Dike: Practical Decentralized e-Voting System with Voter Demographic Obfuscation

ABSTRACT

In several places, (*e.g.* South Asia), elections happen on ethnic lines (*i.e.* caste, religion, community *etc.*). Often voters are loyal to parties that seem to favor their native communities (*a.k.a.*, *votebank*). When parties ally, their voter bases, consisting of voters from different communities, also merge. For the alliance to win, the loyalists should have a majority. Rivals usually aim to disrupt such alliances, by luring the smaller of the constituent voter groups. This is legal as long as no party (or alliance) has access to private information about the individual voters and their voting patterns, and merely relies on regional surveys and peaceful outreach.

However, recent reports suggest that political parties have access to (historical) electoral data (especially information on who *actually* voted), that they abuse for electoral fraud, targeted campaigns, and even tactics like deleting voters' names from electoral rolls. Existing e-voting systems are not geared towards preserving voters' anonymity. Further, their processes to ensure vote accountability, also inadvertently compromise the voters' anonymity.

Dike, is a "first-of-its-kind" decentralized e-voting scheme that strives to obfuscate voters' demographics and provide mechanisms for anonymous vote accountability. It is based on the Ethereum (PoW) blockchain and relies on ring signatures to preserve voters' anonymity, which is not compromised even when voters seek accountability for their votes. Additionally, Dike relies on Damgård and Jurik's threshold encryption to prevent preemptive vote count, which parties could use to identify the potential winner (and thus influence voters). We gauged Dike's scalability in a controlled lab environment where it took 10 hours to cast about 10M votes.

KEYWORDS

E-voting, demographic anonymity, privacy

1 INTRODUCTION

Voting to collectively decide the people's representative is the lifeblood of democracies. Voting has taken different forms in the 21st century. While some rely on technology, others don't. In recent decades, electronic voting systems, *i.e.* those that rely on technology to aid the process of casting and counting votes, have gained traction [1–3]. Such systems can be traced back to Chaum's proposals [4–6], that focussed on anonymizing the votes casted. Several governments are gradually considering (if not adopting) e-voting to solve many problems common with offline processes. Being able to vote, even when one moves beyond one domicile (*i.e.,* constituencies) in search of improved life opportunities is one such major problem. Others include politically motivated violent crimes that could intimidate voters and prevent them from visiting the polling station (and/or booth capturing [7]). Electronic voting systems by-and-large attempt to solve such problems, besides reducing the human labor and financial costs involved in conducting fair and peaceful voting and counting processes.

In this paper, we focus on the electoral process of regions with heterogeneous communities and ethnic groups, many of which have a multi-party electoral democracy like in several South Asian countries. In such regions, it is often observed that voters belonging to a particular community, often vote for a candidate belonging to their own community(*i.e. vote bank* [8]). As a result, the winning candidate of a constituency often belongs to the majority community/ethnicity. Often a significant number of voters from other communities also vote for the same candidate, when political parties (often representing different communities) ally. Rivals target such heterogenous voter groups through targeted campaigns to lure subgroups, smaller among them, persuading them to switch loyalties. This is fair game and democratic, assuming none of the parties resort to unlawful practices (*e.g.* using hateful speeches).

However, this system is not without its shortcomings. Several anecdotes and studies point to electoral frauds [9, 10] and the availability of electoral data to manipulate voters [11–13]. While democratic elections rely on secret ballots, the ethno-demographics of the voting population, those who *actually* voted, and the subsequently winning candidates (and their community), make it relatively easy to guess who may have likely voted for whom (since in several parts of South Asia, last names reveal a person's ethinicity[14] *etc.*). Rival parties can now leverage such data to enhance their targeted campaigns to try and divide the heterogeneous voter demographies and usurp votes of relatively smaller subgroups. Obfuscating such demographic data, to prevent such abuse, seems vital and non-trivial. It is noteworthy, that while voters' lists, bearing the names of eligible voters in a given constituency, are publicly shared, the data about who all *actually* voted is closely guarded. This paper focuses on thwarting efforts to surreptitiously leak such data.

Further, regardless of the apparent fair process, the vote-counting process is largely opaque. The voters have no way to make sure that their votes have been counted or not. Ensuring this must also conceal the voters' identities while preserving ballot secrecy.

Several online processes have been proposed over the years that resolve the existing challenges of offline systems. However, several problems still remain unaddressed. *E.g.* most require authenticating the voters, and in-turn learning about their identities, and thus the demographic information (much like the offline process). Further, most strive to provide vote accountability but then end up sacrificing the voters' anonymity. Finally, existing online systems have no provisions to conceal the vote counts, while the voting is still underway. Such intermediate vote count may be used to predict the probable winner and thus influence the voters.

To ameliorate such problems, we propose a novel practical e-voting system, we call *Dike*, that primarily focuses on solving three vital problems. *Firstly*, Dike obfuscates the electoral demographics that could be obtained from election authorities through coercion by political groups, who could abuse it. *Secondly*, it provides anonymous vote accountability to win the voters' trust, by allowing them to check whether their vote was rightly counted, *without* inadvertently unveiling their identities. *Thirdly*, Dike prevents intermediate vote counts, before the elections conclude. Additionally, like several other e-voting systems, Dike also satisfies requirements like voter authentication, vote-casting confidentiality and privacy (*i.e.* secret ballot), prevention of double vote casting, vote self-tallying, resistance against coercion to reveal whom a voter has voted for, scalable performance, communication security *etc.* Blockchains, due to the inherent decentralization, distributed attestation and double spending detection, distributed ledger (storage) *etc.*, are most suited for e-voting schemes including Dike.

Dike relies on several schemes, working in tandem, to fulfill the aforementioned requirements. It relies on *ring-signatures* [15] and *cancelable biometrics* [16, 17] to authenticate the voter before the election commences, *and* at the time of actually casting the vote, respectively. Further, it also relies on Damgard-Jurik's *threshold-based encryption/ decryption* [18, 19], which serves multiple purposes. The scheme requires the collusion among multiple parties (*i.e.*, the majority among a group) to decrypt the votes, thereby preventing a single entity from decrypting the votes before the elections conclude. Additionally, vote counting does not require decrypting them individually; the cumulative vote count can be obtained through a homomorphic addition operation.

We implemented and tested Dike using a private Ethereum blockchain setup in a controlled setup. To simulate real-world scenarios consisting of voters and miners that actually cast votes and mine blocks, we ran Dike on a testbed consisting of machines (acting as voters and miners) connecting over the Internet. The performance and scalability of Dike, was gauged by casting 1M votes in parallel. The testbed comprised of 44 miners, spending their computational power in mining these transactions. It took around 10 hours to cast 1M votes with 120 parallel sessions and is likely scalable to cater even larger population.

Dike also employs measures to safeguard against various types of adversaries (both internal and external). We elaborate on this in the subsequent sections of the paper, along with highlighting some important design considerations (Sec. 4.3) and challenges (Sec. 6.2). To summarize, the following are the contributions of this work:

(1) Dike, a unique e-voting system, that safeguards against offline voting problems, commonly seen (or are feared to happen) in regions with diverse ethnicities (*e.g.*, South Asia). In particular, the system strives to tackle three important problems.
    (a) Collusion between political parties (and/or ruling dispensations) and election authorities, leading to sharing of demographic information of the voters, which the former could leverage to usurp votes targetting specific votebanks.
    (b) The vote-counting process is mostly opaque and voters have no way to ascertain that their votes were accounted for. Prior efforts have attempted to solve this, but their processes usually end up disclosing the identities of the voters (sometimes even compromising the ballot secrecy).
    (c) Existing online voting systems do not present mechanisms to prevent the counting of votes well before the elections conclude. Malicious insiders (*e.g.* among those who are responsible for tallying the votes), could determine who is likely going to win, and thus accordingly help parties influence voters. Offline voting is mostly devoid of this issue due to the clear separation between the vote-casting and counting processes, along with authoritative overwatch.
(2) The design and implementation of Dike, which uses a private Ethereum blockchain. Dike was tested in a controlled testbed with miners and voters communicating over the Internet. We emulated the voting process (as described in Sec. 4.2) and tested with over a million votes (*i.e.*, the average voting population of one of the largest constituency in South Asia [20]). Dike performed well in such a testbed taking around 10 hours to cast the votes. Also, the *difficulty* of 316000 and *hashrate* of 152MH/sec achieved, is comparable to that of other public mining cyptocurrency such as Monero, Ethereum Classic.
(3) Finally, we address various implementational, deployment and security challenges that could be encountered when deploying Dike. We provide critical security analysis of Dike and the appropriate design choices to thwart various threats.

2 BACKGROUND AND RELATED WORK

2.1 Relevant Cryptographic Schemes

2.1.1 ***Threshold Access Structures.*** A class of threshold access structures (TAS) is defined in the following way.

*Definition 2.1 (Threshold Access Structures (*TAS*)).* For a set of n players $\mathcal{P} = \{P_1, \dots, P_n\}$ an access structure $\mathbb{A}_t$ is called *threshold access structure* if for every set of players $\mathcal{S} \subset \mathcal{P}$, $\mathcal{S} \in \mathbb{A}_t$ if and only if $|S| \geq t$. Define TAS to be the class of all access structures $\mathbb{A}_t$ for all $t \in \mathbb{N}$.

2.1.2 ***Threshold Encryption Schemes.*** Let λ denotes the security parameter and v denotes the number of players involved. Then *threshold homomorphic encryption scheme (*ThHE*)* is defined as:

*Definition 2.2 (Threshold Homomorphic Encryption Scheme (*ThHE*)).* Let $\mathcal{P} = \{P_1, \dots, P_v\}$ be a set of v players and let $\mathbb{A}$ be a class of efficient access structures on $\mathcal{P}$. A *threshold homomorphic encryption scheme (*ThHE*)* Π = (ThHE.Setup, ThHE.Encrypt, ThHE.Eval, ThHE .PartDec, ThHE.FinDec) for $\mathbb{S}$ is a 5-tuple of PPT algorithms defined as follows.

(1) $(\mathrm{pk}, \mathrm{sk}_1, \dots, \mathrm{sk}_v) \leftarrow \mathrm{ThHE.Setup}(1^\lambda, \mathbb{A})$**:** On input the security parameter λ and an access structure $\mathbb{A}$, it outputs a public key share pk, and the set of secret keys $\mathrm{sk}_1, \dots, \mathrm{sk}_v$ for the players $P_1, \dots, P_v$, respectively.
(2) $c \leftarrow \mathrm{ThHE.Encrypt}(\mathrm{pk}, m)$**:** On input the public keys pk and a message $m \in \mathcal{M}$, the algorithm outputs a ciphertext c.
(3) $\widetilde{c} \leftarrow \mathrm{ThHE.Eval}(\mathrm{pk}, c_1, \dots, c_k)$**:** On input the public keys pk and k ciphertexts $c_1, \dots, c_k$, the evaluation algorithm outputs a ciphertext $\widetilde{c}$, which is a function of the ciphertexts $c_1, \dots, c_k$. For example, for an additively homomorphic encryption scheme this function could be the product of the ciphertexts $c_1, \dots, c_k$, i.e., $\widetilde{c} = \prod_{i=1}^{k} c_i$.
(4) $p_i \leftarrow \mathrm{ThHE.PartDec}(\mathrm{pk}, c, \mathrm{sk}_i)$**:** On input the public keys pk, a ciphertext c and P_i's secret key sk_i, the partial decryption algorithm outputs a partial decryption p_i of the player P_i.
(5) $m \leftarrow \mathrm{ThHE.FinDec}(\mathrm{pk}, c, B)$**:** On input the public keys pk and a set $B = \{p_i : i \in S \text{ for some } S \subseteq \mathcal{P}\}$, the final decryption algorithm outputs $m \in \mathcal{M} \cup \{\perp\}$, where $\perp$ means that the final decryption is invalid. Note that the final decryption may be an invalid decryption if $S \notin \mathbb{A}$.

ThHE must satisfy the following notions of correctness and security.

Definition 2.3 (Evaluation Correctness). A ThHE satisfies *evaluation correctness* if for all security parameter λ, access structure $\mathbb{A}, S \in \mathbb{A}$, and $m_i \in \mathcal{M}$ for $i \in [k]$, the following condition holds. For $(\text{pk}, \text{sk}_1, \ldots, \text{sk}_v) \leftarrow \text{ThHE.Setup}(1^\lambda, \mathbb{A})$, $c_i \leftarrow \text{ThHE.Encrypt}(\text{pk}, m)$ for $i \in [k]$, $\widetilde{c} \leftarrow \text{ThHE.Eval}(\text{pk}, c_1, \ldots, c_k)$, $\Pr[\text{ThHE.FinDec}(\text{pk}, \{\text{ThHE.PartDec}(\text{pk}, \widetilde{c}, \text{sk}_i)\}_{i \in S}] = f(m_1, \ldots, m_k)] = 1 - \text{negl}(\lambda)$, where f denotes the underlying homomorphic function. **E.g.** if it is additively homomorphic then $f(m_1, \ldots, m_k) = \sum_{i=1}^{k} m_k$.

2.1.3 ***Signatures***

In public-key cryptography *digital signatures* are used to provide *authenticity (or integrity)*. A *signer* authenticates a message m by signing it using his private key. Anyone, *i.e.*, a *verifier*, verifies it using the public key. The signer generates it's public-private key pair (pk, sk) (using the *Gen* algorithm defined below). Using the *Sign* algorithm, a message m, is signed with the secret key sk, and a signature σ is generated Finally, the *Verify* algorithm uses pk, and checks if σ was derived from m. The objective of an adversary would be to launch an *existential forgery* attack [21], whereby they can generate a valid (m', σ'), without knowing sk.

Definition 2.4 ((Digital) Signature Scheme [22, Definition 12.1]). A *(digital) signature scheme* consists of the following three probabilistic polynomial-time algorithms (PPT) (*Gen*, *Sign*, *Verify*).

(1) The key-generation algorithm *Gen* takes as input a security parameter 1^n and outputs a pair of keys (pk, sk). These are called the *public key* and the *private key*, respectively.
(2) *Sign* takes as input a private key sk and a message $m \in \mathcal{M}$, where the message space $\mathcal{M}$ may depend on pk. It outputs a signature σ; *i.e.*, $\sigma \leftarrow \text{Sign}_{\text{sk}}(m)$.
(3) The verification algorithm *Verify* takes as input a public key pk, a message m, and a signature σ. It outputs a bit b. $b = 1$ denotes a valid, and $b = 0$ denotes an invalid signature. We denote this as, $b := \text{Verify}_{pk}(m, \sigma)$.

Correctness: It is required that except with negligible probability over (pk, sk) output by $\text{Gen}(1^n)$, it holds that $\text{Verify}_{pk}(m, \text{Sign}_{sk}(m)) = 1$, for every (legal) message $m \in \mathcal{M}$.

A signature scheme is considered secure if it can defend against forgery. These are usually modeled either as a (strong) unforgeability game against a chosen message attacker(UF-CMA), or as a much stronger notion of existential unforgeability game against a chosen message attacker(EUF-CMA).

2.1.4 ***Ring Signatures***

Ring signatures are constructs designed for members of a group to sign messages to authenticate their association to the group, *without* revealing their identities [15]. *Ring signatures* were first proposed by Rivest *et al.* in 2001 [23]. These can be seen as a generalization of *group signatures* [24].

Unlike group signatures, where the group manager has an overarching powers and can revoke the anonymity of misbehaving signers, ring signatures have no group managers, no setup procedures, no revocation procedures, and require no coordination. Any ring member of the ring can choose any set of possible signers that including themself. The member can then sign any message using its secret key and the public keys of the other members that the signer selected, without the need for seeking approval/assistance of those members. The ring member who signs using their secret key is called the *signer*, while all others are *non-signers*.

Definition 2.5 (Ring Signature Scheme [23]). Assume a ring of n signers $\mathcal{S}_1, \ldots, \mathcal{S}_n$ each having their public keys $\text{pk}_1, \ldots, \text{pk}_n$, respectively. A *ring signature scheme* is then defined by the following two procedures $\Pi = (\textit{RingSign}, \textit{RingVerify})$.

- ***RingSign***$(m, \text{pk}_{i_1}, \ldots, \text{pk}_{i_r}, s, \text{sk}_s)$**:** Takes as input the public keys of any r distinct ring members, *i.e.*, $\text{pk}_{i_1}, \ldots, \text{pk}_{i_r} \in \{\text{pk}_1, \ldots, \text{pk}_n\}$, together with the secret key of the s^{th} member, where $s \in \{i_1, \ldots, i_r\}$ and the message m to produce a ring signature σ.
- ***RingVerify***(m, σ)**:** Takes as input the message m and the signature σ that bears the public keys $\text{pk}_{i_1}, \ldots, \text{pk}_{i_r}$. The verification then outputs 1 if σ is a valid signature of m produced using the public keys $\text{pk}_{i_1}, \ldots, \text{pk}_{i_r}$ and the secret key sk_s for some $s \in \{i_1, \ldots, i_r\}$. Otherwise it outputs 0.

Ring signatures have the following properties:

(1) Ring signature schemes are *setup-free*, *i.e.*, a signer does not require the consent of all the other ring members before using their public-keys while generating a signature.
(2) Ring members can choose whatever ring sizes they desire.
(3) Ring-signatures are *signer-ambiguous*, *i.e.*, the verifier is unable to individuate and identify the signers, with a probability $>1/r$.

Dike uses the RSA based ring-signature scheme proposed by Rivest [23] (see Appendix B for details). This scheme can be shown to provide *unconditional anonymity*. Even a computationally unbounded adversary with unrestricted access to number of chosen-message signatures produced by the same ring member, cannot guess the identity of the signer with any advantage. Therefore cannot link the additional signatures to the same signer.

2.2 Background

2.2.1 *Evolution of E-Voting Systems:*

Electronic voting systems rely on computer-aided methods to facilitate the voting process. The motivation behind their designs is ensuring efficient and accurate vote casting and counting. To realize this, specially designed Direct Recording Electronic (DRE) machines [25–28] were devised. Naturally, a lot of challenges manifested. The integrity of votes and the voting processes rely on the resilience of these machines to prevent various kinds of physical and electronic threats.

Online e-Voting Systems: With time voting processes shifted to the adoption of online system to cater to issues like migratory populations. It was during this time, the requirements of online voting systems were consolidated.

- *Voter privacy or anonymity:* After a voter has been authenticated, no credentials should be recorded which can establish a link between the voter and the vote that was casted.
- *Ballot secrecy:* At no point, should it be possible to prove whom the voter voted for. This would prevent *vote-selling* [29].
- *Voter authentication:* Only eligible citizens shall be allowed to vote and that too from their respective voting residence [30].
- *Voter auditability/self-tallying:* It should be possible to tally the number of people who voted with the actual votes casted.
- *Vote accountability:* After the voting ends, the voter shall be able to verify whether their vote was correctly counted.
- *Partial vote counting:* It should not be possible for anyone ("insider" or "outsider") to get the intermediate count of votes during the voting process to know who might potentially be winning. Henceforth, we refer to this as *preemptive vote count*.

- *Scalability:* The system should work correctly and efficiently, catering to a large voting population, incurring reasonable deployment costs.

Such systems have reportedly suffered from DoS attacks (*i.e* network flooding) and vote-selling.

Cryptographic e-voting systems: In 1981, Chaum [31] presented a solution that laid the foundation of cryptographically verifiable e-voting systems. He proposed [4–6] the idea of encrypting the votes and the return receipts to build secure e-voting systems, with the hope of gaining the trust of voters and observers. But, the election processes were still biased, as the control was with a handful of election authorities who could manipulate the election outcomes. This gave way to decentralized voting systems.

Decentralized Voting Systems: Blockchains [32] have been used to decentralize many traditionally centralized systems. Their inherent properties like pseudonymity, transparency, provenance, cryptographic verification *etc.* are naturally suited for distributed e-voting solutions. Such systems have been adopted by some countries (on a trial basis) [33–35] to conduct elections. However, hitherto such systems have largely ignored electoral frauds involving voter data manipulation to usurp votes [11–13], which have surfaced in multi-party multi-ethnic regions, like those in South Asia. We now elaborate on such problems.

2.2.2 *Voter-demographic data abuse:*

In several multi-ethnic/multi-party democratic countries, voters often vote for candidates whom they ethnically identify with. Thus, parties often tailor their campaigns for their chosen/favored/loyal ethnic groups (based on religion, caste [14], linguistic groups *etc.*), also known as a *votebank* [8]. Now, consider a region (in a democratic country) where on average 60% eligible voters cast their votes every cycle. The winning party would thus require receiving a little over 50% votes of the said 60% regular voters, *i.e.* voted by $\approx$ 30.6% of the eligible voters. To defeat the winner, the opponents often try to spot distinct communities/ethnicities that make up the said 30.6%, and possibly usurp votes of the smaller subgroups (usually through targeted campaigns). Alternatively, the opponents (losers) would try to rework their campaign and pitch to target the 70% eligible voters who did not vote for the winner (including those who did not vote at all). Fears regarding such unfair practices are not unjustified [36, 37].

2.3 Related Work

The basic features of blockchain such as users' anonymity, cryptography, provenance, transparency, decentralization [38] aligns perfectly with the requirements of an online voting system. In the past researchers have extensively explored various aspects of such systems. We now discuss the existing decentralized e-voting systems based on they have been tested and deployed.

Systems tested in controlled (lab) environment: *OpenVoteNetwork* [OPN] [39] was one of the earliest to propose a decentralized voting system using Ethereum. They provide voter privacy by decoupling their real identity from their voting keys. But, it had a mandate that all registered voters must cast the vote and cooperate for vote counting, which generally is infeasible in real scenarios.

Later in 2018, *BroncoVote* [BV] [40], a university scale voting system, focused on providing voter privacy, vote auditability and voter authentication was designed. But, it was tested with just 35 voters. In the same year, Hardwick *et al.* [41] used a bit commitment protocol for casting votes, that mandated the involvement of voters (to present sealed commitments) during the counting phase. This does not scale in practice for large-scale voting. They tested their system with merely 10 voters. They also consider voters as nodes, and make an assumption that all are available for mining.

WeVoting [WV] [42] focused on providing user anonymity and a verifiable vote-counting mechanism. They claim to be resistant against preemptive vote count attacks. But similar to OpenVoteNetwork, the design required the cooperation of all the voters to obtain the final vote count. This is infeasible for large voting populations. It was tested only for 100 voters.

To the best of our knowledge, these systems have fairly managed to achieve most of the requirements of any voting system, albeit often with *a few* unrealistic assumptions. Moreover, the designs have been evaluated for small populations, *e.g.* for student council elections, corporate board member nominations *etc.* They often sidestep genuine concerns of threats like DDoS attacks, eavesdropping, database manipulations *etc.*[43].

Systems deployed in real scenarios: *Voatz* [VZ] [33] was one of those systems that were deployed and tested in real elections. It was deployed in 2016 in Massachusetts for student senate elections. Later, it was also deployed as a pilot project for West Virginia's mid-term elections. Voatz uses blockchain to store the casted votes, biometrics to authenticate the user, Mixnet to provide voter anonymity, and encryption to provide end-to-end verifiability.

Democracy's Live *OmniBallot* [35], another e-voting system had been deployed in several U.S. elections. In addition to embodying the properties of Voatz, Ominballot also provided three modes of operation, *viz. online blank ballot delivery* (downloading blank ballot, marking it manually and returning it back in the physical form), *online ballot marking* (marking the ballot online, downloading the ballot and returning it the physical form), and *online ballot return* (marking and submitting ballot online).

Like several other systems that have been used in real scenarios, these too are proprietary systems lacking formal detailed documentation. Thus, in 2020, Specter *et al.* [44, 45] reverse-engineered these to spot vulnerabilities. The authors observed that these systems are prone to database (bearing voter information, authentication data, *etc.*) compromises. Further, the return receipts, required for vote accountability, neither preserve voters' anonymity nor are they completely secure against server compromises that could force the generation of fake receipts.

Agora [AG] [34], a Swiss product was another blockchain based e-voting system used in Sierra Leone during the 2018 presidential elections. Though it was successfully deployed, it failed to ensure some grave requirements. It sought voters' credentials at the time of vote casting, which, if stolen, could be used by miscreants to cast votes on the victims' behalf. Secondly, these credentials could be also used to avail other public services services that require similar credentials (*e.g.* medical benefits)[46]. Finally, when used in countries like those in South Asia, could easily reveal the voters' demographics which may be abused by parties with vested interests.

Another system *Polys* [PS] [47], designed by Kaspersky Lab was deployed in several student council polls. It satisfies several requirements of any online voting system. But fails to ensure the privacy of voters in the real sense. It maintains a database that

Table 1: Comparison of online voting requirements as achieved by different blockchain based approaches. (Note that '?' indicates that the system design may support but authors make no claims about it.)

	VZ	AG	PS	FMV	WV	BV	OPV	**Dike**
Voter anonymity	✗	✓	✓	✗	✓	✓	✓	✓
Ballot secrecy	✓	✓	✓	✓	✓	✓	✗	✓
Voter auditability	✓	✓	✓	✓	✓	✓	✓	✓
Voter authentication	✓	✓	✓	✓	✓	✓	✓	✓
Vote accountability	✗	✓	✗	✗	✓	✓	✓	✓
Barring preemptive vote count	✗	✗	?	✗	?	✗	✗	✓
Demographics obfuscation	✗	✗	✗	✗	✗	✗	✗	✓
Anonymous vote accountability	✗	✗	✗	✗	✗	✗	✗	✓
Scalability	?	?	?	?	✗	✗	✗	✓

maps the voter's real identity with a pseudonymous identity. While suitable for the purpose it was designed for, would not benefit scenarios that this paper focuses on.

FollowMyVote [FMV] [48], strives to achieve voter anonymity, vote accountability and vote auditability. It use ElGamal cryptosystem for threshold-based vote encryption and decryption. In this, the voter is required to generate a random number to derive the vote encryption key. Hence voters can prove whom they have casted votes for. This could enable the voters to sell their votes.

To summarize this, the performance and scalability benchmarks for such systems are scant. Those deployed in the wild, like Voatz and Agora, are proprietary and present no means to verify if they live up to their claims. Some require the voter to generate their own keys for encrypting the votes. In such cases, the voters may reveal these to adversaries willfully (to sell votes or under coercion), thereby proving whom they have voted for. Further, in these systems, voters can tally their votes by traversing the blockchain containing the votes. However, these systems require saving the mapping between the real and pseudonymous identities, in centrally controlled databases. These could be compromised or their administrators coerced, thereby deanonymizing the voters. Dike strives to fill such gaps. It provides end-to-end voter anonymity and anonymous vote accountability while also being scalable to handle over 1M votes (*i.e.*, the voting population of large constituencies in South Asia [20]). Table 1 summarizes the properties of Dike, juxtapositioning it with those of others'.

3 PRELIMINARIES

3.1 Entities in Dike

This section lists all entities involved in the functioning of *Dike* along with their roles.

Voters: *Voters*, include individuals of a constituency who have voting rights. They exercise such rights to choose political and administrative legislators of their respective constituencies. The voters' primary task is to login into the system and cast the vote. The system assumes that voters are literate enough to use any electronic device to cast the vote electronically.

Miners: *Miners* are legitimate voters who are willing to offer their computational power for validating the transactions (*i.e.*, the votes). Our model assumes that miners receive incentives (as discussed in section 6.2) in exchange of their service.

Central Authority (CA) - *Central Authority* is a group of people entrusted to ensure the smooth working of the voting process. They are responsible to initialize and commence the voting process. CA is bound by the judiciary system and accountable for its actions to any problems/issues being reported.

EA's Decryption Committee (EDC): The *EDC* comprises of individuals chosen specifically from diverse institutions like the judiciary, foreign observers, members of the opposition and ruling parties, election commissioners *etc.* This should make the coercion (by authoritative regimes) of committee members to engage in unethical behavior (like partial) vote count difficult. The process of forming the EDC could in itself be initiated by independent individuals/groups like the chief justice of a country.

3.2 Threat Model and Attacks

The aim of electronic voting systems is to not only ease the process of conducting elections but also to prevent various kinds of malpractices associated [7, 11]. Thus, while several of those malpractices could be easily avoided using prior proposals [33, 34], there are several that persist. Dike, is designed to mitigate the important ones among those, *viz.* abusing voters' demographic data by motivated parties to usurp votes, deliberately excluding votes of a particular group of people, and pre-emptively counting votes while elections are still underway, so as predict who is likely winning (and thus influencing voters). To that end, we now describe the various adversaries and their capabilities.

Voters: They electronically cast votes and can act maliciously by:

- Authenticating into the system using fake biometrics, also referred as *spoofing attack* [49, 50] .
- Casting vote on others's behalf, *non-repudiation attack* [51]
- Voter may falsely claim that his vote has not been registered or may claim that his vote is not correctly recorded. Such claims are commonly known as *hoax attacks* [52].
- Casting multiple votes[1], *i.e. ballot stuffing attack* [53].

CA and EDC: The CA and/or the EDC are responsible for smooth functioning of voting process, but could act maliciously by:

- Decrypting the votes while being cast, to obtain a partial vote count, well before the electoral processes conclude. This could be used to determine who is *likely* going to win. We refer to this as the *preemptive vote count attack.*
- Surreptitiously deleting names of voters from the electoral lists [54, 55], thereby preventing them from casting their votes. We collectively term such malicious efforts as *voter deletions.*

Miners: The miners select transactions from the transaction pool, verify and append them to the blockchain. The miners may collude to launch *Sybil attacks* [56] as follows:

- Withholding blocks to get greater incentives, *i.e.*, *selfish mining attack* [57, 58].
- Colluding with fellow miners and dropping transactions from being recorded in the blockchain, *i.e. 51% attack* [58].

[1]When only one vote is allowed

Apart from these adversaries, there also exist threats from external non-system entities, whose primary objective is to cripple the system, often at the behest of political parties. Such attacks could be launched to dissuade people from using such systems (who may later demand reversion to older offline processes). Additionally, the attackers may still be able to exploit inadvertently revealed vulnerabilities, and access voters' demographic data. To summarize, these are some of the additional threats to the system:

- Eavesdropping on sensitive information by snooping on (i) the voters' traffic, bearing casted votes (ii) the traffic broadcasted among miners (bearing the transactions), to gather voter and/or voting-related information.
- Generating fake or redundant voting transactions. The adversary may attempt to use fake (invalid) token and/or biometric samples to record transactions in the system. Alternatively, they may attempt to generate duplicate transactions using previously used token-biometric combinations. This may lead to corrupt the vote-counts. This also taxes the overall system as miners expend computational resources to mine such transactions. We refer to this as the *fake transactions attack*.
- Flooding the system with random traffic to congest the network channels *i.e. packet-flooding attack*. An external adversary may attempt to throttle the system or the blockchain-based network by generating a lot of spoofed packets.
- Compromising the database(s) maintaining voting data *i.e. database attack*. An adversary may attempt to get the voter's credentials saved in the voting database. In another scenario, (s)he may try to remove voters' information from the database.

4 SYSTEM DESCRIPTION

While several decentralized voting systems has been proposed in the past [33, 34, 40, 41], that are most proprietary, with only a few being tested in realistic scenarios. Scrutinizing these systems helps gauge if they majorly satisfy all the requirements for e-voting. But to the best of our knowledge, none try to hide the voter demographics while also ensuring anonymous vote accountability and preventing preemptive vote counting. We start by formalizing Dike's requirements, in addition to those of existing e-voting systems 2.2.1. This would be followed by our detailed system design.

Demographics privacy: The demographics of the voter must not be revealed anytime during or after the process. In other words, there shall be no way anyone (overseeing/managing parties) can find whether a particular voter casted the vote or not. A voter must however still be authenticated.

Anonymous vote accountability: The system should allow a voter to verify whether their vote is rightly recorded while still concealing their true identity. However, the design should also prevent maliciously acting voters from abusing this anonymity and getting malformed votes counted.

Preemptive vote counting: The system should prevent the intermediate vote count before the elections conclude, such that an adversary cannot abuse the information about the "potential winner", by influencing voters.

4.1 Assumptions

Technical:

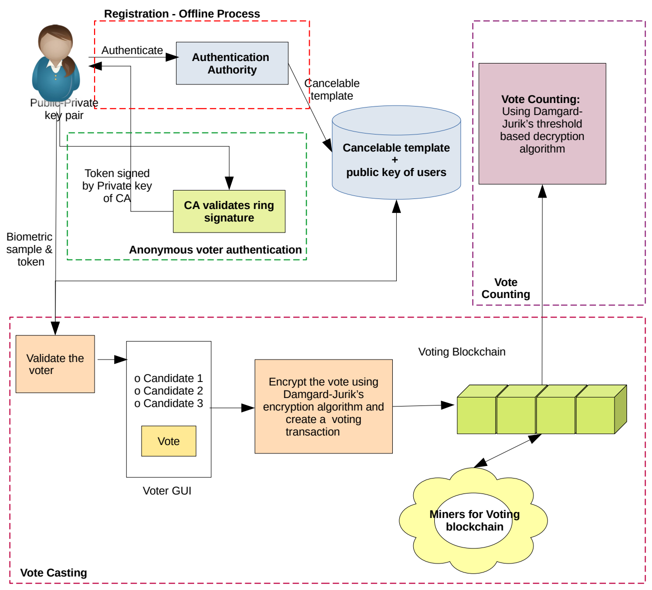

Registration - Offline Process
Authenticate
Authentication Authority
Public-Private key pair
Cancelable template
Cancelable template + public key of users
Vote Counting: Using Damgard-Jurik's threshold based decryption algorithm
Token signed by Private key of CA
CA validates ring signature
Anonymous voter authentication
Biometric sample & token
Vote Counting
Validate the voter
o Candidate 1
o Candidate 2
o Candidate 3
Vote
Voter GUI
Encrypt the vote using Damgard-Jurik's encryption algorithm and create a voting transaction
Voting Blockchain
Miners for Voting blockchain
Vote Casting

Figure 1: **Overall system design of Dike.**

(1) Majority of the EDC members are honest such that they do not collude among themselves (*i.e.* share their individual secrets) and engage in unethical behavior (*e.g.,* preemptive vote count).
(2) Majority of Voters (acting as miners) are honest. "Honest" means that they follow the mining protocol and do not "misbehave" (*i.e.*, do not collude among themselves, drop transactions *etc.*)
(3) The voters are considered responsible only during the election process for casting their vote and willingly mining the transactions. Whence the election concludes, they shall not be accountable. We assume EDC to be responsible and available for carrying out the vote counting after the elections are over.
(4) There must be enough voters with valid tokens, willing to volunteer their service as miners during the election process. This would be necessary for the functioning of the mining protocol by appending the verified voting transactions on blockchain.
(5) The *archival nodes* [59], under government's control, ensures that voting data can be retrieved in its intermediate state, in case all the miners go down.
(6) The identity of miners must be hidden. This would make their coercion, and collusion among themselves, difficult. To achieve this, miners' traffic must be encrypted and encapsulated via VPNs. For this, VPNs' gateways must be hosted outside the country's jurisdiction, such that the anonymity of miners is not compromised. The hosting of VPN outside the jurisdiction of country will render the coercion of VPN provider through legal instruments, infeasible.
(7) The voters are honest to choose one candidate in one vote, *i.e.*, every transaction must correspond to exactly one or no vote [2].

Situational:

(1) Voters are technically savvy for casting a vote digitally.
(2) Voters, CA and EDC have a stable internet connectivity to carry out the election process online.

[2]The current implementation of Dike does not take into account cases such threats, but we have outlined possible ways of dealing with such threats through ZKPs in 6.2

4.2 System Design of Dike

. The major phases involved in Dike are as follows :

User Registration: This step requires the users to register themselves using valid government issued identification document(s). This is an offline step managed by statutory body and overseen by the courts to maintain its legitimacy. It also ensures that at no step, it should be easy to stealthily record the user's identification details and use them later. The detailed steps involved are as follows:

(1) The user v visits a fully secured government office (CA's office) and presents their ID card to get themsself verified.
(2) On verification, the user is assigned a signed one-time token and directed to a sentineled private room within the premises.
(3) Here their face, fingerprint and iris sample, are captured and transformed to get a multimodal cancelable template using a non-invertible function [60–62]. This is driven by the user's chosen password, that serves two purposes – (1) it can be only generated with the chosen secret, known *only* to the voter and (2) adds entropy to the template. The biometric samples in the transformed state B_v^T (also known as *cancelable biometric*) is saved to the database DB_B for future use. The cancelable biometric is used as a part of the anonymous user authentication as its inversion to the original state is infeasible, thus hiding the voter's identity. The template matching is henceforth performed in this transformed state [16].
(4) Subsequently, the user generates an RSA public-private key pair (K_v, S_v) using a key generation algorithm as described in appendix B. This key pair *must not* bear any personally identifiable information. The user uploads the corresponding public key K_v ($O(2048bits)$ to a publicly available database DB_K, that also contains other voters' public keys. The public key K_v, along with the cancelable template B_v^T, is used to anonymously authenticate the voter (as discussed ahead). It must be noted that Dike does not maintain any association between biometric template and the public key of a user. Moreover, to ensure no one in the premise, *e.g.* security personnel who may be colluding with the databases' administrator(s), acting maliciously, can aid the linking of the cancelable biometric to the public key. Dike is designed to insert records in DB_K in ascending order. This unlinks the key's association with the corresponding cancelable template. This is illustrated in Figure 2a.

Anonymous voter authentication : Before the commencement of the electoral process, the system needs to validate if the user is a legitimate voter. This involves a two-factor authentication process.

The voter v captures and simultaneously transforms his live biometric samples (face, fingerprint and iris) B_v (using their device), into $B_v^T{}'$. Thereafter, the voter generates a ring signature σ_v (as described in 2.1.4 and detailed in B). The signers/members in the scheme are voters with their public keys generated during *registration phase*, and verifiers are CAs.

The signature is created and sent to the CA as follows:

(1) The voter v selects a random subset of $m-1$ public keys (except his/her own) from the pool of n public keys (K_vs) (of other voters in their constituency) saved in DB_B.
(2) The voter creates a signature σ_v comprising of the chosen m public keys (including their key), a random number g, known as glue, and m random numbers (corresponding to each selected public key). The ring signature algorithm is discussed in detail in Appendix B. This ring signature σ_v is generated over the cancelable biometric template $B_v^T{}'$. Thereafter, the template $B_v^T{}'$ and the ring signature σ_v, are sent to the CA.

The CA validates signature σ_v, and then the biometric template $B_v^T{}'$, by testing whether the glue g equals a function that involves the symmetric cipher key (*i.e.* cancelable template), the glue, the public keys, and the random numbers, which the voter supplies.

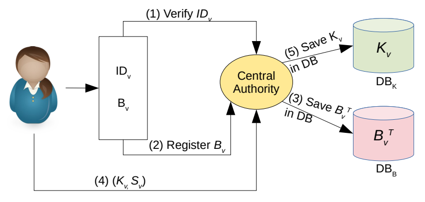
Upon successful validation, CA checks if $B_v^T{}'$ is present in the database DB_B. If it is, then the CA generates a signed token (t_v), digitally signs with its private key pr (as described in section 2.1.3, definition 2.4), sends it to the voter and saves it in DB_B along with B_v^T. The corresponding public key pb of CA is used by miners to validate the token during *Vote Casting Phase.* This token t_v also bears details of its validity period as depicted in Figure 2b.

System Initialization: The CA prepares the system for the election day by:

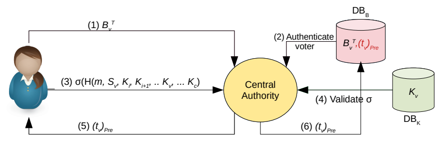
(1) Populating the smart contract with candidate information (C_1, C_2, ..., C_n) and freezing it. The blockchain ensures that once deployed, a smart contract is immutable.
(2) Identifying/freezing the minimum number of EDC members, *i.e.* threshold th ($th > sh/2{+}1$, where sh is total number of members in EDC) required to pool in their partially decrypted vote count, at the end of voting cycle. This is required for obtaining the final vote count by combining the partial counts generated by the EDC members. CA also generates the public key pk (common across all EDC members) for Damgard-Jurik-based encryption of casted votes (as described in Appendix A, under A.2 (1))
(3) To facilitate (2), for each EDC member $e \; \epsilon \; (1, sh)$, a private keyshare sk is generated. This is required for the Damgard-Jurik's threshold-based decryption of the cumulated votes, after the voting concludes. Note that to decrypt cumulative vote count, atleast th EDCs must agree to pool in their secret (as described in Appendix A, under A.2 (1)) and demonstrated in Figure 2c.

Vote Cast : To allow the voter to anonymously cast their vote, they are presented with a frontend of a DApp that interacts with the smart contract at the backend, so as to record the vote as a transaction. To successfully cast the vote following steps are taken:

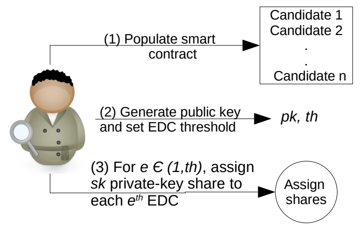
(1) The voter v captures and submits their multi-modal cancelable biometric template. This is verified against the one saved in database DB_B, as described above.
(2) Voter v presents his signed token t_v which is verifiable using CA's public key pb, generated during the *authentication phase.*
(3) Voter is then directed to cast their vote, where they choose a candidate C.
(4) The plain text C, corresponding to the selected candidate, is converted to its string format. *E.g.*, in an election having three candidates, with a voting population of size 15 individuals, four bits per candidate are required to record the choices. If a voter wants to cast a vote for candidate 1, then the corresponding bit string should look like 000000000001. Then the string is encrypted using EDC's public key pk (generated in step (3)) using Damgard-Jurik's encryption algorithm (as detailed in Appendix A (2)), on the user's device. To ensure that a voter casts his vote for only one candidate, a non-interactive zero-knowledge proof may be employed. One such scheme is described ahead in 6.2.



(a) User Registration

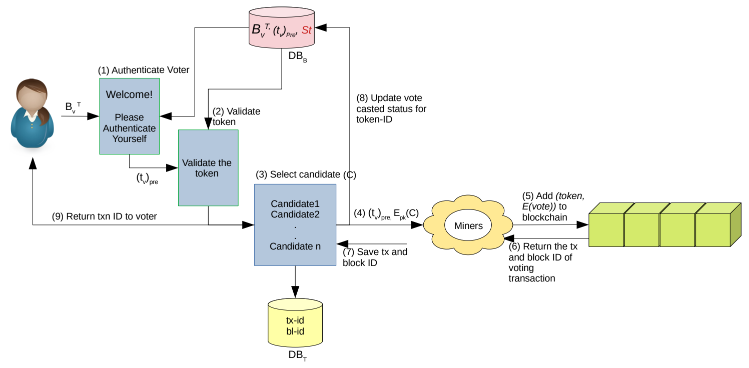


(b) Anonymous Voter Authentication



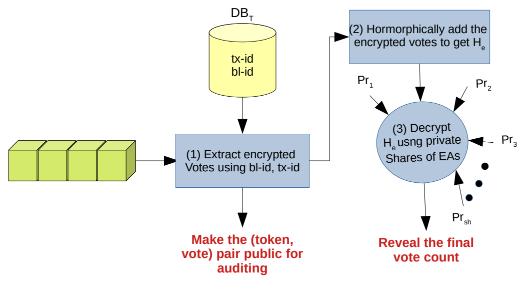
(1) Populate smart contract
Candidate 1
Candidate 2
.
.
Candidate n
(2) Generate public key and set EDC threshold
pk, th
(3) For e ∈ (1,th), assign sk private-key share to each eth EDC
Assign shares

(c) System initialization by central authority



B$_v^T$, (t$_v$)$_{Pre}$, St
DB$_B$
(1) Authenticate Voter
B$_v^T$
Welcome!
Please Authenticate Yourself
(2) Validate token
(t$_v$)$_{pre}$
Validate the token
(8) Update vote casted status for token-ID
(3) Select candidate (C)
Candidate1
Candidate2
.
Candidate n
(9) Return txn ID to voter
(4) (t$_v$)$_{pre}$, E$_{pk}$(C)
(7) Save tx and block ID
Miners
(5) Add (token, E(vote)) to blockchain
(6) Return the tx and block ID of voting transaction
tx-id
bl-id
DB$_T$

(d) Vote Casting Phase



DB$_T$
tx-id
bl-id
(1) Extract encrypted Votes using bl-id, tx-id
Make the (token, vote) pair public for auditing
(2) Hormorphically add the encrypted votes to get H$_e$
Pr$_1$
Pr$_2$
Pr$_3$
Pr$_{sh}$
(3) Decrypt H$_e$ usng private Shares of EAs
Reveal the final vote count

(e) Vote Counting Phase

Figure 2: Individual phases of Dike's operation.

(5) This token and encrypted vote pair $(t_v, E(vote))$, generated as a transaction by the smart contract, is then broadcast over the network. The encrypted vote would prevent an eavesdropper from peeking into the network and getting a partial vote count, during the voting cycle. The voting transaction is selected by miners. They verify (*i.e.*, check whether it uses legal encryption) and validate (*i.e.* check if it conforms to the smart contract and is well-formed) these votes, before permanently recording them on the voting blockchain. Note, that the votes that are malformed (*i.e.*, bear no or more than one vote), are flagged invalid, and still recorded in the blockchain. But they are never counted. This ensures transparency to the observers by ensuring that the votes casted are equal to the votes recorded in the blockchain. This provides vote accountability to voter, by aiding them to check if the vote is rightly recorded in the system.

(6) The transaction ID tx_id and block ID bl_id for the recorded transaction is sent back to the DApp (*i.e.*, to the voter) as proof.

(7) The vote casting status for the respective token ID t_v is updated as $True$ in the database DB_B to check if voters later make hoax claims (defined in 3.2), regarding their vote accountability. The step is also illustrated in Figure 2d.

Vote Counting and Accountability: The encrypted votes are read by traversing the blockchain. These votes are multiplied together[3] using the additively homomorphic property as follows:

(1) The encrypted transactions are read from the blockchain and every pair $(t_{v_i}, E(vote)_i)$ is made public for voters to audit their casted vote. The presence of a pair in the blockchain, confirms that the vote has been casted properly.

(2) The unflagged (*i.e* valid) encrypted votes are multiplied together to get the encrypted cumulative vote count H_e where $H_e = E\Sigma(vote)_i$. This is detailed in Appendix A, under A.2 (3).

(3) *Only* the accumulated votes, H_e, and *not* the individual ones, are decrypted using private key shares of the EDC members. *I.e.* every EDC member performs their part of the decryption. This is described in Appendix A, under A.2 (4).
This ensures that the vote counting cannot be performed by a single entity, thereby delegating it to the EDC. Hence, decentralizing the process. The final decryption involves collating partially decrypted output of atleast th EDC members and then applying Damgard-Jurik's threshold-based decryption, as described in A, under A.2 (5). This step is depicted in Figure 2e.

4.3 Design Decisions

We now list out the important design considerations and the motivations behind choosing them.

(1) Dike employs a private Ethereum-based blockchain, that uses proof-of-work (PoW), due to the following reasons:
- PoW-based consensus protocol provides an opportunity for the voters to participate in mining transactions. Their involvement shall help win citizens' trust, while also preventing any foreign intervention. Though, proof-of-stake (PoS) based consensus is energy conserving, it provides more authority to miners who have high funds to stake. This could be abused by political parties who can eventually resort into dropping or deliberately delaying the inclusion of transactions in the block. Eventually, complaints from citizens and administrators could provide valid grounds to roll back to

[3]In the encrypted form

offline voting system. Any other centralized service would be easy to compromise and also violate the principles of Dike.
- Private network helps deal with the design and implementational complexity. It is easy to keep track of voting transactions and the length of the blockchain.
- Finally, a private network helps evaluate the system's performance in a controlled environment.

(2) To prevent the counting of votes, before elections conclude, Dike relies on threshold-encryption (with distributed key shares). Decryption requires a consensus among a majority of the shareholders. This prevents a powerful actor from maliciously decrypting votes at will. Our proof of concept uses Damgård-Jurik's threshold-encryption, but others like ElGamal [63] and Cramer-Shoup cryptosystems [64] could also be used.
(3) Additionally, the Damgård-Jurik threshold encryption, being additively homomorphic, does not mandate the decryption of the individual votes for counting them. Multiplying the encrypted vote casts is equivalent to the encryption of the cummulative count of the votes cast, which when decrypted, provides the final vote tally.
(4) We use 2048-bit RSA public-private keys (for the threshold encryption). Coupled with the fact that the design considers about a million voters (approximate size of large constituencies in South Asia [20]), Dike can support nearly 90 candidates.

5 EXPERIMENTAL SETUP AND EVALUATION

Dike is implemented using an Ethereum (PoW) based private blockchain. The private blockchain has been purposely chosen to gauge the performance of Dike in a fully-controlled environment consisting of machines acting as miners, web frontend, and bootnodes. We tested it under load conditions corresponding to peak voting hours, measuring the time taken to cast and count votes.

5.1 Empirical Evaluations

We begin by describing our system components and how they were connected together. Thereafter, we describe how we emulated the entire voting process and measured the metrics described above.

5.1.1 Implementational and setup details. Our private blockchain-based setup consisted of the following components:

(1) *Bootnodes:* Bootnodes, are publicly accessible nodes that help peers discover each other.
(2) *Frontend app/node:* Frontend is the host running e-voting service, providing web interface to voters, developed using Javascript.
(3) *Miners:* Miners mine transactions that are added to the blockchain. Some of these are directly connected with the frontend nodes and are responsible for (i) creating the transactions governed by the rules stated in the smart contract, (ii) broadcasting the transactions to the blockchain network.

Our testbed consisted of 44 VMs acting as miners (nodes), each provisioned with 2 vCPUs and 8GB RAM. These nodes were configured using geth version 1.11.5. As each node joins the network, the discovery ensues through the bootnode. The miners communicate with each other through an in-lab OpenVPN gateway, thereby never exposing their IP address to others (*i.e.* peers or eavesdroppers).

To test Dike, we created a smart contract that supported 90 candidates. The smart contract is implemented using Solidity pragma $\geq$ 0.7.0 and $<$ 0.9.0. We cast 1m votes through 120 simultaneous sessions of the frontend. This is the approximate number of voters in one of the largest constituencies of South Asia [20]. This helped simulate the peak load the system may experience during real voting hours.

The voting transaction comprising of the token and voter's encrypted choice (following steps as discussed in 4.2) is sent from the web app to the connected miners. These node(s) create the transaction as per the rules stated in the smart contract. The transaction is then broadcast to the entire network (of miners). We measure various system parameters and metrics that help evaluate the feasibility of Dike. We describe them next.

- ***Votes per half hour (VPH)*:** This is a measure of the throughput of the system, in terms of votes cast every half hour. This is a prime metric that determines the rate at which the system can handle requests, *i.e.*, it decides the time it takes to cast all votes.
- ***Block Processing Time (BPT)*:** This is the time required to create a new block in the voting blockchain.
- ***Difficulty*:** Difficulty, a standard metric, measures of the complexity of the mining problem (*i.e.*, mining voting transactions) that needs to be solved to generate a cryptographic hash of a new block. It decides the rate of block generation.
- ***Hashrate*:** It is a measure of computational power spent by miners to generate (mine) a new block. It is an important metric in gauging the health (security) of any blockchain network, *i.e.* higher the difficulty the harder it is to brute force computations.
- ***Gas Consumed*:** It is a measure of the computational power required to execute specific operations in a blockchain (*e.g.* deploying a smart contract) and measured in generic units.

We cast about 1M fake votes and recorded the performance of Dike at each step. This includes the gas consumed during the initialization process, measurements *w.r.t* ring signatures used for voter authentication, and performance metrics during the vote-casting and vote-counting phase. We also measure the change in RAM consumption, difficulty, and hashrate while casting the votes.

5.2 Results

Initialization Phase: The gas consumed for deploying smart contracts in the network was 353008 units. A transaction is created from token ID and encrypted vote, as per the rules governed by the smart contract.

Voter authentication: The public-private key pair which is generated by voter in *registration phase* is 2048 bits long. Each voter creates a ring signature governed by the ring size (as determined by CAs during *initialization phase*) on his device. The signature generation, tested on a Ryzen 9 5950x (16C/32T), provisioned with 96GB RAM, was relatively fast. *E.g.*, it took about 38 seconds to generate a signature, with a ring size of 90,000 voters. The corresponding signature verification, tested on the same machine, took about 120 sec. These numbers are presented in Fig. 3.

Vote Casting: We tested our system by casting 1,000,000 votes parallely from 120 sessions, that took around 10 hours to complete. The time taken to cast one vote on the network averages to 4.3sec, with vote encryption using Damgard-Jurik encryption

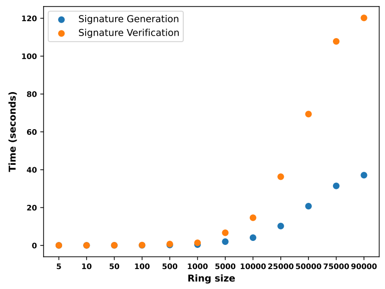


Figure 3: Time taken by a voter to create a ring signature and by CA to verify it.

algorithm [65, 66] taking ≈ 8.7ms (on the same machine used for generating and verifying the signatures). The blockchain height after casting 1M votes was 11000. The change in the number of blocks and transactions generated every half hour *i.e.* BPT and VPH over time is depicted in 4. The number of blocks was initially high, because of low initial difficulty. Later, with an increase in difficulty, the block generation rate also dropped. The transactions cast in the blockchain stabilize over time and drop towards the end of the voting cycle. On average 91 transactions are accommodated in each block, with each encrypted transaction (raw transaction size in geth) being ≈ 1519 bytes.

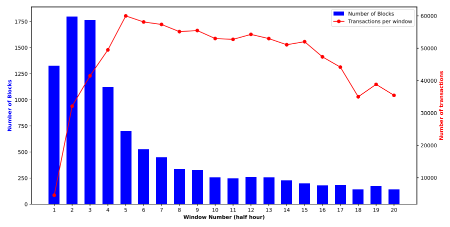


Figure 4: Number of blocks and transactions generated every half hour.

We start with the low difficulty of 316, 000 which rose to approx 49M by the end of the voting cycle, with the maximum hashrate surging to 152 Megahash per sec.(ref. Fig 5). This ensures that the blocks are quickly generated resulting in low end-to-end voting latency. The difficulty is further automatically adjusted as in other PoW based blockchain networks. These trends are comparable to other public blockchains like *Monero* [67, 68], *Ethereum Classic* [69, 70], *etc.*.

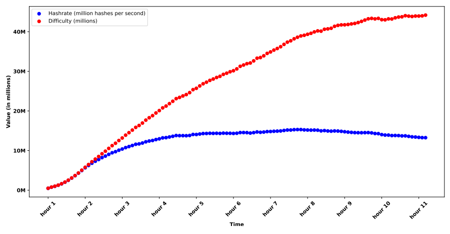


Figure 5: Hashrate and difficulty observed when 10M votes were casted.

Vote Counting: It took about 6.2*secs* to homomorphically add the transactions and decrypt the cumulated count. This is much faster compared to existing offline counting.

6 DISCUSSIONS

Here we provide detailed security analysis of Dike highlighting how it evades various threats enlisted in 3.2. Additionally, we discuss some open challenges we plan to incorporate in future iterations.

6.1 Security Analysis

We describe how Dike resists attacks *w.r.t.* the threat model.

Potential attacks by voters

Ballot stuffing attack: It is a kind of electoral fraud in which number of votes casted is greater than the total number of legitimate voters [71]. To prevent this attack, the system maintains a voter database containing transformed biometric template that is tightly coupled with the signed token, assigned to the voter. This helps validate a permissible set of tokens that can be used to cast a vote. This safety check prevents the voter from casting multiple votes using the same credentials (cancelable biometric and signed token).

Spoofing attack: An attacker could impersonate voters and attempt to cast votes on their behalf [72]. To ensure that no user is able to spoof others' biometric, we employ a multi-biometric system (involving samples of face, iris and fingerprint). For an adversary to successfully impersonate a voter, they would require all these biometric samples, *along* with voter's chosen password (which is used for generating the cancelable template), as described in 4.2. This makes it difficult to carry out such an attack. [4]

Non-repudiation attack: This attack involves an attacker attempting to cast votes on others' behalf, *claiming to have obtained the latters consent.* Dike is designed to disapprove such claims by tightly coupling the signed token with the transformed biometric template in the authentication phase. To cast the vote, a user must present the same cancelable biometric template as used during authentication phase, along with the token.

Hoax attack: The voter could allege that their vote was not recorded in the blockchain, leading to the following possibilities.

(1) The voter presents a bogus token to justify their claim. However, the tokens in database DB_B would disprove their claim.
(2) The voter, even though acquires a token, never actually casts his/her vote. Here, the vote casting status in database DB_B, corresponding to the token would reveal the truth.

Potential threats from CAs and/or EDC

Preemptive vote count attack: The EDC members may attempt to obtain the partial vote count before the voting process concludes. While individual members may not succeed (due to threshold encryption), a significant number (exceeding the threshold) may collude (*e.g.* under coercion by repressive regimes) and decrypt the votes. Parties could use this to gain a glimpse of the possible voting outcome, and thus resort to tactics to influence voters [11].

We propose reliance on two strategies to make such efforts hard. First, the committee must consist of entities from diverse organizations (*e.g.*, judiciary, executive, legislative, press, and even foreign observers), selected by the head of the judiciary[5]. Secondly, the decryption keys could be generated in a distributed manner, such that no single authority would have access to all of it. An outline of such a scheme is presented in appendix D.

[4]Voters could still be coerced to cast votes to candidates. While such threats are realistic, they might not be rampant enough to change electoral patterns across a constituency.
[5]Judiciary is the apex body of democracy whose compromise marks the latter's end.

Deletion attack: People in power could drop the bulk of votes (transactions) from the system by coercing the miners. This would require the former to be aware of the (potential) miners in the system. Dike prevents such attacks as it requires only the token of the voters, to volunteer service as miners. This is not linked with their true identities. Also, Dike requires miners to join the mining network through a VPN service that hides their public IP addresses.

Further, attackers may attempt to delete the names of voters from the voter database DB_B, thereby preventing them from exercising their right to vote. However, since DB_B does not retain the voters' actual identities, parties would refrain from arbitrarily deleting names as the "collateral damages" could be significant. Even if this is attempted, voters could prove this by presenting their tokens that were signed using the CA's private key, making CAs accountable.

Potential threats from miners

Selfish Mining Attack: The miners may act malicioulsy and temporarily withhold blocks. This can impact the revenue generated in case of cryptocurrency blockchains. But there will be no impact on the security of *Dike*.

51% attack: The miners collude and resort into activities like dropping some to bulk transactions. This is mitigated by Dike, as it recruits only verified miners whose network identity is hidden behind VPNs. This will make collusion among miners difficult.

Potential threats from external adversaries

Eavesdropping attack: An external observer/entity may snoop on on Dike's traffic in an attempt to de-anonymize it and gain voter-related confidential data. There are two possible locations where an adversary may eavesdrop. Firstly, between voter's device and Dike's frontend server. Secondly, on intervening network links connecting the miners' (transaction) traffic.

In Dike, the traffic to the frontend web interface is TLS encrypted, thus preserving confidentiality. Simultaneously, the communication between the miners is encrypted and encapsulated via VPNs. Commercial VPN services could be employed, especially those with gateways and headquarters outside the country (in which Dike is used). By and large most commercial VPN services, with users in South Asia, have resisted government coercion [73]. Moreover, even if they complied, it may not be trivial for them to distinguish between the packets of regular cryptocurrency blockchains and those of Dike [74]. Furthermore, it is not yet established whether an adversary can distinguish between different types of traffic generated by Dike and hence launch targetted traffic attacks. Additionally, an adversary could launch timing correlation attack on voting data. This is less of a concern Dike allows one machine-one IP to be used by more than one voter. Thus, correlating the vote with voter and hence revealing its demographic information is not a concern. Moreover, the candidate choice is also protected using Damgard-Jurik's encryption algorithm. Currently, we rely on OpenVPN for this, and not on commercial VPN service. Future revisions shall use the latter. Other alternative to VPN could be mixnets. But to the best of our knowledge, there aren't many known functional mixnets which could also scale to large magnitude.

Fake transactions attack: An attacker could flood the system with fake transactions, thereby forcing miners to unnecessarily expend computation powers. These are prevented by validating the (biometric, token) pair to ensure the legitimacy of voters. Additionally, proving that a vote is well-formed, requires a ZKP-based solution as outlined in the 6.2.

Denial-of-Service Attack: By observing the network traffic, an adversary could learn the IP address of Dike's frontend hosting server. Thereby (s)he can attempt to launch various attacks (*e.g.* DoS). Dike prevents those by proposing to host the frontend as Tor [75] hidden services or behind reverse proxies[76]. Tor hidden services by and large seem to handle a relatively large fraction of the Tor traffic daily [77]. Thus we believe front-end web interface hosted as a hidden service could be used to support this.

Secondly, an adversary may pretend to be an honest miner and learn the IP addresses of many peers. Packet flooding attacks could be launched on these peers. The honest miners would spend precious computing resources processing bogus transactions. Dike mitigates such threats by encrypting and encapsulating miners' traffic through VPNs, that prevents the adversary from learning their miners' IP addresses. Further, commercial VPNs often host their gateways on cloud/distributed hosting architectures and rely on the latters' DDoS protection mechanism [78]. Such protections are generally absent on private VPNs.

Coercing web frontends: The frontend web service needs to be hosted on hosting infrastructures. Some of the miners/citizen volunteers may choose to host them (after paying the necessary hosting fee). These nodes, since technically under the control of the said miners/volunteers could be coerced to deliberately drop votes. This way, once citizens learn their votes were not casted, could protest against the government's decision to switch to online voting. The latter would "gladly" oblige by restoring the older offline voting scheme. The following is a workaround:

(1) The volunteers register themselves with the chief of the judiciary. The latter's job would mostly involve overseeing the process and spotting fraudulent behavior.
(2) The volunteers subscribe to the hosting service(s). Then they upload the front-end program on the servers. Hitherto, the volunteers would communicate with the front end through management ports (*e.g.* SSH) Therefore, to revoke the volunteers' control after uploading, the web service scripts could include commands to shut down all such management ports. The chief of the judiciary, knowing the identity of the volunteers and the IP address of their respective front-end sites, would periodically check the latter to see if such access is surreptitiously enabled.
(3) If the chief of judiciary suspects malicious access, then they could hold the corresponding volunteer accountable.

Attack on voting database: Adversary may attempt to compromise the database DB_B, in an attempt to learn about voter's identities and delete their names altogether. The database maintains transformed biometric templates. It does not reveal the voters' identities. The token ID t_v alone is also not useful without knowing the corresponding identity of its owner. Thus, blindly deleting entries from the database, may prevent a lot of people (including loyalists) from casting their votes. Further, as the database is publicly visible, it would be trivial for the victim to spot such misbehavior.

Bribing the miners: An influential adversary may attempt to bribe the miners by offering them higher incentives. To carry out this attack, they need to know the identity or their IP address. Both these are well concealed in Dike. Firstly, Dike never records the true identity of the registering entities in its database. It stores transformed biometric templates along with their pseudonymous tokens, which can never be mapped to their real identity. Secondly, the real IP addresses of miners are concealed by the use of VPN tunneling for any communication amongst them. Thirdly, for an adversary to potentially attain from this act requires them to have a majority of miners by their side. This requires them to know the miners in the network. The process of joining and leaving the network is very dynamic and is not trivial to guess. Also, it would require a hefty amount to be spent by an adversary in carry this attack, yet with no certainty of it working to their advantage.

Bribing the voters: The influential adversary may attempt to bribe the voter by offering them higher incentive to vote in their favor. Consider the largest legislative assembly [79] with 400 legislative constituencies. For a party to win in a region/province, it would require at least 51% votes in the majority of the constituencies (*i.e.*, 204 constituencies). Based on statistics [79], on average every constituency has about 400k voters, and the average voter turnout is $\approx$ 60%. This equals 240k voters per constituency. The winner needs >50%, *i.e.*, 120k votes. Considering the bribe amount to be \$400 (*i.e.*, average per capita monthly income [80]), the price for winning is \$8 billion $(120,000 * 400 * 204)$. We believe that such large sums of money, that too for a single region, would deter parties from attempting something similar. Even then, it provides no certainty the voters would keep up with their promises.

6.2 Open Challenges

Despite our best efforts, we believe that certain challenges still linger. We now discuss those challenges and also propose potential solutions we plan to implement in the subsequent phase of Dike_v2.

(1) At present, a CA generates the keyshare for all EDC members to decrypt the final vote. This central authority can act maliciously and have a partial vote count. To mitigate this threat, we plan to incorporate a Distributed Key Generation scheme as outlined in appendix D, to aid the EDC members in deriving a key without a central authority. The idea is largely derived from seminal work by Boneh et al. [81, 82], that was later adapted in [83].

(2) It is essential to incentivize the miners to use their services. They must create a wallet on any public cryptocurrency exchange (*e.g.* Coinbase, Binance, *etc.*). Secondly, they also create a local wallet on their mining node. The public addresses (*i.e.* keys) of both these wallets along with the token shall be recorded (as transactions) on the voting blockchain itself.
To incentivize the miners, the CA does the following: (i) Verifies the computation power spent by the miners (by accessing currency earned while mining, in their local wallets), (ii) applies conversion charges on earned currency (as decided by the CAs) (iii) pays the corresponding incentive to the public wallet address shared by the miners. This ensures that the miner's real identity is never compromised during and after the process.

(3) For proving that the votes are correctly formed, one requires presenting evidences. However, this should neither require compromising the vote's secrecy, nor voter's own anonymity. This could be achieved by presenting non-interactive zero knowledge proofs, validating that the votes cast are "well formed", *i.e.* vote that bears at most one vote, to exactly one candidate.
Consider an election with ℓ candidates. Thus a vote is a binary ℓ-tuple where the i^{th} bit represents the if the voter has chosen the i^{th} candidate or not (1:yes and 0:no). Thus a well-formed vote is an ℓ-tuple with hamming weight of *at most* one. This can be achieved by the voter sending $c_1, \ldots, c_\ell$ ciphertexts where c_i denotes either an encryption of zero or one. These are accompanied by proofs $\omega_1, \ldots, \omega_\ell$, where ω_i a non-interactive proof that ensures that the corresponding ciphertext c_i is either an encryption of zero or one (see Protocol 1-out-of-2 $n^{s\text{th}}$ power of Appendix C for details). This proves that the ciphertexts $c_1, \ldots, c_\ell$ contain an encryption of an ℓ-bit string.
But one still needs to prove that the hamming weight is at most one. This is achieved by sending one additional non-interactive proof $\omega_\sum$ that proves that $\prod_{i=1}^{\ell} c_i$ is also an encryption of zero or one. Once all these proofs are verified, the miner requires combining the ciphertexts $c_1, \ldots, c_\ell$ to create the final vote,

$$c = \prod_{i=1}^{\ell} c_i^{(i-1)2^b},$$

where it is assumed that c_1 corresponds to the least significant b-bits of the final encrypted vote, with b being the smallest positive integer such that 2^b is greater than the total voter population of the constituency. This idea was originally presented in [65, Section 6].

7 CONCLUSION

Elections are critical for democracies, but they can be maneuvered, even in the largest ones [55]. The diverse voter demographics of such countries are exploited to win elections. So far, this has mostly been restricted to campaigns and on-ground surveys. However, in recent years, reports of electoral data frauds, such as leaking the data of who is the winner of an election and who all *exactly* voted to powerful parties to win elections, have surfaced [11–13].

We present a scalable, blockchain-based e-voting system, Dike, which aims to prevent such frauds. Apart from the usual requirements of e-voting systems, Dike strives to fulfill three additional ones, *viz.* (1) electoral demographics privacy, (2) anonymous vote accountability, and (3) preventing preemptive vote counting.

Dike preserves demographic privacy by neither seeking nor recording the voter's real identity during the entire voting cycle. For this, Dike employs ring-signatures coupled with multi-biometric cancelable template, to anonymously authenticate the voters. The voter is then given a signed token, that is used by him to cast votes. Also, the votes are encrypted with Damgård Jurik's threshold-based decryption, which prevents preemptive vote count. Dike was tested for performance and scalability in a controlled environment by casting 1M votes which took around 10 hours with 44 miners. The vote-counting time was greatly reduced when compared to offline processes (< 7 sec. to obtain the final count for 1M votes).

REFERENCES

[1] B. Adida, "Helios: Web-based open-audit voting." in USENIX security symposium, vol. 17, 2008, pp. 335–348.
[2] N. Huber, R. Küsters, T. Krips, J. Liedtke, J. Müller, D. Rausch, P. Reisert, and A. Vogt, "Kryvos: Publicly tally-hiding verifiable e-voting," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, pp. 1443–1457.

[3] M. Bougon, H. Chabanne, V. Cortier, A. Debant, E. Dottax, J. Dreier, P. Gaudry, and M. Turuani, "Themis: an on-site voting system with systematic cast-as-intended verification and partial accountability," 2022, pp. 397–410.

[4] D. Chaum, "Blind Signatures for Untraceable Payments," in Advances in Cryptology - CRYPTO, 1982, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Boston, MA: Springer US, 1983, pp. 199–203, DOI: 10.1007/978-1-4757-0602-4_18, ISBN: 978-1-4757-0602-4.

[5] ——, "Secret-ballot receipts: True voter-verifiable elections," IEEE Security Privacy, vol. 2, pp. 38–47, 1983.

[6] D. Chaum, P. Y. Ryan, and S. Schneider, "A practical voter-verifiable election scheme," vol. 10, pp. 118–139, 2005.

[7] "Current offline voting systems are neither safe nor secure," https://right2vote.in/current-offline-voting-systems-are-neither-safe-nor-secure/.

[8] "Votebank politics," https://en.wikipedia.org/wiki/Votebank.

[9] N. Martin and D. Picherit, "Special issue: electoral fraud and manipulation in india and pakistan," Commonwealth & Comparative Politics, vol. 58, no. 1, pp. 1–20, 2020.

[10] L. Michelutti, "Electoral manipulation and impunity: ethnographic notes from uttar pradesh," Commonwealth & Comparative Politics, vol. 58, no. 1, pp. 21–42, 2020.

[11] V. Matthew, "When big data topples the secret ballot: The dangerous evolution of electioneering in india."

[12] K. Saini, "How political parties can use voter data to manipulate your vote," https://www.livemint.com/technology/tech-news/how-political-parties-can-use-voter-data-to-manipulate-your-vote-11581012155214.html.

[13] "It's an important issue: Supreme court issues notice on plea alleging voter profiling by election commission," https://www.livelaw.in/top-stories/supreme-court-voter-profiling-aadhaar-linking-election-commission-telangana-andhra-pradesh-216677.

[14] "What is india's caste system?" https://www.bbc.com/news/world-asia-india-35650616.

[15] "Ring Signatures," https://en.wikipedia.org/wiki/Ring_signature.

[16] Scholarpedia, "Cancelable biometrics," "http://www.scholarpedia.org/article/Cancelable_biometrics", 2010.

[17] R. A. Eltaieb, G. M. El-Banby, W. El-Shafai, F. E. Abd El-Samie, and A. M. Abbas, "Efficient implementation of cancelable face recognition based on elliptic curve cryptography," Optical and Quantum Electronics, vol. 55, no. 9, p. 841, 2023.

[18] I. Damgård and M. Jurik, "A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System," in Public Key Cryptography (PKC), K. Kim, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 119–136, DOI: 10.1007/3-540-44586-2_9, ISBN: 978-3-540-44586-9.

[19] ——, "A Length-Flexible Threshold Cryptosystem with Applications," BRICS report series, vol. record 03/16, 2003, Available at http://www.brics.dk/RS/03/16/.

[20] Wikipedia, "Malkajgiri lok sabha constituency," "https://en.wikipedia.org/wiki/Malkajgiri_Lok_Sabha_constituency#:~:text=As%20of%202019%2C%20Malkajgiri%20is,number%20of%20electors%20with%203%2C150%2C303", 2023.

[21] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook of applied cryptography. CRC press, 2018.

[22] J. Katz and Y. Lindell, Introduction to Modern Cryptography, Second Edition. CRC Press, 2014, ISBN: 9781466570269.

[23] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 552–565, 10.1007/3-540-45682-1_32.

[24] D. Chaum and E. van Heyst, "Group signatures," in Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings, ser. Lecture Notes in Computer Science, D. W. Davies, Ed., vol. 547. Springer, 1991, pp. 257–265, 10.1007/3-540-46416-6_22.

[25] F. S. Wood, "Electric Voting-machine," U. S. Patent 616,174, Granted on 20th December, 1898.

[26] R. H. McKay, W. R. Smith, and H. Deutsch, "Voting System," U. S. Patent 4,025,757, Granted on 24th May, 1977.

[27] R. J. Boram, "Electronic voting Machine and System," U. S. Patent 4,641,240, Granted on 3rd February, 1987.

[28] W. H. Carson, "Electronic Voting Machine," U. S. Patent 4,649,264, Granted on 10th March, 1987.

[29] S. Rieber, "Vote selling and self-interested voting," Public Affairs Quarterly, vol. 15, no. 1, pp. 35–49, 2001. [Online]. Available: http://www.jstor.org/stable/40441274

[30] "Voting residence," https://www.fvap.gov/info/laws/voting-residence.

[31] D. L. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," Communications of the ACM, vol. 24, no. 2, pp. 84–90, February 1981, DOI: 10.1145/358549.358563, ISSN: 0001-0782.

[32] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," bitcoin.org, 2008.

[33] "Voatz," https://voatz.com/.

[34] "Agora," https://www.agora.vote/.

[35] "Democracy live," https://democracylive.com/omniballot-portal/.

[36] V. Mathew, "When Big Data topples the Secret Ballot: The Dangerous Evolution of Electioneering in India," The Leaflet: Constitution First, 2019.

[37] K. Saini, "How Political Parties Can Use Voter Data to Manipulate Your Vote," Mint, 2020.

[38] "Features of blockchain," "https://www.geeksforgeeks.org/features-of-blockchain/".

[39] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21. Springer, 2017, pp. 357–375.

[40] G. G. Dagher, P. B. Marella, M. Milojkovic, and J. Mohler, "Broncovote: Secure voting system using ethereum's blockchain," 2018.

[41] F. S. Hardwick, A. Gioulis, R. N. Akram, and K. Markantonakis, "E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy," in 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE, 2018, pp. 1561–1567.

[42] Z. Wang, X. Luo, M. Li, W. Sun, and K. Xue, "Wevoting: Blockchain-based weighted e-voting with voter anonymity and usability," in GLOBECOM 2022-2022 IEEE Global Communications Conference. IEEE, 2022, pp. 2585–2590.

[43] S. Park, M. Specter, N. Narula, and R. L. Rivest, "Going from bad to worse: from internet voting to blockchain voting," Journal of Cybersecurity, vol. 7, pp. 1–15, 2021.

[44] M. A. Specter, J. Koppel, and D. Weitzner, "The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in {US}. federal elections," in 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 1535–1553.

[45] M. Specter and J. A. Halderman, "Security analysis of the democracy live online voting system," in 30th USENIX Security Symposium (USENIX Security 21), 2021, pp. 3077–3092.

[46] S. Wu and D. Galindo, "Evaluation and improvement of two blockchain based e-voting system: Agora and proof of vote," Edited by David Galindo. University of Birmingham. http://www. dgalindo. es/mscprojects/shuang. pdf, 2018.

[47] "Polys: Online voting solution," https://www.linkedin.com/company/polys-online-voting-systems/?originalSubdomain=il.

[48] "followmyvote.com," "https://followmyvote.com/".

[49] A. Hadid, "Face biometrics under spoofing attacks: Vulnerabilities, countermeasures, open issues, and research directions," in 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 113–118.

[50] Liopa, "Biometric authentication and spoofing," "https://liopa.ai/biometric-authentication-spoofing-liopa-guest-article-from-biometric-technology-today", 2023.

[51] C. Bowers, "Security properties for electronic voting," "https://www.cs.cornell.edu/courses/cs513/2002SP/proj.00.StuSolns/mbt91", 2002.

[52] T. Print, "Election panel wants to revisit rule on punishment to voters for false mismatch claims," "https://theprint.in/india/election-panel-wants-to-revisit-rule-on-punishment-to-voters-for-false-mismatch-claims/246139/", 2019.

[53] "Videos online show blatant ballot-stuffing in russia," https://www.washingtonpost.com/news/worldviews/wp/2018/03/19/videos-online-show-blatant-ballot-stuffing-in-russia/.

[54] "Allegation on deletion of voters' names false, says karnataka cm," "https://www.thehindu.com/news/national/karnataka/allegation-on-deletion-of-voters-names-false-says-karnataka-cm/article66213931.ece".

[55] S. Das, "Democratic backsliding in the world's largest democracy," Available at SSRN 4512936, 2023.

[56] Hacken, "Sybil attack in blockchain: Examples prevention," "https://hacken.io/insights/sybil-attacks/#:~:text=A%20Sybil%20Attack%20is%20a%20form%20of%20online%20security%20violation,1973%20novel%20by%20Flora%20Schreiber", 2023.

[57] C. Feng and J. Niu, "Selfish mining in ethereum," in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019, pp. 1306–1316.

[58] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," ACM Computing Surveys (CSUR), vol. 53, no. 3, pp. 1–43, 2020.

[59] Ethereum, "Ethereum archive node," "https://ethereum.org/en/developers/docs/nodes-and-clients/archive-nodes/#:~:text=An%20archive%20node%20is%20an,run%20than%20a%20full%20node.", 2023.

[60] P. Sharma, G. S. Walia, and R. Rohilla, "Recent advancement in cancelable biometric for user recognition: A brief survey," in 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), 2020, pp. 137–146.

[61] K. Gupta, G. S. Walia, and K. Sharma, "Novel approach for multimodal feature fusion to generate cancelable biometric," The Visual Computer, vol. 37, pp. 1401–1413, 2021.

[62] R. Asthana, G. S. Walia, and A. Gupta, "Random area-perimeter method for generation of unimodal and multimodal cancelable biometric templates," Applied Intelligence, vol. 51, pp. 7281–7297, 2021.

[63] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," Journal of Cryptology, vol. 15, pp. 75–96, 2002.

[64] R. Canetti and S. Goldwasser, "An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack (extended abstract)," in Advances in Cryptology — EUROCRYPT '99, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 90–106.

[65] I. Damgård, M. Jurik, and J. B. Nielsen, "A Generalization of Paillier's Public-key System With Applications to Electronic Voting," International Journal of Information Security, vol. 9, no. 6, 2010, DOI: 10.1007/s10207-010-0119-9, ISSN (electronic): 1615-5270, ISSN (print): 1615-5262.

[66] I. Damgård and G. L. Mikkelsen, "Efficient, Robust and Constant-Round Distributed RSA Key Generation," in Theory of Cryptography, D. Micciancio, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 183–200, DOI: 10.1007/978-3-642-11799-2_12, ISBN: 978-3-642-11799-2.

[67] "Monero difficulty chart," https://www.coinwarz.com/mining/monero/difficulty-chart.

[68] "Monero hashrate chart," https://www.coinwarz.com/mining/monero/hashrate-chart.

[69] "Ethereum classic difficulty chart," https://www.coinwarz.com/mining/ethereum-classic/difficulty-chart/.

[70] "Ethereum classic hashrate chart," https://www.coinwarz.com/mining/ethereum-classic/hashrate-chart.

[71] Ballotpedia, "Ballotpedia," "https://ballotpedia.org/Ballot_stuffing", 2023.

[72] Malwarebytes, "What is spoofing attack?" "https://www.malwarebytes.com/spoofing", 2023.

[73] "2022 in tech policy: India on the brink," "https://entrackr.com/2022/12/2022-in-tech-policy-india-on-the-brink/".

[74] "This vpn built on blockchain could be the next step in privacy tech," "https://www.cnet.com/tech/services-and-software/this-vpn-built-on-blockchain-could-be-the-next-step-in-privacy-tech".

[75] "Onion services," https://tb-manual.torproject.org/onion-services/.

[76] "What is a reverse proxy? | proxy servers explained," https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/.

[77] G. Kadianakis and K. Loesing, "Extrapolating network totals from hidden-service statistics," Tor Tech Report01 (001 (2015), 2015.

[78] "Cloudflare," https://cloudflarewarp.com/.

[79] "2022 uttar pradesh legislative assembly election," https://en.wikipedia.org/wiki/2022_Uttar_Pradesh_Legislative_Assembly_election.

[80] "Salaries in india," https://www.timedoctor.com/blog/average-salary-in-india/#:~:text=If%20you're%20wondering%20what,exchange%20rates%20in%20June%202023.

[81] D. Boneh and M. K. Franklin, "Efficient generation of shared RSA keys (extended abstract)," in Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings, ser. Lecture Notes in Computer Science, B. S. K. Jr., Ed., vol. 1294. Springer, 1997, pp. 425–439, DOI: 10.1007/BFb0052253.

[82] ——, "Efficient generation of shared RSA keys," Journal of ACM, vol. 48, no. 4, pp. 702–722, 2001, DOI: 10.1145/502090.502094.

[83] T. Nishide and K. Sakurai, "Distributed Paillier Cryptosystem without Trusted Dealer," in Information Security Applications - 11th International Workshop WISA, 2010, Jeju Island, Korea, August 24-26, 2010, Revised Selected Papers, ser. Lecture Notes in Computer Science, Y. Chung and M. Yung, Eds., vol. 6513. Springer, 2010, pp. 44–60, DOI: 10.1007/978-3-642-17955-6_4.

[84] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in Advances in Cryptology — EUROCRYPT '99, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238, DOI: 10.1007/3-540-48910-X_16, ISBN: 978-3-540-48910-8.

[85] A. Shamir, "How to Share a Secret," Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979, 10.1145/359168.359176.

[86] I. Damgård and M. Jurik, "A Length-Flexible Threshold Cryptosystem with Applications," in Information Security and Privacy, R. Safavi-Naini and J. Seberry, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 350–364, DOI: 10.1007/3-540-45067-X_30, ISBN: 978-3-540-45067-2.

[87] V. Shoup, "Practical Threshold Signatures," in Advances in Cryptology — EUROCRYPT 2000, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 207–220, DOI: 10.1007/3-540-45539-6_15, ISBN: 978-3-540-45539-4.

[88] Y. Frankel, P. D. MacKenzie, and M. Yung, "Robust efficient distributed rsa-key generation," in Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998, B. A. Coan and Y. Afek, Eds. ACM, 1998, p. 320, DOI: 10.1145/277697.277779.

[89] J. Algesheimer, J. Camenisch, and V. Shoup, "Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products," in Advances in Cryptology — CRYPTO 2002, M. Yung, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 417–432, DOI: 10.1007/3-540-45708-9_27 ISBN: 978-3-540-45708-4.

[90] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, ser. STOC '88. New York, NY, USA: Association for Computing Machinery, 1988, p. 1–10, DOI: 10.1145/62212.62213.

[91] C. Hazay, G. L. Mikkelsen, T. Rabin, T. Toft, and A. A. Nicolosi, "Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting," Journal of Cryptology, vol. 32, no. 2, pp. 265–323, 2019, DOI: 10.1007/s00145-017-9275-7.

A DAMGÅRD AND JURIK'S THRESHOLD ENCRYPTION SCHEME

Damgård and Jurik proposed a length flexible, additively homomorphic threshold encryption scheme [18]. It was a generalisation of earlier Paillier cryptosystem [84]. The cryptosystem is based on the *Decisional Composite Residuosity Assumption (DCRA)* and the *Decisional Diffie-Hellman (composite DDH) Assumption.* These can be formally stated as follows.

Conjecture A.1 (The Decisional Composite Residuosity Assumption (DCRA)). *Let $\mathcal{A}$ be any* PPT *algorithm. Assume $\mathcal{A}$ gets n, x as input, where $n = p \cdot q$ is an admissible RSA modulus of length k bits, and x is either random in $\mathbb{Z}^*_{n^2}$ or it is a random n^{th} power in $\mathbb{Z}^*_{n^2}$. $\mathcal{A}$ outputs a bit b. Let,*

$p_0(\mathcal{A}, k) \overset{\Delta}{=} \Pr[b = 1 | x \xleftarrow{\$} \mathbb{Z}^*_{n^2}]$, and $p_1(\mathcal{A}, k) \overset{\Delta}{=} \Pr[b = 1 | x$ is a n^{th} power in $\mathbb{Z}^*_{n^2}]$.

Then,
$|p_0(\mathcal{A}, k) - p_1(\mathcal{A}, k)| \leq \text{negl}(n)$.

Conjecture A.2 (The Decisional Diffie-Hellman (composite DDH)). *Let $\mathcal{A}$ be any* PPT *algorithm. Assume $\mathcal{A}$ gets $(n, g, g^a \mod n, g^b \mod n, y)$ as input, where $n = p \cdot q$ is an admissible RSA modulus of length k bits, g is an element of Q_n the group of squares (i.e., quadratic residues) in $\mathbb{Z}^*_n$. The values a and b are chosen uniformly random in $\mathbb{Z}_{\varphi(n)/4}$, $\varphi(\cdot)$ denotes the Euler totient function and the value y is either random in Q_n or satisfies $y = g^{ab} \mod n$. $\mathcal{A}$ outputs a bit b. Let,*

$p_0(\mathcal{A}, k) \overset{\Delta}{=} \Pr[b = 1 | y \xleftarrow{\$} Q_n]$, and $p_1(\mathcal{A}, k) \overset{\Delta}{=} \Pr[b = 1 | y = g^{ab} \mod n]$.
Then,

$$|p_0(\mathcal{A}, k) - p_1(\mathcal{A}, k)| \leq \text{negl}(n).$$

We now describe the Damgård and Jurik threshold homomorphic encryption scheme (DJThHE). We assume that there are v players $\mathcal{P} = \{P_1, \dots, P_v\}$ and a access structure $\mathbb{A}_t$ based on the Shamir's secret sharing [85], where $\mathbb{A}_t = \{\mathcal{S} \subseteq \mathcal{P} : |\mathcal{S}| \geq t\}$.

(1) $(\text{pk}, \text{sk}_1, \dots, \text{sk}_v)\text{DJThHE.Setup}(1^\lambda)$**:** The setup algorithm does the following.
- Chooses an RSA modulus $n = p \cdot q$ of length λ bits, with $p = 2p' + 1$ and $q = 2q' + 1$, where p, q, p', q' are primes.
- Selects an element $g \in Q_n$, which is the group of all squares of $\mathbb{Z}^*_n$, i.e., the set of quadratic residues modulo n, and $\alpha \in \mathbb{Z}_\tau$, where $\tau = p' \cdot q' = |Q_n|$.

- Pick the secret value $a_0 = \alpha \in \mathbb{Z}_\tau$, and $a_i \xleftarrow{\$} \mathbb{Z}_\tau$ for $1 \leq i \leq t$, where we take $t \geq \lfloor v/2 \rfloor + 1$ is the threshold of the system with v players. Note that the threshold can be increased or decreased depending on the system preference. This result in the polynomial,

$$f(x) = a_t x^t + a_{t-1}x^{t-1} + \cdots + a_1 x + a_0,$$

where $f(0) = a_0 = \alpha$ is the secret.
- The secret shares are then calculated as

$$\mathrm{sk}_i = f(i).$$

and the corresponding public value is $h = g^\alpha \mod n$, and the verification value of the i^{th} layer is $h_i = g^{\alpha_i} \mod n$.
- The algorithm then outputs the public key $\mathrm{pk} = (n, g, h)$, the verification values $(h_1, \ldots, h_v)$, and the private key of the i^{th} player as $\mathrm{sk}_i = \alpha_i$ for all $i \in [v]$.

(2) $c \leftarrow$ DJThHE.Encrypt(pk, m)**:** Given a message $m \in \mathbb{N} \cup \{0\}$, do the following.
- Choose an integer $s > 0$, such that $m \in \mathbb{Z}_{n^s}$.
- Pick a random $r \xleftarrow{\$} \mathbb{Z}_n$.
- Then the ciphertext c is given by $c = (G, H)$, where

$$G = g^r \mod n \quad \text{and} \quad H = (h^{4\Delta^2 r})^{n^s} \cdot (n+1)^m \mod n^{s+1}.$$

(3) $\widetilde{c} \leftarrow$ DJThHE.Eval(pk, $c_1, \ldots, c_k$)**:** Assume that for all $i \in [k]$, $c_i \in \mathbb{Z}_{n^s}$. Then on input the public keys pk and k ciphertexts $c_1, \ldots, c_k$, the evaluation algorithm outputs a ciphertext

$$\widetilde{c} = \left(\prod_{i=1}^{k} G_i, \prod_{i=1}^{k} H_i \right) = (\widetilde{G}, \widetilde{H}) \text{ (say)},$$

where $c_i = (G_i, H_i) = (g^{r_i} \mod n, (h^{4\Delta^2 r_i})^{n^s} \cdot (n+1)^{m_i} \mod n^{s+1}) \quad \forall\, i \in [k]$,

$$\widetilde{G} = \prod_{i=1}^{k} G_i = g^{\sum_{i=1}^{k} r_i} \mod n$$

and

$$\widetilde{H} = \prod_{i=1}^{k} H_i = \left(h^{4\Delta^2 \left(\sum_{i=1}^{k} r_i \right)} \right)^{n^s} \cdot (n+1)^{\sum_{i=1}^{k} m_i} \mod n^{s+1},$$

and $\Delta \in \mathbb{Q}$ will be defined later.

(4) $(d_i, \sigma_i) \leftarrow$ DJThHE.PartDec(pk, c, sk$_i$)**:** Each player computes it's share d_i as $d_i = G^{2\Delta \mathrm{sk}_i} \mod n$ along with a zero-knowledge proof σ_i that $\log_g(h_i) = \log_{G^{4\Delta}}(d_i^2)$. That is a proof that the player has not deviated from the protocol.

(5) $m \leftarrow$ DJThHE.FinDec(pk, c, B)**:** Given a ciphertext $c = (G, H)$ and $B = \{(d_i, \sigma_i) : i \in S$ for some $S \subseteq \mathcal{P}\}$, the final decryption proceeds as follows.
- Parse c to determine s.
- Check if the Jacobi symbol $\left(\frac{G}{H}\right) = 1$ or not. If not, then return $\perp$.
- For all $i \in [v]$, check the proofs σ_i to determine if the i^{th} player has not deviated from the protocol or not. Let S denote the set of all i's that pass this test.
- If $|S| < t$, then return $\perp$.
- Use Lagrange's interpolation to create the exponent $4\Delta^2\alpha$. Then observing that $h = g^\alpha$, we get

$$d = \prod_{i \in S} d_i^{2\Lambda_i^S} = G^{4\Delta^2 \alpha} = h^{4\Delta^2 r} \mod n$$

, where

$$\Lambda_i^S = \Delta \cdot \prod_{j \in S \setminus \{i\}} \frac{j}{j-i},$$

and Δ is a positive rational to ensure that $\Lambda_i^S \in \mathbb{Z}$.
- Compute $H' = H^2 \cdot d^{-2n^s} = (n+1)^{2m} \mod n^{s+1}$. Note that to ensure that we work in $\mathcal{Q}_n$, it is required to take the square of H.
- Find m by using the polynomial time algorithm given in [18, Section 3] on H'.

B RSA RING SIGNATURE

Consider the RSA setup. Assume that there is a list $\mathcal{L}$ of public keys $\mathrm{pk}_i = (e_i, n_i)$ corresponding to each member of ring containing a total of v members, i.e., the ring $\mathcal{R} = \{(e_1, n_1), \ldots, (e_v, n_v)\}$. Then each public key (e_i, n_i) specifies a one-way trapdoor permutation f_i on $\mathbb{Z}_{n_i}$ as follows

$$f_i(x) = x^{e_i} \mod n_i,$$

where $i = 1, \ldots, v$.

Notice that the trap-door RSA permutations of different ring members have different domain sizes n_i (even thought they may have the same number of bits). This makes it difficult to combine the individual signatures. To solve this issue, the domain of all the trapdoor functions are extended in the following way so that all of them have the same domain $\{0,1\}^b$, where b is such that $n_i < 2^b$ for all $i = 1, \ldots, v$. Then for each RSA trap-door function f_i the domain extended function g_i is defined in the following way. For any $m \in \{0,1\}^m$, g_i is defined as

$$g_i(m) = \begin{cases} q_i n_i + f_i(r_i) & \text{if } (q_i + 1) \cdot n_i \leq 2^b \\ m & \text{else,} \end{cases}$$

where q_i and r_i are non-negative integers such that $m = q_i n_i + r_i$ with $0 \leq r_i < n_i$ for all $i = 1, 2, \ldots, v$. Also assume that Enc is a symmetric encryption function with the corresponding decryption function denoted by Dec. Finally assume that $\mathcal{H}$ is a collision resistant hash function. Note that the proofs given in [23]are in the random oracle model.

Consider a family of keyed *combining functions* $C_{k,v}(y_1, \ldots, y_r)$ which takes a key k, an initialisation value v, and arbitrary values $y_1, \ldots, y_r \in \{0,1\}^b$ defined as follows.

$$C_{k,v}(y_1, y_2, \ldots, y_r) \stackrel{\Delta}{=} \mathrm{Enc_k}(\mathrm{y_r} \oplus \mathrm{Enc_k}(\mathrm{y_{r-1}} \oplus \mathrm{Enc_k}(\mathrm{y_{r-2}} \oplus \mathrm{Enc_k}(\cdots \oplus \mathrm{Enc_k}(\mathrm{y_1} \oplus \mathrm{v}) \cdots)))).$$

Then a signer s with secret key sk$_s$ generates the ring signature corresponding to the message m in the following manner.

- **Chooses a key:** Selects r public keys $pk_{i_1}, \ldots, \mathrm{pk}_{i_r}$ from the list $\mathcal{L}$ including signer s's own public key pk_{i_s} (say). Also sets $k = \mathcal{H}(m)$.

- **Picks a random glue:** Picks an initialisation or "glue" value v uniformly at random from $\{0,1\}^b$, i.e., $b \xleftarrow{\$} \{0,1\}^b$.
- **Picks random x_i's:** Picks $x_i \xleftarrow{\$} \{0,1\}^b$ for all the other ring members $i \in \{i_1, \ldots, i_r\} \setminus \{i_s\}$ and compute $y_i = g_i(x_i)$.
- **Solves for y_s:** Solves the following ring equation for y_s.
$$C_{k,v}(y_1, y_2, \ldots, y_r) = v. \quad (1)$$
Given arbitrary values $y_1, y_2, \ldots, y_{s-1}, y_{s+1}, \ldots, y_r, v$ and a key k, one can efficiently compute y_s satisfying equation (1).
- **Inverts the signer's trap-door permutation:** Uses it's trapdoor, i.e, sk_s, to compute $x_s = g_s^{-1}(y_s)$.
- **Outputs the ring signature:** Outputs the signature on the message m, which is a $(2r+1)$-tuple of the form
$$\sigma = (\mathrm{pk}_{i_1}, \mathrm{pk}_{i_2}, \ldots, \mathrm{pk}_{i_r}; v; x_1, x_2, \ldots, x_r).$$

On receiving the message-signature pair (m, σ), the verifier does the following:

- **Applies the trap-door permutations:** For each $j = 1, 2, \ldots, r$ computes $y_i = g_i(x_i)$.
- **Obtains k:** Hashes the message to compute the encryption key k, i.e., $k = \mathcal{H}(m)$.
- **Verifies the ring equation:** Checks that the y_j's satisfies the equation
$$C_{k,v}(y_1, y_2, \ldots, y_r) = v.$$
If the equation is satisfied, it accepts the signature as valid. Otherwise it rejects the message-signature pair (m, σ).

C SOME ZKPS BASED ON DAMGÅRD AND JURIK'S CRYPTOSYSTEM

We recall some of the ZKPs that appear in [18, 19, 65, 86]. For ease of understanding, we provide the interactive versions of these protocols. Note that all the protocols mentioned in this section are non-interactive which is obtained by a straightforward application of Fiat-Shamir transform on the interactive versions of these protocols given in [18, 19, 65, 86].

Protocol for n^sth powers: An interactive protocol to determine if the given ciphertext c is an $n^{s\text{th}}$ power of some number $\mod n^{s+1}$, *i.e.*, if it is an encryption of zero or not, was presented in**fix!** [65, Section 5.2]. Then applying Fiat-Shamir transform, it is straightforward to convert it into an equivalent non-interactive protocol (see Protocol 1). The correctness of the protocol 1 follows from the following simple observation.
$a' = h^{4\Delta^2 an^s} H^e H^{-e} \mod n^{s+1} = h^{4\Delta^2 an^s} \mod n^{s+1} \Rightarrow e = \mathcal{H}(a')$.
Note that hash function $\mathcal{H}$ used in the protocol 1 is a collision resistant hash function that maps to $\mathbb{Z}_n$. It is important to point out here that it may seem that it should be easy to determine if c is an encryption of zero by checking if $H = h^{4\Delta^2 rn^s}$, since $G = g^r \mod n$ is known. But this is not the case, as recall that $h = g^\alpha \mod n$. Therefore, computing $h^{4\Delta^2 rn^s}$ from G and h would imply solving the computational Diffie-Hellman problem (CDH), which is hard since decisional Diffie-Hellman DDH is hard and DDH $\leq_p$ CDH.

Protocol 1: Non-interactive Protocol for n^sth powers

Input: n, g, s, h, c, Δ
Private input for $\mathcal{P}$: $r \in \mathbb{Z}_n^*$, such that, $c = (G, H)$, where $G = g^r \mod n$ and $H = h^{4\Delta^2 rn^s} \mod n^{s+1}$

Prover $\mathcal{P}(n, g, s, h, c, \Delta, r)$:

- Chooses $a \xleftarrow{\$} \mathbb{Z}_n^*$.
- Compute $H' = h^{4\Delta^2 an^s}$.
- Compute $e = \mathcal{H}(H')$.
- Computes $\widehat{r} = a + e \cdot r \mod n$.
- Sends $(H', e, \widehat{r})$ to the verifier.

Verifier $\mathcal{V}(n, g, s, h, c, \Delta)$:

- Does the following.
  - Checks if H and H' are relatively prime to n.
  - Computes $z = h^{4\Delta^2 \widehat{r} n^s} \mod n^{s+1}$.
  - Computes $H'' = z \cdot H^{-e} \mod n^{s+1}$
  - Checks if $e = \mathcal{H}(H'')$.
- The verifier accepts that H is an $n^{s\text{th}}$ power if and only if it passes all the above checks.

Protocol 1-out-of-2 $n^{s\text{th}}$ power: An interactive protocol to prove that a given ciphertext c is an encryption of either one of two possible values m_1 and m_2, without revealing the actual plaintext was presented in [65, Section 5.2]. Applying Fiat-Shamir transform, it is straightforward to convert this interactive protocol into an equivalent non-interactive one (see Protocol 2). Given a ciphertext $c = (G, H)$ which is an encryption of either m_1 or m_2, both the prover and the verifier computes,
$$u_1 = H \cdot (1+n)^{-m_1}$$
, and
$$u_2 = H \cdot (1+n)^{-m_2}.$$
The prover shows to the verifier that either u_1 or u_2 is an encryption of zero. This proves to the verifier that only one of u_1 and u_2 is an $n^{s\text{th}}$ power. This is presented in the protocol 2, where without loss of generality it is assumed that the prover knows v_1, *s.t.*, $u_1 = H \cdot (1+n)^{-m_1} \mod n^{s+1} = \left(h^{4\Delta^2 v_1}\right)^{n^s} \mod n^{s+1}$. The correctness easily follows from the correctness of the protocol 2 and some easy observations. Note that the non-interactive version of this protocol does not appear in their original papers, but they do provide pointers. It is important to point out here that we require using protocol 2 to prove that a given ciphertext is either an encryption of zero or one, *i.e.*, $m_1 = 0$ and $m_2 = 1$.

Decryption proof: Recall that during threshold decryption A), the parties $P_1, \ldots, P_\nu$ needs to calculate
$$d_i = G^{2\Delta \alpha_i} \mod n.$$
The i^{th} player P_i needs to prove that this was indeed what it submitted. This is done by proving that following identity
$$\alpha_i = \log_g h_i = \log_{G^{4\Delta}} \left(d_i^2\right) \mod \tau,$$
where recall that $\tau = p' \cdot q'$, $p = 2p' + 1, q = 2q' + 1$ and $n = p \cdot q$. The proof of this is identical to that of Shoup RSA threshold signature [87]. In Protocol 3, $\mathcal{H}$ denotes a hash function that outputs t bits and $\mathcal{P}$ denotes the i^{th} player P_i $\forall\, i \in [\nu]$.

Protocol 2: Non-interactive Protocol for 1-out-of-2 n^{s}th power

Input: n, g, s, h, c, Δ

Private input for $\mathcal{P}$: $v_1 = r \in \mathbb{Z}_n^*$, such that, $u_1 = h^{4\Delta^2 v_1 n^s} \mod n^{s+1}$, where without loss of generality we assume that $c = (G, H)$, $G = g^r \mod n$, $H = h^{4\Delta^2 r n^s} \cdot (n+1)^{m_1} \mod n^{s+1}$ and $u_1 = H \cdot (n+1)^{-m_1} \mod n^{s+1} = h^{4\Delta^2 r n^s} \mod n^{s+1}$

Prover $\mathcal{P}(n, g, s, h, c, \Delta, v_1)$:

- Chooses $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_n^*$.
- Compute $H_1 = h^{4\Delta^2 r_1 n^s} \mod n^{s+1}$.
- Compute $H_2 = h^{4\Delta^2 r_2 n^s} \mod n^{s+1}$.
- Compute $e_2 = \mathcal{H}(H_2)$.
- Express

$$\begin{aligned} u_2 &= H \cdot (1+n)^{-m_2} \mod n^{s+1} = h^{4\Delta^2 r n^s} \cdot (1+n)^{m_1 - m_2} \mod n^{s+1} \\ &= \left(h^{4\Delta^2 r} \cdot (1+n)^{(m_1-m_2)/n^s}\right)^{n^s} \mod n^{s+1} = h^{4\Delta^2 v_2 n^s} \mod n^{s+1}, \end{aligned}$$

  where $v_2 = v_1 + ((m_1 - m_2)/(4\Delta^2 n^s)) \cdot \log_h(1+n)$.
- Compute $\widehat{r}_2 = r_2 + e_2 \cdot v_2$.
- Computes $e_1 = \mathcal{H}(H_1, H_2, \widehat{r}_2)$.
- Sends $(H_1, H_2, e_1, \widehat{r}_1, e_2, \widehat{r}_2)$ to $\mathcal{V}$.

Verifier $\mathcal{V}$:

- Does the following.
  - Computes
    * $u_1 = H \cdot (n+1)^{-m_1} \mod n^{s+1}$ and $u_2 = H \cdot (n+1)^{-m_2} \mod n^{s+1}$
    * $H_1' = h^{\widehat{r}_1 n^s} u_1^{-e_1} \mod n^{s+1}$.
    * $H_2' = h^{\widehat{r}_2 n^s} u_2^{-e_2} \mod n^{s+1}$.
  - Checks if
    * $e_1 = \mathcal{H}(H_1', H_2', \widehat{r}_2)$ and
    * $e_2 = \mathcal{H}(H_2')$.
  - Checks if $u_1, u_2, H_1', H_2', \widehat{r}_1$ and $\widehat{r}_2$ are each relatively prime to n.
- The verifier is convinced that c is either an encryption of m_1 or m_2 if and only if it passes all the above checks.

Protocol 3: Non-interactive Protocol for decryption proof.

Input: $n, g, s, \Delta, c = (G, H), h_i = g^{\alpha_i} \mod n, d_i$

Private input for P: Secret α_i of Player P_i

Prover $\mathcal{P}(n, g, s, h, c, \Delta, \alpha_i)$:

- Chooses $r \xleftarrow{\$} \{0, 1, \ldots, 2^{|n|+2t} - 1\}$.
- Computes
  - $\widehat{g} = g^r \mod n$
  - $\widehat{G} = G^{4\Delta r} \mod n$
  - $e = \mathcal{H}(g, G^{4\Delta}, h_i, d_i^2, \widehat{g}, \widehat{G})$
  - $z = r + \alpha_i \cdot e$.
- Sends (e, z) to $\mathcal{V}$.

Verifier $\mathcal{V}$:

- Computes the following.
  - $\widehat{g}' = g^z \cdot h_i^{-e} \mod n$
  - $\widehat{G}' = G^{4\Delta z} \cdot d_i^{-2e} \mod n$
- Accept the proof if and only if the following holds.

$$e = \mathcal{H}(g, G^{4\Delta}, h_i, d_i^2, \widehat{g}, \widehat{G}).$$

D DISTRIBUTED KEY GENERATION

The problem of generating RSA modulus in shared manner was first introduced by Boneh and Franklin in their seminal work [81, 82]. Their protocol was based on the *bi-primality test* which tests if a given integer is a product of two primes or not. It assumed that the player involved in the protocol are *honest but curious adversaries* (*passive adversaries*). At the end, the protocol generates an RSA modulus $n = p \cdot q$ which is publicly know. But none of the parties involved in the distributed algorithm knows the factorization of n, i.e., p and q. Additionally, the protocol also generates a public key for encryption and each player holds a share of their secret key, thereby enabling threshold decryption. Later Frankel et al. [88] made the scheme robust by considering at most $\lfloor (\nu - 1)/2 \rfloor$ malicious players, where ν denotes the total number of players involved in the distributed protocol.

But it is important to note here that for Damgård-Jurik or Paillier cryptosystems the modulus needed is a product of two safe-primes, i.e., of the form $p = 2p' + 1$, where both p and p' are primes. The first efficient solution to this problem was provided by Algesheimer et al. in [89]. Unlike [81] which used bi-primality test this work by Algesheimer et al. showed how to securely implement Miller-Rabin test when the primes are additively shared. Prior to this work by Algesheimer et al. the only known method for solving this problem was to apply the much less efficient general circuit technique of Ben-Or et al. [90]. Algesheimer et al. showed that their protocol is secure against a static honest-but-curious adversaries with up to $\lfloor \frac{k-1}{2} \rfloor$ players, where k denotes the number of players involved in the scheme. This was further improved by Damgård and Mikkelsen [66], where they gave a constant round protocol that is secure against malicious adversaries. But their scheme was restricted to only three parties and assumes the common reference string (CRS) model in which an RSA key n_{RSA} used for integer commitments must be given with a trusted setup. This was improved by Nishide and Sakurai [83], where they gave a distributed key generation scheme where the RSA modulus does not need to be a product of two primes. Additionally, they showed that their protocol is secure against an active and static adversary corrupting any minority of the parties. A complete solution to the (Pallier) Damgård-Jurik threshold encryption scheme in the two-party setting with security against malicious attack was provided Hazay et al. in 2019 [91]. They further described how their scheme can be extended to the multiparty setting with dishonest majority.

To give a rough idea of the workings of these distributed key generation algorithms, we provide the high-level overview of the basic Nishide and Sakurai scheme [83]. Assume that ν parties $P_1, \ldots, P_\nu$ that perform the following steps to compute the RSA modulus $n = p \cdot q$. Further assume that the parties need 2κ-bits of security for the RSA modulus, where κ is typically taken as 1024. Then the protocol proceeds as follows.

(1) Every party P_i except P_1 picks their own random secrets $p_i, q_i \in [2^{\kappa-1}, 2^{\kappa} - 1]$, such that $p_i \equiv q_i \equiv 0 \mod 4$.
(2) P_1 picks $p_1, q_1 \in [2^{\kappa-1}, 2^{\kappa} - 1]$, such that $p_1 \equiv q_1 \equiv 3 \mod 4$.
(3) The parties agrees on some large prime P', where $P' > \{\nu(3 \times 2^{\kappa-1})\}^2 > 2N$.
(4) The ν parities then uses the BGW protocol [90] to compute

$$N = p \times q = \left(\sum_{i=1}^{\nu} p_i\right) \times \left(\sum_{i=1}^{\nu} q_i\right) \mod P'$$

without revealing p and q. Since, $N < P'$, therefore the parties can compute N.

(5) The parties then uses a distributed bi-primality test to check if N computed in the previous step is a product of two prime or not. If not, the protocol is restarted.

For the distributed key generation for the DIKE voting scheme presented here we suggest [91]. We assume that the players involved in the distributed key generation algorithm are the EAs. As it is reasonable to assume that the EAs would execute the protocols honestly but there could be at most $\lfloor \frac{k-1}{2} \rfloor$ EAs who as dishonest, i.e., we use the honest-but-curious setting.

Received ; revised ; accepted