published within time of length ϵ after the end of j^{th} contribution stage, and all L VDFs were correctly computed by running $\texttt{Verify}$ for each stage j. When $j > 1$, $\texttt{Verify}\,(vk, H(x_{root,j}||y_{j-1}), y_j, \pi_j)$ must return $True$, and when $j = 1$, $\texttt{Verify}\,(vk, x_{root,1}, y_1, \pi_1)$ must return $True$.

C. Performance Analysis

HeadStart is designed for public participation where a large group of ordinary people seeks to ensure fair allocation of limited social resources. According to this design concept, we analyze HeadStart's performance by its RP latency, verification efficiency, scalability, and usability.

HeadStart provides a parameter L that can reduce the RP latency. Compared with existing work based on delay functions or VDFs which result in high RP latency, HeadStart reduces the RP latency to $L \times t_{vdf} + \epsilon - (L-1) \times t_{contri}$. By setting the

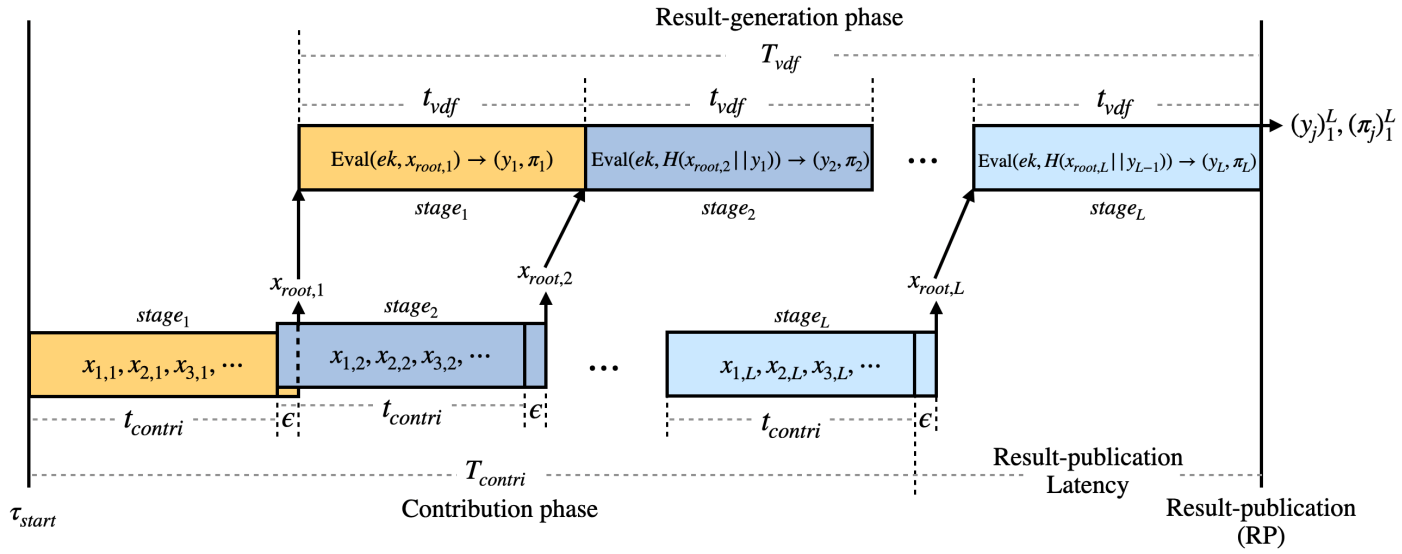

Fig. 2: The HeadStart general case in which $L > 1$

parameters to be $t_{vdf} \gtrsim \sigma(t_{vdf}) \gtrsim t_{contri} + \epsilon \gtrsim t_{contri}$, the RP latency would be approximately t_{vdf}, which is L times less than the previous work's. Consider when the allocation is about medical supplies or refugee housing. This acceleration would be extremely important because it allows the contributors to have a reasonably long time to register and contribute to ensure the fairness of participation and meanwhile guarantees them to receive the result significantly faster than previous work.

The verification complexity of HeadStart is $O(L \times polylog(T) + \log C)$ only because an honest contributor is only required to verify L VDFs and one Merkle tree. Although the VDF proof size grows linearly with the stage number L, we can adopt a VDF supporting proof aggregation [56], which can aggregate L VDF proofs into a single one, to reduce the proof size further.

About the scalability of HeadStart, the communication and verification complexity of a contributor are both $O(\log C)$ only with respect to the number of contributions C. In addition, HeadStart allows a large dynamic set of contributors to freely participate within the contribution phase without preregistration, whereas previous work [19], [53], [52] assumes a fixed set of participants which cannot be more or less during the protocol execution. Their requirements impair the scalability of public participation where the participants are ordinary people.

The usability of HeadStart is from the low RP latency, ease of direct participation, and reasonable assumption on the honest contributors. First, people can contribute in HeadStart within a reasonably long contribution phase, while the RP latency can remain considerably low. Second, people can efficiently verify the result of HeadStart on commodity devices as we demonstrate in Section VIII. Third, in reality, it is more convincing and practical to use HeadStart because the public can easily fulfill the assumption that at least one contributor is honest via direct participation, rather than asking them to believe that a system has a majority of honest participants.

In addition, the contributors in HeadStart cannot prevent the protocol from making progress, which avoids the public from denying the service whether intentionally or not. Previous work relies on a majority of honest participants to recover the unrevealed values [19], [53], or requires them to recompute the VDF evaluation to obtain the missing values hidden by the adversaries [52]. For many public participation events, it is reasonable to place the responsibility of protocol execution on the organizer such as a government or a large enterprise instead of spreading upon the participants, since the organizer has to maintain his public reputation. If the organizer does not care about its public reputation, it does not have to adopt a verifiable randomness protocol in the first place.

D. A Use Case of HeadStart: Vaccine Allocation

Let us consider the equitable COVID vaccine allocation launched in the US [40], [49] in 2021. First, the contribution phase must be open for a reasonable time interval, such as several days, for public registration and contribution. Second, the government starts the result-generation phase. Without the L-stage optimization provided by HeadStart, the result publication would have been prolonged for another several days. However, every hour of delay in the vaccination program significantly undermines public health.

Using the HeadStart protocol to start an urgent vaccine allocation event whose registration lasts for L stages, we can publish the vaccination list $L-1$ stages earlier. As a result, this allows people to receive their doses in a fair manner, avoids the vaccines from expiring, and provides flexible reaction time for the government.

Although existing randomness beacon protocols can reduce their RP latency by decreasing the beacon interval, the shortened RP latency either requires all participants to register and contribute within a same beacon period, or re-register frequently in each interval until they eventually get chosen. To conclude, the advantages over high-latency randomness beacons [12], [52] mentioned above demonstrate the importance of HeadStart's $L-1$ stages acceleration.

V. Security Analysis

This section provides evidence that HeadStart satisfies the security properties of participatory randomness generation. Without loss of generality, we let $\tau_{start} = 0$ in this section. We use the same notations as described in Table I.

A. Unpredictability

Unpredictability prevents an adversary from predicting the result before the publication of the Merkle tree root. Since the Merkle tree root is published right after the contribution phase ends, an adversary cannot predict the result if an honest contribution is included.

Lemma 1 (Unpredictability of the special case)**.** *When $L = 1$, an adversary Eve can not predict the result before the publication of the Merkle tree root.*

Proof: Suppose an honest contribution is submitted at time $\tau_s \in [0, t_{contri}]$. Eve cannot predict the result before time $\tau_s + \sigma(t_{vdf})$ because of σ-sequentiality. Also, HeadStart requires the server to publish the Merkle tree root before $t_{contri} + \epsilon$, so Eve cannot predict the result before the publication since $t_{contri} + \epsilon < \tau_s + \sigma(t_{vdf})$. ■

Next, we prove the unpredictability of jth result-generation stage's output in the general case with L stages.

Lemma 2. *If an honest contribution is included within any of the first j contribution stages ($1 \leq j \leq L$), an adversary Eve cannot predict the j^{th} result-generation stage's output y_j before the publication of the Merkle tree root of the j^{th} contribution stage.*

Proof: We prove it by induction.

For the case when $j = 1$, the honest contribution must be in the first (and only) contribution stage. According to Lemma 1, Eve cannot predict the output before the publication of the Merkle tree root of this (first) contribution stage.

Assuming that the statement holds when $j = j'$: if an honest contribution is included within any of the first j' contribution stages ($1 \leq j \leq L$), Eve cannot predict the j'^{th} result-generation stage's output y_j before the publication of the Merkle tree root of the j'^{th} contribution stage.

When $j = j' + 1$, there are two possible cases: the honest contribution is either within the first j' contribution stages, or in the $(j'+1)^{th}$ stage.

Case 1: The honest contribution is within the first j' contribution stages. According to the induction hypothesis, Eve cannot predict the output $y_{j'}$ before the publication of the Merkle tree root of the j'^{th} contribution stage. So $y_{j'}$ is unpredictable at time $t_{contri} \times j'$ for Eve, and so is the $(j'+1)^{th}$ result-generation stage's input $H(x_{root,(j'+1)}||y_{j'})$. By σ-sequentiality, Eve cannot predict the $(j'+1)^{th}$ result-generation stage's output $y_{j'+1}$ before time $t_{contri} \times j' + \sigma(t_{vdf})$. Also, HeadStart requires the server to publish the Merkle tree root of the $(j'+1)^{th}$ contribution stage before $t_{contri} \times (j'+1) + \epsilon$, so Eve cannot predict the result before the publication because $t_{contri} \times (j'+1) + \epsilon < t_{contri} \times j' + \sigma(t_{vdf})$, which is the end of the $(j'+1)^{th}$ contribution phase.

Case 2: The honest contribution is in the $(j'+1)^{th}$ contribution stage. Suppose the honest contribution is submitted at time $t_{contri} \times j' + \tau_s$, where $\tau_s \in [0, t_{contri}]$, then the $(j'+1)^{th}$ result-generation stage's input $H(x_{root,(j'+1)}||y_{j'}$ is unpredictable for Eve at time $t_{contri} \times j' + \tau_s$. By σ-sequentiality, Eve cannot predict the $(j'+1)^{th}$ result-generation stage's output $y_{j'+1}$ before time $t_{contri} \times j' + \tau_s + \sigma(t_{vdf})$. Also, HeadStart requires the server to published the Merkle tree root of the $(j'+1)^{th}$ contribution stage before $t_{contri} \times (j'+1) + \epsilon$, so Eve cannot predict the result before the publication because $t_{contri} \times (j'+1) + \epsilon < t_{contri} \times j' + \tau_s + \sigma(t_{vdf})$.

Thus, as shown by Cases 1 and 2, the statement holds when $j = j' + 1$. By induction, the statement holds for every j ($1 \leq j \leq L$). ■

Then we use Lemma 2 to prove the unpredictability of the general case with L stages.

Theorem 1 (Unpredictability)**.** *An adversary Eve can not predict the result before the publication of the final Merkle tree root.*

Proof: We apply $j = L$ on Lemma 2: An adversary Eve cannot predict the L^{th} result-generation stage's output y_L before the publication of the Merkle tree root of the L^{th} contribution stage if an honest contribution is included within any of the first L contribution stages. The assumption holds because HeadStart assumes that there exists at least one honest contribution and there are exactly L contribution stages. Also, the L^{th} result-generation stage's output y_L is the result, and the Merkle tree root of the L^{th} contribution stage is the final Merkle tree root. So we can conclude that Eve cannot predict the result before the final Merkle tree root. ■

B. Bias Resistance

Bias resistance defends against adversaries who can contribute to or manipulate the input from leading the result to their advantage. Although the adversaries may change the result, we show that they cannot predict the changed result to gain advantages.

Theorem 2 (Bias resistance)**.** *An adversary Eve cannot actively manipulate the result in a meaningful (i.e., predictable) way.*

Proof: In the HeadStart protocol, the only inputs that Eve can manipulate are the leaves of the Merkle tree. Eve can add, exclude, and reorder the leaves. If Eve excludes an honest contribution, then the proof-of-inclusion of that contribution can not be provided and the result would not be accepted. In other cases, there is still at least one honest contribution. Following the unpredictability of HeadStart (Theorem 1), Eve cannot predict the result before the publication of the final Merkle tree root, no matter what kind of manipulation is performed. Therefore, Eve can not influence the result in a meaningful way. After the publication of the final Merkle tree root, the inputs of each VDF are unchangeable, so the result cannot be influenced in any way. ■

C. Verifiability

An honest contributor can act as a verifier to verify the unpredictability and bias resistance of HeadStart's result with

the following information:

1) the audit path of the Merkle tree containing the contributor's contribution,
2) H, VDF, vk,
3) $x_{root,j}$, $\forall 1 \leq j \leq L$, and their timestamps,
4) (y_j, π_j), $\forall 1 \leq j \leq L$.

These values are public on the bulletin board, so verifiers can acquire all these messages. The organizer cannot abort or refuse to follow the protocol because it will be detected by the verifiers who check the validity of information on the bulletin board. If an honest contributor has verified that his contribution is indeed in the Merkle tree, then there is at least one honest contribution and thus the unpredictability and bias resistance hold.

VI. HeadStart Extensions

A. HeadStart beacon

HeadStart can be applied to implement a randomness beacon service [17], [24], [29], [36], [12], [52] by continuously accepting public contributions and outputting the results from each stage periodically. It can be viewed as an extension of HeadStart with endless stages: each stage outputs the result and passes on the result as an input to the next stage. For those who want to use the beacon, they can directly contribute within a range of stages and verify the result in exactly the same way as HeadStart.

Aside from the advantages inherited from HeadStart, a HeadStart beacon has another crucial advantage based on Theorems 1 and 2: every honest contributor can ensure the security (unpredictability and bias-resistance) of the subsequent L stages at the cost of L verifications. Consider when the beacon interval is configured to be a short time (such as several minutes) to ensure a low RP latency. In a HeadStart beacon, the contributors only need to contribute once and compute L verifications after L intervals to ensure that all the results within these intervals are secure. Compared with the existing randomness beacon services [17], [24], [29], [36], [12], [52], where the contributors must re-register and re-contribute frequently for the following L stages to achieve the same purpose, the HeadStart beacon provides an L-fold trade-off to alleviate the burden.

B. Publicly-verifiable HeadStart

HeadStart can be extended to achieve public verifiability, which ensures that a third party can verify the correct execution of the protocol via a protocol transcript.

Recall that in HeadStart, a contributor accepts the verification if his honest contribution is included because this is enough to ensure unpredictability and bias-resistance of the result. Although contributors can provide contributions for a third party to verify, HeadStart does not strictly achieve the above definition of public verifiability because this verification does not ensure the inclusion of other contributions.

We can extend HeadStart to be publicly verifiable by requesting verifiers to recompute the Merkle tree from all contributions. Alternatively, we can replace the Merkle tree with a hash chain where $chain_1 = H(x_1), chain_i = H(chain_{i-1}||x_i)$ for $i > 1$, and using the result as the input of the VDF in each stage. Via these modifications, the verifiers can ensure that all contributions are correctly included by recomputing the Merkle tree or the hash chain. Both of them increase the verification and communication complexity from $O(\log C)$ to $O(C)$, and the hash chain version has a constant factor improvement.

Formally, third parties and honest contributors can verify the result in the hash chain version of publicly-verifiable HeadStart by using the following public information:

1) all the contributions $(x_1, x_2, \cdots, x_C)$ within the contribution phase,
2) H, VDF, vk,
3) $x_{chain,j}$, $\forall 1 \leq j \leq L$, and their timestamps,
4) (y_j, π_j), $\forall 1 \leq j \leq L$.

These values are public on the bulletin board, so third parties and honest contributors can recompute the hash chain in each stage to verify proof-of-inclusion of every contributor, construct the input of VDF, and verify the protocol the same way as in HeadStart.

Frankly, public verifiability is not required for public participation scenarios because anyone who wants to verify the result can easily participate within the HeadStart protocol. HeadStart allows a large dynamic set of contributors to participate, whereas previous schemes [19], [53], [52] assume a fixed set of participants throughout the protocol execution. Those schemes are designed for the case when a group of preregistered or private users wants to generate verifiable random outputs for others outside the group, so they need to achieve public verifiability to convince those outside the group. In those schemes, public verifiability is based on the assumption that the majority of the preregistered users are honest; otherwise, their security is unverifiable because third parties do not know the number of honest users.

It is practical and convincible for public participation events to use HeadStart because the public can easily fulfill the assumption that at least one contributor is honest via direct participation, rather than asking them to believe that a system has a majority of honest users or nodes.

VII. Implementation

This section describes how we implement a fully functional HeadStart system with improved performance to demonstrate its practicality. We first describe our VDF selection criteria and the construction of the VDF in Section VII-A, because our improvements are highly related to its detailed construction.

We present our two implementation-specific improvements: (1) *ClHash, an efficient algorithm that hashes to a class group of an imaginary quadratic field*, which brings class-group VDF proof aggregation from theory into practice (§VII-B), and (2) *determined hash prime iteration*, which can further speed up the initialization of verification (§VII-C). In addition, we describe our programming language choices for our HeadStart implementation (§VII-D).

A. VDF Selection and Construction

a) VDF selection: We choose to use Wesolowski's *class-group VDF* [56], which will be referred to as class-group VDF, for three reasons. First, the verification of class-group VDF is very efficient because its complexity of group operations is $O(1)$, faster than the formal requirement of VDF, which is $poly(\log(T), k)$. Second, it is based on groups of unknown order and can be initialized without a trusted setup. Third, it supports proof aggregation in theory [56], allowing further reductions in the proof size, and we present our algorithm ClHash to bring this theory into reality.

Although using a succinct argument with a sequential function (e.g., a hash chain) is a straightforward method to construct a VDF, we did not use SNARKs [15], [30] because of its trusted setup requirement. As for STARKs [8], its proof size and verification complexity are $poly(\log(T), k)$. For public participation choosing large T because of the need for a long contribution phase, it is better to use a class-group VDF because the verification complexity of its costly group operations is only $O(1)$.

b) Class-group VDF construction: We now briefly introduce the construction of the class-group VDF proposed by Wesolowski [56] to provide sufficient information before presenting our improvements in the following subsections.

The basis of class-group VDF is the class group of an imaginary quadratic field. In practice, the binary quadratic form is used to represent the group elements, defined as follows.

$$(a, b, c) = ax^2 + bxy + cy^2 \in \mathbb{Z}[x, y] \tag{1}$$

The discriminant of (a, b, c) is $d = b^2 - 4ac$ if $d \equiv 1 \pmod 4$. When $d < 0$, each (a, b, c) uniquely represents an element in the class group of an imaginary quadratic field with discriminant d. The notation $Cl(d)$ represents a class group with discriminant d.

Wesolowski's work provides a non-interactive VDF proof based on the Fiat-Shamir heuristic. It takes the following parameters: a hash function $d \leftarrow H_D(x)$ mapping an input string x to a discriminant d, a group $Cl(d)$, a hash function $g \leftarrow H_{Cl}(x, d)$ mapping an input string x to a group element $g = (a, b, c) \in Cl(d)$, a time parameter T, a security parameter k_ℓ, a uniformly random string x_{root} (which in HeadStart is the Merkle tree root), and a hash function H_{prime} mapping an input string onto a prime p (where $2^{k_\ell - 1} < p < 2^{k_\ell}$). We denote $bin(s)$ as the binary representation of s and $||$ as the string concatenation. Its process is as follows:

1) The prover takes the string x_{root} as the input, computes $d \leftarrow H_D(x_{root})$, $g \leftarrow H_{Cl}(x_{root}, d)$, and evaluates the group operations as $y \leftarrow g^{2^T}$, where $y \in Cl(d)$ is the outcome.
2) The prover computes $\ell \leftarrow H_{prime}(bin(g)||bin(y))$, and generates the proof via $\pi \leftarrow g^q$, where $q = \lfloor 2^T/\ell \rfloor$; and finally, sends y and π to the verifier.
3) The verifier receives the outcome y and π, then computes the discriminant $d \leftarrow H_D(x_{root})$, the group element $g \leftarrow H_{Cl}(x_{root}, d)$, the non-interactive challenge prime $\ell \leftarrow H_{prime}(bin(g)||bin(y))$, and the remainder $r = 2^T \mod \ell$. Finally, The verifier accepts the result if $\pi^\ell g^r = y$.

This process allows individuals to verify the correctness of the VDF evaluation in a constant number of group operations with respect to the time parameter T. Additionally, without knowing the group order of $Cl(d)$, there is currently no practical way to compute g^{2^T} faster than directly performing T sequential squarings. [56] For this reason, the class groups of imaginary quadratic fields are used to construct such groups of unknown order.

The class groups were introduced in cryptography by Buchmann et al. [16], presenting the difficulty of computing their orders. Boneh et al. [12] and Wesolowski [56] then describe its use in constructing VDFs. By selecting a sufficiently large negative prime discriminant $d \equiv 1 \pmod 4$, the order of a class group $Cl(d)$ is believed to be hard to compute [34], [35]. Hence, class-group VDF is used to construct a VDF without a trusted setup.

In order to reach a 128-bit security level, Dobson et al. [28] proposed that one must set the discriminant to 6,656 bits. Also, Belabas et al. showed a specific form of discriminant called trapdoor discriminant, which breaks the unknown order property [6]. They further stated that the probability of having a trapdoor discriminant is negligible if the discriminant is chosen at random. Thus, we adopted these suggested settings in our implementation to ensure a sufficient level of security. Below, we denote the bit-length of a discriminant as k_d; when we make reference to "k_d-bit discriminant d", it means d must meet the condition $2^{k_d - 1} < |d| < 2^{k_d}$.

c) VDF proof aggregation: We describe how we can aggregate HeadStart's L VDF proofs to a single proof to show the correctness of multiple VDFs based on Wesolowski's theory [56].

First, the prover computes: $g_j \leftarrow H_{Cl}(d)(x_{root,j})$, $s = bin(g_1)||\cdots||bin(g_n)||bin(y_1)||\cdots||bin(y_n)$, $\ell \leftarrow H_{prime}(s)$ and $\alpha_j \leftarrow int(H(bin(j)||s))$. Second, the prover computes and publishes the aggregated proof $\widetilde{\pi} \in Cl(d)$ as follows:

$$\widetilde{\pi} = \left(\prod_{j=1}^{L} g_j^{\alpha_j} \right)^{\lfloor 2^T/\ell \rfloor} \tag{2}$$

Finally, the contributor can verify VDFs of all L stages by computing $r = 2^T \mod \ell$ and accepts if

$$\widetilde{\pi}^\ell \left(\prod_{j=1}^{L} g_j^{\alpha_j} \right)^r = \prod_{j=1}^{L} y_j^{\alpha_j} \tag{3}$$

However, the procedure $g \leftarrow H_{Cl}(x, d)$ is an abstact function defined as H_G in Wesolowski's paper [56]. In fact, it is not trivial to construct a function $g \leftarrow H_{Cl}(x, d)$ that receives a string x and a large negative prime as discriminant d to output a $g = (a, b, c) \in Cl(d)$ with respect to x. As a result, we invent ClHash, an algorithm that solves this problem and concretizes this abstract function.

B. ClHash

We invent an algorithm (ClHash) to construct an efficient hash function that maps a string to a class group of an imaginary quadratic field. To the best of our knowledge, ClHash is the first efficient implementation to bring class-group VDF proof aggregation from theory into practice.

To construct ClHash, we state the problem as follows. Given an input x and a large negative prime discriminant d, where $d \equiv 1 \pmod 4$, find the $g = (a, b, c) \in Cl(d)$ with respect to x such that $d = b^2 - 4ac$ and $c \geq a \geq |b|$. We present the pseudo-code of our algorithm in Fig. 3, and the following is the procedure with detailed explanation of our algorithm:

1) We use x to deterministically generate a prime a where $a \equiv 3 \pmod 4$.
2) By observing $d = b^2 - 4ac$, we can see that $4 \mid b^2 - d$, so b must be an odd number. As a result, we have to find an odd number b satisfying $b^2 \equiv d \pmod a$ to ensure that $c = \frac{b^2-d}{4a}$ is also an integer.
3) We adopt Euler's criterion: there is an integer b such that $b^2 \equiv d \pmod a \iff d^{\frac{(a-1)}{2}} \equiv 1 \pmod a$, to test whether a satisfies the requirement. According to [26], the success rate of this step is about 50%.
4) If a satisfies the requirement, we can compute $b \equiv d^{\frac{a+1}{4}} \mod a$ to satisfy $b^2 \equiv d \pmod a$, because $b^2 \equiv (d^{\frac{a+1}{4}})^2 = d^{\frac{a+1}{2}} = d \cdot d^{\frac{a-1}{2}} \equiv d \cdot 1 \equiv d \pmod a$.
5) If the final b is not an odd number, let $b \leftarrow a - b$. This is because if b is even and $b^2 \equiv d \pmod a$, then $a - b$ is an odd number and $(a - b)^2 \equiv d \pmod a$.
6) Finally, we obtain the element $g = (a, b, \frac{b^2-d}{4a}) \in Cl(d)$.

We analyze the image of ClHash for the security of the VDF construction. According to the prime number theorem [26], the number of prime numbers $\leq N$ is $\pi(N) \approx \frac{N}{\ln(N)}$, so the number of 256-bit prime numbers is $\pi(2^{256}) - \pi(2^{255}) \approx 2^{247.5}$. By Dirichlet's theorem on arithmetic progressions, prime numbers are equally distributed between the classes of the form $4K + 1$ and $4K + 3$, so there are about $2^{246.5}$ 256-bit primes congruent to 3 mod 4. Now the question is: Given a 6656-bit prime number d, how many of these $2^{246.5}$ prime numbers a satisfying the constraint that d is a quadratic residue modulo a? To answer this question, we can see that given one of these a, half of the number are a quadratic residue modulo a according to Euler's criterion. Since the distribution of d is not related to a, we can estimate that about half of the 6656-bits prime numbers are a quadratic residue modulo a. In other words, half of these (a, d) pairs are valid pairs. Putting these together, given a 6656-prime number d, the expected number of valid a is half of the $2^{246.5}$ 256-bit prime numbers congruent to 3 mod 4, which is $2^{245.5}$.

To analyze the time complexity of ClHash, let $iter_a$ be the expected number of iterations needed to generate a prime with k_l bits. For each generated prime, the success rate of passing the primality test is about 50% in step (3), so we need $2 \times iter_a$ iterations in expectation to generate a. Because ClHash's computation is dominated by a's generation (b and c can be efficiently computed), ClHash's expected time complexity is about $2 \times iter_a$ primality tests with k_l-bit numbers..

```python
def ClHash(x, d, k_l):
    while True:
        sprout = sha256(x).hexdigest()
        a = int(sprout, 16)
        a |= (1<<(k_l-1))
        # make a=3(mod 4)
        a |= 3

        if isprime(a):
            if pow(d, (a-1)/2, a)==1:
                b = pow(d, (a+1)/4, a)
                if b%2!=1:
                    b = a-b
                c = (b*b-d)/(4*a)
                return (a, b, c)

        # Increamentally change the sprout
        # to generate next candidate.
        t = ""
        for i in range(0, len(sprout)):
            t += chr(ord(sprout[i]) + 1)
        x = t
```

Fig. 3: Pseudo-code of ClHash, a hash function which receives given string x, discriminant d and security parameter k_l for a to generate a group element $(a, b, c) \in Cl(d)$.

C. Determined Hash Prime Iteration (DHPI)

Our second improvement is *determined hash prime iteration (DHPI)*, which can speed up the verification further by skipping those failed primality tests.

To verify the VDFs, contributors need to compute $d \leftarrow H_D(x, k_d)$, whose pseudo-code is presented in Fig. 4, to obtain the discriminant. H_D involves a primality-test procedure for verifying whether the discriminant candidate d is a prime. However, most of its running time is spent on failed primality tests, and it is time-consuming to find a valid 6,656-bit discriminant.

DHPI allows contributors to skip failed primality tests and verify the correct one directly, thereby significantly reducing the verification overhead. To achieve this, the organizer is asked to provide $iter_d$ (i.e., the number of iterations needed to generate the corresponding d) and publish it together with the Merkle tree root. The contributors can skip the primality tests in the first $iter_d - 1$ iterations, and only run it in the i^{th} iteration to ensure that the final d is truly a prime.

To prevent an adversary from gaining advantage by repeating the iteration until it gets a trapdoor discriminant, we set an upper bound of $iter_d$ to ensure that the probability of finding a trapdoor discriminant within $iter_{upper}$ is negligible. During the verification phase, contributors can choose to calculate d if $iter_d \leq iter_{upper}$ is false, which occurs rarely, as estimated below.

We perform an estimation on the probability of H_D failure when setting $iter_{upper}$ (fix $k_d = 6656$). First, by prime number theorem [26], the number of 6656-bit prime numbers is $\pi(2^{6656}) - \pi(2^{6655})$, where $\pi(N) \approx \frac{N}{\log(N)}$ is the prime-counting function. The number is approximately $\frac{2^{6656}}{6656 \ln 2} - \frac{2^{6655}}{6655 \ln 2} = 2^{6655} \frac{6654}{6656 \cdot 6655 \ln 2}$. By Dirichlet's theorem on arithmetic progressions [54], prime numbers are equally distributed between the classes of form $8K+1, 8K+3, 8K+5, 8K+7$. Therefore, when choosing a candidate d of form $8K+7$ in each iteration of H_D, it has probability $p = \frac{2 \cdot 6654}{6656 \cdot 6655 \ln 2}$ to pass the

```
def H_D(x, k_d):
    while True:
        sprout = sha256(x).hexdigest()
        d = int(sprout, 16)
        d |= (1<<(k_d-1))
        d |= 7

        # In our optimized implementation
        # Organizer directly tell contributors
        # which iteration of the while loop
        # will pass this primality test.
        if isprime(d):
            return -d

        # Increamentally change the sprout
        # to generate next discriminant candidate.
        t = ""
        for i in range(0, len(sprout)):
            t += chr(ord(sprout[i]) + 1)
        x = t
```

Fig. 4: Pseudo-code of H_D, a function returning a k_d-bit negative prime $d \equiv 1 \pmod 8$.

primary test, and the probability that the organizer fails to find a valid discriminant within $iter_{upper}$ steps is approximately $(1-p)^{iter_{upper}}$. By setting $iter_{upper} = 10^5$, the probability is about 9×10^{-10}.

D. Implementation Details

Our VDF implementation, AggVDF, adopts our ClHash algorithm to support proof aggregation in class groups of imaginary quadratic fields and supports DHPI. With ClHash, which can map strings into class groups with identical discriminants, AggVDF can create multiple elements with the same discriminant to compress their size. Compared with existing implementations [1], [4], [38], in which all the group elements must be represented in the form of (a, b, d) with different d, our elements can be easily represented in the form of (a, b) to further reduce the proof size.

In addition, we set the bit-length of discriminant to be 6,656 bits as proposed by Dobson et al. [28]. To reach a 128-bit security level, 6,656-bit is a much secure discriminant bit length, whereas most of the current implementations of class-group VDFs [20], [38], [4] use much smaller 512-bit, 1024-bit, or 2048-bit discriminants, thereby providing insufficient levels of security.

Our AggVDF adopts the available implementation of class-group evaluation provided by Chia Network [1] because Chia Network held worldwide competitions for the fastest implementation of class-group VDFs [21] to reduce the chance that an adversary gains advantage via faster VDF evaluation. We will refer to Chia Network's implementation as the *ChiaVDF*.

Our implementation of HeadStart contains a server for the organizer and clients for contributors. On the server side, we implement the VDF in C++ and the rest in Golang. On the client side, we implement HeadStart as a mobile native application and a web application. The mobile application is written in Objective-C and Objective-C++, and the web application is written in WebAssembly. Both Objective-C++ and WebAssembly use the same C++ code base, with cross-compilation to ensure implementation consistency and good performance. Our code can be accessed via this link: https://github.com/csienslab. We plan to open-source our implementation after publication.

VIII. Evaluation

We evaluate HeadStart's scalability and usability to demonstrate its practicality in public participation. Our usability metrics are verification efficiency and RP latency, where the former aims to increase ease of participation so that the contributors can easily join HeadStart with their own mobile phones, and the latter aims to provide fast result publication with a sufficiently long contribution phase for the public.

We evaluate HeadStart using both theoretical analysis and experiments based on the ClHash and DHPI implementation presented in Section VII. We examine its RP latency and verification cost under different numbers of contribution stages (L) and types of commodity devices. Our results confirm that HeadStart achieves efficient verification and low RP latency, as the verification time is consistently low on commodity devices with realistic settings.

Specifically, we conduct experiments to answer the following research questions:

1) Scalability: Does the scale of contributions affect the performance of the verification? (§VIII-A)
2) Verification efficiency: How efficient is HeadStart's client verification? How effective is our DHPI? How much proof size can we reduce via proof aggregation? (§VIII-B, §VIII-C)
3) RP latency: What are the factors influencing the length of the RP latency in practice? (§VIII-D)

To further ensure practicality in public participation, we aim to suggest a practical parameter setting not only on generic mobile phones and laptops but also on old mobile phones. As a result, we use iPhone 6 (two cores and used for seven years) to represent old mobile phones, iPhone XR (six cores) to represent generic mobile phones, and MacBook (2.3 GHz, four cores, Intel Core i7) to represent laptops, in order to demonstrate the practicality on these devices.

A. Scalability of HeadStart

To show that HeadStart can scale to a large number of contributions, we analyze the communication complexity from two perspectives, that of the organizer and that of a contributor, as C grows, and we evaluate its verification complexity with respect to the number of contributions (C).

The communication complexity with respect to the number of contributions can be evaluated by the space complexity of the communication messages within HeadStart. For the organizer, it takes $O(C \log C)$ space complexity to gather C contributions and post C audit paths of the Merkle trees to non-interactively show the proof-of-inclusions to each contributor. For the contributors, each of them takes $O(\log C)$ space complexity to read his audit path of the Merkle tree for proof-of-inclusion. As a result, the communication complexity for each contributor is quite small, increasing ease of participation.

Recall that HeadStart's verification complexity is $O(L \times polylog(T) + \log C)$, where $O(L \times polylog(T))$ is for verifying L class-group VDFs and $O(\log C)$ is for proof-of-inclusion.

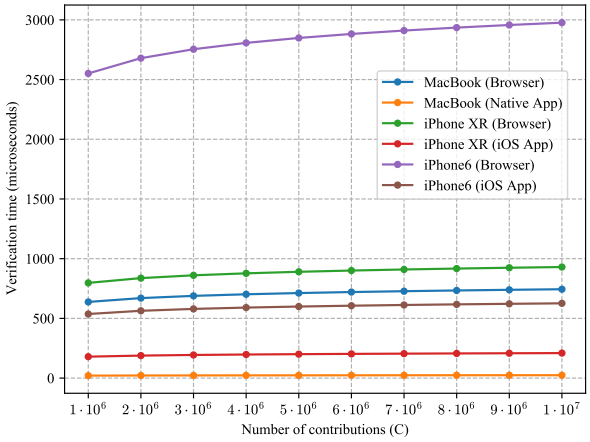

Fig. 5: Verification time of proof-of-inclusion with respect to C, which is of logarithmic growth.

Validating a proof-of-inclusion in HeadStart can be done efficiently, as it requires only $O(\log C)$ hash computation and storage. A hash function such as SHA256 can be computed in a few microseconds on commodity devices.[1] We evaluate the speed of the proof-of-inclusion verification on different platforms as C increases and confirm that the time grows logarithmically, as shown in Fig. 5. Moreover, it takes only three milliseconds to verify when $C = 10M$ on the slowest platform, a browser-based implementation on an iPhone 6. Hence, our HeadStart protocol provides high scalability in practice.

B. Verification Efficiency ($L = 1$)

We present the verification efficiency on both desktop (MacBook) and mobile devices (iPhone XR and iPhone 6) when $L = 1$ before advancing to $L > 1$ for ease of understanding. Aside from the results of native macOS and iOS applications, we also present the experiment results of our WebAssembly implementation running on the different devices' browsers. Although native applications usually perform better than browser applications, we still conduct this experiment to test HeadStart's feasibility on built-in browsers because using built-in browsers is more convenient than native applications for the contributors. In this evaluation, we choose a length of $T_{vdf} = 24h$ result-generation phase and fix C to be one million because the verification of proof-of-inclusion is relatively fast as discussed in Section VIII-A.

As shown in Table II, the results with DHPI on different platforms are about five times faster than those without this optimization. By adopting DHPI, the verification can finish within several seconds on native applications and in around a minute when using the browser on iPhone 6. Note that increasing the time parameter T of class-group VDF does not drastically increase the execution time of verification. Namely, with a reasonable length of the result-generation phase, such as several days or weeks, the contributors can always verify in around the time lengths shown in Table II if $L = 1$.

[1]It takes around 26μs to perform one SHA256 hash on an iPhone 6 with a 256-bit input.

TABLE II: Verification time of HeadStart with/without DHPI ($L = 1$, $C = 1M$, $T_{vdf} = 24h$).

Platform	*Without DHPI*	*With DHPI*
MacBook (Native App)	9.63s	1.74s
iPhone XR (iOS App)	33.36s	6.21s
iPhone 6 (iOS App)	94.60s	16.17s
MacBook (Browser)	127.37s	29.76s
iPhone XR (Browser)	143.26	30.33s
iPhone 6(Browser)	413.5199s	84.985s

However, we discovered that the browser implementation on mobile phones provides less than ideal user experience when L grows larger. First, the runtime of the verification procedure will be stalled when the browser tab is moved to the background. As a result, the contributor needs to keep his browser tab in the foreground for several minutes until the verification is completed. Second, multithreading is rarely supported in mobile browsers [47], and the only platform supporting this feature so far (Firefox57 on Android) has disabled it by default to mitigate speculative execution side-channel attacks [47]. However, in the case when $L > 1$, we can adopt parallel computing to accelerate the procedure because the verifications of L VDFs can be performed independently. Although these are some real-world technical details, we still present the experiments of browsers for future investigations. Consequently, our current suggestion of client implementation on mobile phones is to use native mobile applications.

C. Verification Efficiency ($L > 1$)

The verification's space and time overheads grow roughly linearly with the stage number L, because the complexity of the costly operations of class groups in each stage is $O(1)$ with respect to the time parameter T. In addition, the L VDF proofs can be reduced to a single proof $\widetilde{\pi}$ via proof aggregation, and we can accelerate the verification of L VDFs by parallel computing. Consider the case when the contribution phase is opened for a week. Our experiment shows that everyone can receive the result in around 1 hour and 41 minutes after the contribution phase ends by setting $L = 100$, while previous work [17], [22], [44], [45], [32], [13], [12], [52] needs to delay for another whole week. Hence, we consider this a reasonable parameter suggestion. In this evaluation, we set $T_{vdf} = 24h$ and start the parameters from $L = 100$.

The verification proof is composed of the audit path of the Merkle tree, the inputs $x_{root,j}$, outputs y_j, and proofs π_j of VDFs across L stages. As shown in Table III, our AggVDF has a much smaller proof size than ChiaVDF. Specifically, each increment in L when $L > 1$ will increase the difference of our proof sizes by roughly 2×832 bytes. We explain the reason in detail from two aspects. First, ChiaVDF has to use different discriminants with different input strings [2] because it does not implement our ClHash function. As a result, it requires $L-1$ additional space to represent those additional discriminants, which are large negative primes (832 bytes) where $2^{6655} < |d| < 2^{6656}$. Second, ChiaVDF cannot use proof aggregation because all the VDFs are in different class groups with different discriminants, and each proof is also around 832 bytes because $d = b^2 - 4ac$. Consequently, ChiaVDF requires additional space for $L - 1$ more discriminants and proofs.

TABLE III: Comparison of the Total Proof Size

L	*ChiaVDF*	*HeadStart AggVDF*
100	334 KB	169 KB
150	500 KB	252 KB
200	667 KB	336 KB
250	833 KB	419 KB

Note. This table presents the comparison of total proof size of one contributor, which includes the audit path, $(x_{root,j}, y_j)$ $\forall 1 \leq j \leq L$ and the aggregated proof $\widetilde{\pi}$.

The verification of L VDFs can be computed in parallel because the group operations can be independently executed as shown in Equation 3. To investigate how well these tested devices can perform when set to the best condition, we present the multithreading results in Fig. 6, and the hyperthreading of MacBook is enabled. The results, when given different L, are influenced by the verification of the aggregated proof (Equation 3). The contributors have to compute the group operations $g_j^{\alpha_j}$, where α_j are the SHA256 checksums in practice. Therefore, every additional L results in the execution of a large number of additional operations.

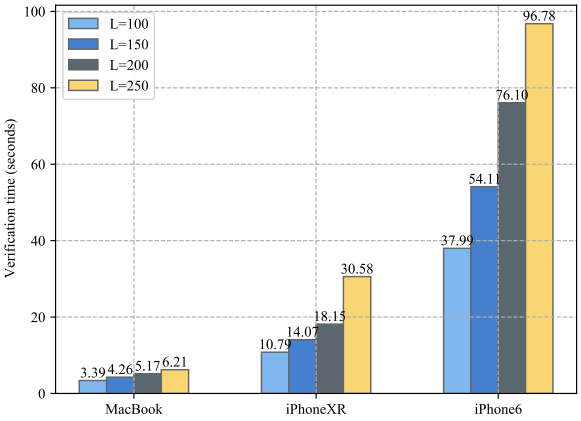

Fig. 6: Verification time on different devices

D. Influencing factors of RP Latency

The factors that might influence the length of RP latency are the time to gather contributions, build Merkle trees, publish audit paths and roots, evaluate VDFs, and publish the final result. In reality, the organizer can prefetch the contributions and prebuild the Merkle trees during the contribution phase. Besides, the audit paths and roots can be published simultaneously without delaying the procedures because VDF evaluations can be started right after their inputs are ready. The organizer simply needs to ensure that the publications will be available in time to fulfill HeadStart's requirements. The delay of the final result publication depends on the system implementation's network latency. As a result, the primary factor that we should discuss is the time variation of the VDF evaluation.

We then discuss how the variation of a VDF's evaluation time affects the RP latency. Although ideally, a VDF should run for t_{vdf} time consistently in every stage, the actual duration may be slightly different and longer than t_{vdf} because of the variation of CPU time. These short delays would accumulate in each result-generation stage and thus lengthen the RP latency. However, if the performance of the chosen VDF is stable, then the short delays would not cause a severe impact. Therefore, we present our experiment to show that the performance of our AggVDF is stable.

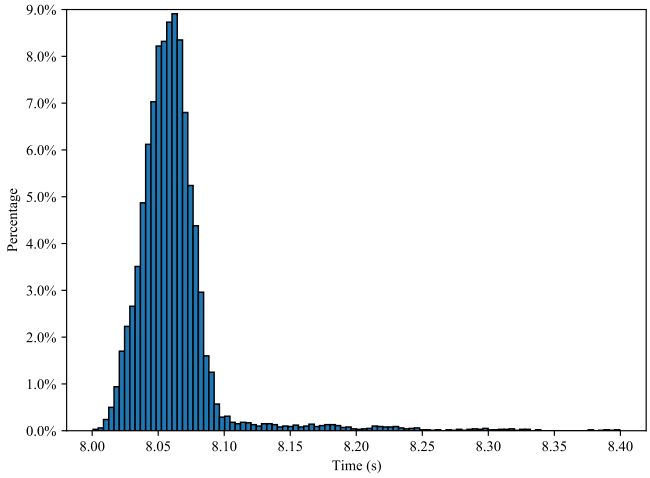

Fig. 7: Variation of the evaluation time of AggVDF when the bit-length of the discriminant $k_d = 6656$ and the number of sequential squaring $t = 83886$.

We deploy our AggVDF on the AWS EC2 instances (c5.xlarge) with AVX enabled and set its process execution priority to the highest. We collect 10,000 records and present their distribution to show the variation. As shown in Fig. 7, the range of the data is only about 0.3 seconds, where the minimum is 8.0006 seconds, and the maximum is 8.3995 seconds. Additionally, about 95% of the data are in 8.09 seconds. As a result, we demonstrate that the performance of AggVDF is relatively stable, which shows that the variations in VDF evaluation will not significantly lengthen the RP latency.

IX. Related Work

Since the introduction of coin-flipping protocols [11], [7] and randomness beacons [51], several randomness generation techniques have been proposed. Algorand [31] and Ouroboros Praos [27] applied verifiable random functions (VRFs) to compute and verify the local randomness. In these schemes, participants are responsible for computing the next randomness by combining their local randomness with the previously available randomness to produce the random result. Some work extracts randomness from Bitcoin [14], [9], [50] or public financial data [24] for the same purpose. However, these protocols are not suitable for public participation because they do not prevent malicious participants from withholding the random output until gaining advantage as described by [52], [53].

To defend against biasing attacks, there are mainly two types of strategies. One delays the inputs of the result generation, such as commitment scheme, PVSS, and threshold signature schemes. The other delays the duration of result generation, such as delay functions and VDFs. The main idea of them is to prevent an adversary from stealthily crafting a desirable result.

TABLE IV: Comparison of related work by usability and scalability properties

	Usability			Scalability	Misc.
	Verif. complexity	RP latency	No. of honest contributors	Communication complexity (as a contributor)	Cryptographic primitive(s)
Chow et al. [22]	$O(T+C)$	$O(T)$	1	$O(C)$	VRF + delay function
Liu et al. [45]	$O(T+\log C)$	$O(T)$	1	$O(\log C)$	VRF + delay function
HydRand [53]	$O(C)$	✗	$\frac{2C}{3}+1$	$O(C)$	PVSS
RandRunner [52]	$O(C \log T)$	$O(T)$	$\frac{C}{2}$	$O(C)$	VDF
HeadStart	$O(L \log T + \log C)$	$O(\frac{T}{L})$	1	$O(\log C)$	VDF

Note. We compare HeadStart with the latest work based on delay functions, PVSS and VDFs. (L = number of divided contribution stages; C = number of contributions; T = time duration of the contribution phase.)

Many recent schemes, such as Ouroboros [43], RandHound and RandHerd [55], Scrape [19], and HydRand [53], adopted PVSS to increase the system availability. By adopting such threshold cryptographic techniques, the majority of participants can recover those unrevealed secrets, alleviating the issue in traditional commitment schemes. However, these protocols require an honest majority to reveal the secret securely; one honest contributor is insufficient to guarantee their security.

To delay the duration of the result generation, several schemes [17], [22], [45] adopt delay functions or time-lock puzzles but with high verification costs. The recently proposed RandRunner [52] adopts VDF to construct randomness beacons, which requires a predefined fixed set of participants to propagate their VDF results in a round-robin manner. Its design allows an honest participant to compute his own VDF through the trapdoor in $O(\log T)$. However, if there are m malicious participants who do not propagate their results, the RP latency will be drastically lengthened by $m \times T$ because the remaining participants need to compute for a length of time T to recover each missing result.

Most of the recent schemes [19], [53], [19], [52] that we mentioned assume a predefined fixed set of participants or rely on an honest majority. One of their main applications is to generate a verifiable result for the third-party verifiers by a group of preregistered or private participants with powerful machines, because these protocols have high communication or computational overheads. However, in the scenario of public participation events, it is impractical to ask ordinary people to participate in such protocol due to the honest majority assumption. For public participation events, it is much more convincing for the public to directly contribute, like in HeadStart, so that they can easily fulfill the assumption that at least one contributor is honest, rather than asking them to believe that the system has an honest majority.

The two lottery systems proposed by Chow et al. [22] and Liu et al. [45] are much more similar to our public participation scenarios, which open for the public to participate and only assumes one honest participation. As a result, we present table IV to summarize representative related protocols: the two lottery systems [22], [45], HyRand [53], which is the latest work based on PVSS, and RandRunner, which is the latest work based on VDF. Because all schemes included in the table satisfy unpredictability and bias-resistance, we only compare them based on our desired usability and scalability properties.

X. Conclusion

We proposed HeadStart, a participatory randomness protocol designed for public participation at scale. HeadStart accepts the contributions from the public and generates unpredictable and bias-resistant results that can be efficiently verified on commodity devices without assuming other contributors are honest. The contribution phase can be sufficiently long with low RP latency to produce the result earlier. We present an algorithm ClHash, the first efficient algorithm that hashes to a class group of an imaginary quadratic field, bringing proof aggregation of class-group VDF from theory into practice. We then evaluate our implementation to show HeadStart's feasibility when running on moderate or low-end devices. Future work includes: (1) the adoption of other groups of unknown order that can provide 128-bit or higher security with smaller security parameters, and (2) integration with real-world applications, such as auditing and social-resource allocation.

Acknowledgments

We thank Jia-Chi Huo and Peng Lo for precious discussions and their comprehensive writing advice, as well as the anonymous reviewers for the valuable feedback we received. This research was supported by the Ministry of Science and Technology of Taiwan under grants MOST 109-2636-E-002-021 and 110-2628-E-002-002. This work was performed in part while Yih-Chun Hu was visiting Academia Sinica and National Chiao Tung University in Taiwan.

References

[1] "Chia network," https://github.com/Chia-Network/chiavdf, accessed on 2020-09-15.

[2] "The evaluation of chiavdf cannot use the same discriminant with different input strings." https://github.com/Chia-Network/chiavdf, accessed on 2020-09-15.

[3] "Hot lotto fraud scandal," https://www.nytimes.com/interactive/2018/05/03/magazine/money-issue-iowa-lottery-fraud-mystery.html, accessed on 2020-12-01.

[4] "Poa network," https://github.com/poanetwork/vdf, accessed on 2020-09-15.

[5] "US apologizes for visa lottery error," https://www.voanews.com/a/us-mistakenly-tells-22000-they-were-eligible-for-visas-121790664/174692.html, accessed on 2020-12-01.

[6] K. Belabas, T. Kleinjung, A. Sanso, and B. Wesolowski, "A note on the low order assumption in class group of an imaginary quadratic number fields," Cryptology ePrint Archive, Report 2020/1310, 2020, https://eprint.iacr.org/2020/1310.

[7] M. Ben-Or and N. Linial, "Collective coin flipping, robust voting schemes and minima of banzhaf values," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE, 1985, pp. 408–416.

[8] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 46, 2018.

[9] I. Bentov, A. Gabizon, and D. Zuckerman, "Bitcoin beacon," *arXiv preprint arXiv:1605.04559*, 2016.

[10] B. Blanke, "Chia vdf competition round 2 results and announcements." "https://aws.amazon.com/tw/blogs/startups/competition-forever-change-blockchain/", 2019, accessed on 2020-12-01.

[11] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *ACM SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983.

[12] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[13] D. Boneh and M. Naor, "Timed commitments," in *Annual international cryptology conference*. Springer, 2000, pp. 236–254.

[14] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.

[15] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 327–357.

[16] J. Buchmann and H. C. Williams, "A key-exchange system based on imaginary quadratic fields," *Journal of Cryptology*, vol. 1, no. 2, pp. 107–118, 1988.

[17] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017.

[18] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[19] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 537–556.

[20] Chia Network, "Chia network sets the bit-length of the security parameter of vdf k=1024," https://github.com/Chia-Network/chiavdf/blob/f4266166e5f79c8375be567fd2624ed4e1229afd/src/proof_common.h#L50, accessed on 2020-11-15.

[21] ——, "Congratulations to sundersoft for winning both tracks of the second chia vdf competition!" https://github.com/Chia-Network/vdfcontest2results, accessed on 2020-12-01.

[22] S. S. Chow, L. C. Hui, S.-M. Yiu, and K. Chow, "Practical electronic lotteries with offline ttp," *Computer Communications*, vol. 29, no. 15, pp. 2830–2840, 2006.

[23] S. Chung, "In some countries, your receipt can be a winning lottery ticket and can help the government collect sales tax," https://abovethelaw.com/2019/10/in-some-countries-your-receipt-can-be-a-winning-lottery-ticket-and-can-help-the-government-collect-sales-tax/, 2019, accessed on 2020-11-15.

[24] J. Clark and U. Hengartner, "On the use of financial data as a random beacon." *EVT/WOTE*, vol. 89, 2010.

[25] M. V. Copeland, "Time, randomness, and a 100,000 prize to forever change blockchain." "https://aws.amazon.com/tw/blogs/startups/competition-forever-change-blockchain/", 2019, accessed on 2020-12-01.

[26] R. Crandall and C. B. Pomerance, *Prime numbers: a computational perspective*. Springer Science & Business Media, 2006, vol. 182.

[27] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[28] S. Dobson, S. D. Galbraith, and B. Smith, "Trustless groups of unknown order with hyperelliptic curves." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 196, 2020.

[29] J. Drake., "Minimal vdf randomness beacon." "https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566", 2018, accessed on 2020-12-01.

[30] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 626–645.

[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[32] D. M. Goldschlag and S. G. Stubblebine, "Publicly verifiable lotteries: Applications of delaying functions," in *International Conference on Financial Cryptography*. Springer, 1998, pp. 214–226.

[33] Google LLC, "Trillian, a transparent, highly scalable and cryptographically verifiable data store." https://github.com/google/trillian, accessed on 2020-11-15.

[34] J. L. Hafner and K. S. McCurley, "A rigorous subexponential algorithm for computation of class groups," *Journal of the American mathematical society*, vol. 2, no. 4, pp. 837–850, 1989.

[35] S. Hamdy and B. Möller, "Security of cryptosystems based on class groups of imaginary quadratic orders," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 234–247.

[36] R. Han, J. Yu, and H. Lin, "Randchain: Decentralised randomness beacon from sequential proof-of-work," Cryptology ePrint Archive, Report 2020/1033, 2020, https://eprint.iacr.org/.

[37] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.

[38] Harmony Project, "Harmony project using k=2048," https://github.com/harmony-one/vdf/blob/620379da88498fe40babc744685253aade77e8a2/src/vdf_go/vdf.go#L13, accessed on 2020-12-03.

[39] B. He and Y. Wei, "Electronic sortition," in *Proceedings. The 2009 International Symposium on Intelligent Information Systems and Applications (IISA 2009)*. Citeseer, 2009, p. 203.

[40] J. Hendricks, "Manatee county explains vaccine distribution protocols," https://thebradentontimes.com/manatee-county-explains-vaccine-distribution-protocols-p22480-158.htm, accessed on 2021-07-04.

[41] S. Jaques, H. Montgomery, and A. Roy, "Time-release cryptography from minimal circuit assumptions." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 755, 2020.

[42] M. S. Kearney, P. Tufano, J. Guryan, and E. Hurst, "Making savers winners: An overview of prize-linked savings products," National Bureau of Economic Research, Tech. Rep., 2010.

[43] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[44] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx," Cryptology ePrint Archive, Report 2015/366, 2015, https://eprint.iacr.org/2015/366.

[45] Y.-N. Liu, H.-G. Liu, L. Hu, and J.-B. Tian, "A new efficient e-lottery scheme using multi-level hash chain," in *2006 International Conference on Communication Technology*. IEEE, 2006, pp. 1–4.

[46] B. Mark Friedenbach, Kalle Alm, "Fast merkle trees in bitcoin," https://github.com/bitcoin/bips/blob/master/bip-0098.mediawiki, accessed on 2020-12-22.

[47] MDN contributors, "Atomics document on mdn web." https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Atomics, accessed on 2020-11-15.

[48] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*. Springer, 1987, pp. 369–378.

[49] K. T. Paul P. Murphy, Rosa Flores and C. Sara Weisfeldt, "Florida county commissioner limited vaccine drive to the two richest zip codes and then created a vip list," https://edition.cnn.com/2021/02/18/politics/manatee-county-vaccine/index.html, accessed on 2021-07-04.

[50] C. Pierrot and B. Wesolowski, "Malleability of the blockchain's entropy," *Cryptography and Communications*, vol. 10, no. 1, pp. 211–233, 2018.

[51] M. O. Rabin, "Randomized byzantine generals," in *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 1983, pp. 403–409.

[52] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness," in *28th Annual Network and Distributed System Security Symposium, NDSS*, 2021.

[53] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Efficient continuous distributed randomness," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 73–89.

[54] J.-P. Serre, *A course in arithmetic*. Springer Science & Business Media, 2012, vol. 7.

[55] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 444–460.

[56] B. Wesolowski, "Efficient verifiable delay functions," *Journal of Cryptology*, pp. 1–35, 2020.

[57] World Health Organization, "Ethics and covid-19: resource allocation and priority-setting," https://www.who.int/ethics/publications/ethics-covid-19-resource-allocation.pdf?ua=1, 2020, accessed on 2022-01-15.