Distributed Key Generation in the Wild

Aniket Kate[1] Yizhou Huang[2] Ian Goldberg[2]

[1]Max Planck Institute for Software Systems (MPI-SWS), Germany
[2]Cheriton School of Computer Science, University of Waterloo, Canada
aniket@mpi-sws.org, {y226huan, iang}@uwaterloo.ca

Abstract

Distributed key generation (DKG) has been studied extensively in the cryptographic literature. However, it has never been examined outside of the synchronous setting, and the known DKG protocols cannot guarantee safety or liveness over the Internet.

In this work, we present the first realistic DKG protocol for use over the Internet. We propose a practical system model for the Internet and define an efficient verifiable secret sharing (VSS) scheme in it. We observe the necessity of Byzantine agreement for asynchronous DKG and analyze the difficulty of using a randomized protocol for it. Using our VSS scheme and a leader-based agreement protocol, we then design a provably secure DKG protocol. We also consider and achieve cryptographic properties such as uniform randomness of the shared secret and compare static versus adaptive adversary models. Finally, we implement our DKG protocol, and establish its efficiency and reliability by extensively testing it on the PlanetLab platform. Counter to a general non-scalability perception about asynchronous systems, our experiments demonstrate that our asynchronous DKG protocol scales well with the system size and it is suitable for realizing multiparty computation and threshold cryptography over the Internet.

Keywords: asynchronous communication model. computational setting. distributed key generation. uniform randomness. implementation

1 Introduction

Numerous online cryptographic applications require a trusted authority to hold a secret. However, this requirement always leads to the problem of single point of failure and sometimes to the more undesirable problem of key escrow. Solving these problems is of paramount importance while designing systems over the Internet where denial-of-service attacks and malicious entities are widespread. A distributed key generation (DKG) [42] protocol overcomes these problems using a complete distribution of the trust among a set of servers. In essence, an (n, t)-DKG protocol allows a set of n nodes to collectively generate a secret with its *shares* spread over the nodes such that any subset of size greater than a threshold t can reveal or use the shared secret, while smaller subsets do not have any knowledge about it. Unlike the original secret sharing schemes [6, 46], where a *dealer* generates a secret and distributes its shares among the nodes, DKG requires no trusted party.

A DKG protocol is a fundamental building block of both symmetric and asymmetric threshold cryptography. In symmetric-key cryptography, DKG is used to design distributed key distribution centres [38]. Here, a group of servers jointly realize the function of a key distribution centre, which generates and provides encryption keys for secure conferences to clients. In public-key cryptography (PKC), DKG is essential for dealerless threshold public-key decryption and signature schemes [16] and for distributed private-key generation in identity-based cryptography (IBC) [7]. In a threshold

decryption scheme, a private key is distributed among a group such that, given a ciphertext, more than a threshold number of them have to combine their decrypted shares to find the plaintext message. On the other hand, in a threshold signature scheme, the signing key is distributed among a group such that more than a threshold number of them have to combine their partial signatures to sign a message. In distributed key distribution centres and threshold decryption and signature schemes, DKG tackles the problem of *single point of failure*. In IBC, it also mitigates the *key escrow* issue, when it is impractical to trust and rely on a single entity, the *private-key generator* (PKG), to generate and distribute private keys to IBC clients. A distributed PKG becomes necessary when IBC is used in practical systems—outside the usual organizational settings—such as key distribution in ad-hoc networks [32] or pairing-based onion routing [30]. DKG is also an important primitive in distributed pseudo-random functions [38], which are useful in distributed coin tossing algorithms [11], random oracles [39] and multiparty computation (MPC) [26].

Although various theoretical aspects of DKG have been thoroughly researched for the last two decades, the systems aspects have been largely ignored. Existing DKG protocols rely on assumptions like synchronous communication (bounded communication delay, with known bounds) or ask for a reliable broadcast channel. As these prerequisites are not readily available over the Internet, the existing DKG protocols are not suitable for the Internet-based applications and there is no DKG available or used in practice yet. In this paper, we design and implement a practical DKG protocol for use over the Internet.

Contributions. In a preliminary version [28] of this paper, we defined the first practical DKG protocol for use over the Internet. Here, we formally prove its safety and liveness properties and present a practical mechanism to obtain cryptographically important property of uniform randomness of the shared secret. We also implement and verify the practicality of our DKG protocol over the PlanetLab platform.

- As our first contribution, we define a realistic system model over the Internet (Section 2). We combine the standard Byzantine adversary with crash recovery and network failures in an asynchronous setting. We also analyze the asynchronous versus partially synchronous dichotomy for the Internet and justify the choice of treating crashes and network failures separately.

- We present a VSS scheme (HybridVSS) that works in our system model (Section 4). Observing the necessity of a protocol for agreement on a set for asynchronous DKG, we define and prove a practical DKG protocol (HybridDKG) for use over the Internet (Section 5). We use a leader-based agreement scheme in our DKG, as we observe a few pragmatic issues with the usually suggested randomized agreement schemes.

- We also consider the uniform randomness of the shared secret in DKG and modify our HybridDKG protocol to achieve uniform randomness in the random oracle model (Section 5.3).

- Finally, we implement our HybridDKG protocol and test its performance over the PlanetLab platform (Section 6). To the best of our knowledge, this is the first time that a distributed *cryptographic* protocol in the asynchronous setting has been tested with up to 70 nodes (replicas) spread across multiple continents. We observe that the HybridDKG protocol scales well in terms of the execution time and the system load, and it can be used to realize threshold cryptography and MPC over the Internet. We also present some system-level optimizations for HybridDKG and consider the system's resilience against denial-of-service (DoS) attacks and Sybil attacks. Our implementation is available from our web site [2].

2 Assumptions and System Model

In this section, we discuss the assumptions and the system model for our protocols, giving special attention to their practicality over the Internet. This generic system model will also be applicable to many other distributed protocols over the Internet.

2.1 Communication Model

Our DKG protocol should be deployable over the Internet. The expected message-transfer delay and the expected clock offset there (a few seconds, in general) is significantly smaller than a possible system utility period which may extend up to a few days. With such an enormous difference, a failure of the network to deliver a message within a fixed time bound can be treated as a failure of the sender; this may lead to a retransmission of the message after appropriate timeout signals. As this is possible without any significant loss in the synchrony of the system, the asynchronous communication assumption seems to be unnecessarily pessimistic here. It is tempting to treat the Internet as a *synchronous network* (bounded message delivery delays and processor speeds) and develop more efficient protocols using well-known message delivery time bounds and system run-time assumptions.

Deciding these time bounds correctly is a difficult problem to solve. Further, even if it is possible to determine tight bounds between the optimistic and pessimistic cases, there is a considerable difference between the selected time bounds and the usual computation and communication time. Protocols based on the synchronous assumption invariably use these time bounds in their definitions, while those based on the asynchronous assumption solely use numbers and types of messages.

A real-world adversary, with knowledge of any time bounds used, can always slow down the protocols by delaying its messages to the verge of the time bounds. In asynchronous protocols, although it is assumed that the adversary manages the communication channels and can delay messages as it wishes, a real-world adversary cannot control communication channels for all the honest nodes. It is practical to assume that network links between most of the honest nodes are perfect. Consequently, even if the adversary delays its messages, an asynchronous protocol completes without any delay when honest nodes communicate promptly. Thus, the asynchrony assumption may increase message complexity or the *latency degree* (number of communication rounds), but in practice does not increase the actual execution time. Observing this, we use the asynchronous communication assumption for our protocols.

Weak Synchrony Assumption (only for liveness). For *liveness* (the protocol eventually terminates), but not *safety* (the protocol does not fail or produce incorrect results), we need a (weak) synchrony assumption. Otherwise, we could implement consensus in the asynchronous model with adversary nodes, which is impossible [21]. We use a weak synchrony assumption by Castro and Liskov [14] to achieve liveness. Let $delay(T)$ be the time between the moment T when a message is sent for the first time and the moment when it is received by its destination. The sender keeps retransmitting the message until it is received correctly. We assume that $delay(T)$ *does not grow faster than* T *indefinitely.* Assuming that network faults are eventually repaired and DoS attacks eventually stop, this assumption seems to be valid in practice.

Note that this assumption is also strictly weaker than the partially synchronous communication assumptions defined by Dwork et al. [18]). In their first partial synchrony assumption message delivery delays and processor speeds are bounded, but the bounds are not known a priori, while in their second partial synchrony assumption, the bounds are known, but are only guaranteed to hold starting some unknown time. In the asymptotic notation, both of these delay notations are $\Theta(1)$.

However, the weak synchrony assumption is $o(T)$, and therefore, weaker than the partial synchrony assumptions.

2.2 Byzantine Adversary, Crash-Recoveries and Link Failures

Most of the distributed computing protocols in the literature assume a t-limited *Byzantine adversary* and a compromised node remains Byzantine and unused after recovery. We also aim at proactive security for our DKG, where the t-limited mobile Byzantine adversary can change its choice of t nodes as time progresses. There, a node compromised during a phase remains unused, after recovery, for the remainder of that phase as its share is already compromised. Any intra-phase share modification for a recovered node leads to intra-phase share modification to all the nodes, which is unacceptable in general.

This does not model failures over the Internet in the best way. Other than malicious attacks leading to compromise, some nodes (say f of them) may just crash silently without showing arbitrary behaviours or get disconnected from the rest of the network due to network failures or partitioning. Importantly, secrets at these f nodes are not available to the adversary and modelling them as Byzantine failures not only leads to sub-optimal resilience of $n \geq 3(t+f)+1$ instead of $n \geq 3t+2f+1$, but it also increases the bit complexity with added security requirements ($t+f$ instead of t). Keeping such nodes inactive, after their recovery, until the start of next phase is not ideal. This prompts us to use a *hybrid model*.

Our system adopts the hybrid model of Backes and Cachin [4], but with a modification to accommodate *broken links*. From any honest node's perspective, a crashed node behaves similarly to a node whose link with it is broken and we model link failures in the form of crashes. For every broken link between two nodes, we assume that at least one of two nodes is among the list of currently crashed nodes. A node that is crashed means that *some* of its links are down, not necessarily that they *all* are. Further, all non-Byzantine nodes may crash and recover repeatedly with a maximum of f crashed nodes at any instant and a recovering honest node recovers from a well-defined state using, for example, a read-only memory. In addition to these nodes that crash completely (losing any keys they may have) and subsequently recover from that state, other non-Byzantine nodes may remain up, but have *some* of their links down; those links remain down throughout the protocol (otherwise, they are simply normal asynchronous links), and the nodes may or may not realize that the links are down. These nodes will not need to recover their keys, since they have not lost them. As long as the size of the list of crashed nodes, including all completely crashed nodes, and one of the endpoints of every broken link, does not exceed f at any time, our protocols will succeed. We also assume that the adversary (eventually) delivers all the messages between two uncrashed nodes.

Formally, we consider an asynchronous network of $n \geq 3t+2f+1$ nodes $P_1, \ldots, P_n$ of which the adversary may corrupt up to t nodes during its existence and may crash another f nodes at any time. For $f = 0$, $3t+1$ nodes are required as a differentiation between slow honest nodes and Byzantine nodes is not possible in an asynchronous network, while for $t = 0$, $2f+1$ nodes are mandatory to achieve consistency. At least $n-t-f$ nodes, which are not in the crashed state at the end of a protocol, are termed *finally up* nodes.

2.3 Cryptographic Assumptions and Setup

Definition 2.1. *A function $\epsilon(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is called* negligible *if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < 1/\kappa^c$ for all $\kappa > \kappa_0$.*

In the remainder of the paper, $\epsilon(\cdot)$ denotes a negligible function. Our adversary is computationally bounded with a security parameter κ and it has to solve the DLog problem [37, Sec. 3.6] (Definition 2.2) for the security parameter κ to break the security of the protocols. Further, as discussed in Section 3.3, our t-limited adversary is also *static*.

Definition 2.2. *Let p be a prime whose size is linear in κ. Given a generator g of a multiplicative group $\mathbb{G}$ of order p and $a \in \mathbb{Z}_p^*$, the* discrete logarithm (DLog) *assumption suggests that* $\Pr[\mathcal{A}_{DLog}(g, g^a) = a] = \epsilon(\kappa)$ *for every polynomial-time adversary $\mathcal{A}_{DLog}$.*

We use a PKI infrastructure in the form of a PKI hierarchy with an external certifying authority or web-of-trust among nodes to achieve authenticated and confidential communication with TLS links, and message authentication with any digital signature scheme secure against adaptive chosen-message attacks [25]. Each node also has a unique identifying index. We assume that indices and public keys for all nodes are publicly available in the form of certificates. It is possible to achieve similar security guarantees in a symmetric-key setting with long-term keys.

2.4 Complexity Assumptions

In our hybrid model, nodes may crash and recover repeatedly with a maximum of f crashed nodes at any instant. We still need to bound the crashes as an unbounded number of crashes can cause the protocol execution time to be unbounded. We restrict the adversary by a function $d(\cdot)$ that represents the maximum number of crashes that it is allowed to perform during its lifetime and parametrize $d(\cdot)$ with the security parameter κ. Note that, based on the system requirements, it is possible to make $d(\cdot)$ constant or parametrize it with n, t or f.

Work done by honest parties can be measured by a *protocol statistic* X, which is a family of real-valued non-negative random variables $\{X_A(\kappa)\}$, parametrized by adversary A and κ. Each $X_A(\kappa)$ is a random variable induced by running the system with A. A protocol statistic X is called *uniformly bounded* if there exists a fixed polynomial $T(\kappa)$ such that for all adversaries A, there is a negligible function ϵ_A, such that for all $\kappa \geq 0$, $Pr[X_A(\kappa) > T(\kappa)] \leq \epsilon_A(\kappa)$. As we consider a computationally bounded adversary, we aim at polynomially bounded system execution space and time and bounding protocol complexities by a polynomial in the adversary's running time.

For crash-recovery situations, Backes and Cachin introduce the notion of *d-uniformly bounded statistics* [4, Def. 1]. Here, a bounded protocol statistic X is considered to be d-uniformly bounded (by T_1 and T_2) for a function $d(\kappa)$ if there exist two fixed polynomials T_1 and T_2 such that for all adversaries A, there exists a negligible function $\epsilon_A(\kappa)$ such that for all $\kappa \geq 0$, $Pr[X_A(\kappa) > d(\kappa)T_1(\kappa) + T_2(\kappa)] \leq \epsilon_A(\kappa)$. In order words, the complexity of the protocol is uniformly bounded if no crash occurs (which is ensured by T_2), and the computational overhead caused by each crash is also uniformly bounded (ensured by T_1). Similar to [4], we expect that the bit complexity of a protocol is d-uniformly bounded for some polynomial d.

3 Background

In this section, we survey the concepts of VSS, homomorphic commitments and DKG.

3.1 Verifiable Secret Sharing

The notion of secret sharing was introduced independently by Shamir [46] and Blakley [6]. Since then, it has remained an important topic in security research.

Definition 3.1. $(n, t+\delta, t)$**-Secret Sharing**. *For integers n, t and δ such that $n \geq t+\delta > t \geq 0$, an* $(n, t+\delta, t)$-secret sharing *scheme is a protocol used by a dealer to share a secret s among a set of n nodes in such a way that any subset of $t+\delta$ or more nodes can compute the secret s, but subsets of size t or fewer have no information about s.*

A secret sharing scheme with $\delta = 1$ is called a *threshold* secret sharing scheme, and for $\delta > 1$, it is called a *ramp* secret sharing scheme. In this work, we concentrate on threshold secret sharing and denote it as (n, t)*-secret sharing* instead of $(n, t+1, t)$-secret sharing. Note that all polynomial-based threshold secret sharing schemes can easily be converted to ramp secret sharing schemes [47].

In some secret sharing applications, clients may need to verify a consistent dealing to prevent malicious behaviour by the dealer. A scheme with such a verifiability guarantee is known as *verifiable secret sharing* (VSS) scheme [15]. Feldman [19] developed the first non-interactive and efficient VSS scheme and Pedersen [41] presented a modification to it.

Definition 3.2. *An* (n, t)-VSS *scheme consists of two phases: the* sharing *(*Sh*) phase and the* reconstruction *(*Rec*) phase.*

Sh phase. *A dealer P_d distributes a secret $s \in \mathcal{K}$ among n nodes, where $\mathcal{K}$ is a sufficiently large key space. At the end of the* Sh *phase, each honest node P_i holds a share s_i of the distributed secret s.*

Rec phase. *In this phase, each node P_i broadcasts its secret share s'_i and a reconstruction function is applied in order to compute the secret $s = \mathsf{Rec}(s'_1, s'_2, \ldots, s'_n)$ or output $\perp$ indicating that P_d is malicious. For honest nodes $s'_i = s_i$, while for malicious nodes s'_i may be different from s_i or even absent.*

It has two security requirements:

Secrecy (VSS-wS). *A* t-limited *adversary who can compromise t nodes cannot compute s during the* Sh *phase.*

Correctness (VSS-C). *The reconstructed value should be equal to the shared secret s or every honest node concludes that P_d is malicious by outputting $\perp$.*

We consider VSS schemes in the computational complexity setting. Here, any malicious behaviour by P_d is caught by the honest nodes in the Sh phase itself and the VSS-C property simplifies to the following: the reconstructed value should be equal to the shared secret s. Further, many VSS applications avoid participation by all parties during the Rec phase. It is required that shares from any $t+1$ honest nodes (or any $2t+1$ nodes) is sufficient to reconstruct s. Therefore, we mandate the correctness property that we refer as *strong correctness* requirement.

Strong Correctness (VSS-SC). The same unique value s is reconstructed regardless of the subset of nodes (of size greater than $2t$) chosen by the adversary in the Rec algorithm.

Further, some VSS schemes achieve a stronger secrecy guarantee.

Strong Secrecy (VSS-S). The adversary who can compromise t nodes does not have any more information about s except what is implied by the public parameters.

Asynchronous VSS. Although the literature for VSS has been vast, VSS in the asynchronous communication model has not yet received the required attention. Canetti and Rabin [13] developed the first complete asynchronous VSS scheme with unconditional security. However, this scheme and

its successors [3,40], due to their $\Omega(n^5)$ bit complexities, are prohibitively expensive for any realistic use.

Compromising the unconditional security assumption, Cachin et al. [9] (AVSS), Zhou et al. [50] (APSS), and more recently Schultz et al. [45] (MPSS) suggested more practical asynchronous VSS schemes. Of these, the APSS protocol is impractical for any reasonable system size, as it uses a combinatorial secret sharing scheme by Ito, Saito and Nishizeki [27], which leads to an exponential $\binom{n}{t}$ factor in its message complexity. MPSS, on the other hand, is developed for a more mobile setting where set of the system nodes has to change completely between two consecutive phases to maintain the secrecy and correctness properties.

AVSS is the most general and practical scheme in the asynchronous communication model against Byzantine adversaries, but it does not handle crash recoveries. It assimilates a bivariate polynomial into Bracha's reliable broadcast [8] and can provide complete flexibility with the sets used without hampering the security. In asynchronous VSS, any two participants need to verify the dealer's commitment with each other to achieve correctness; thus, a protocol with $o(n^2)$ message complexity does not seem to be possible. Therefore, AVSS, with its optimal message complexity, forms the basis for our HybridVSS and HybridDKG protocols appearing in sections 4 and 5.

3.2 Homomorphic Commitments

A verification mechanism for a consistent dealing is fundamental to VSS. It is achieved using distributed computing techniques in the unconditional setting. In the computational setting, *homomorphic commitments* provide an efficient alternative. Let $\mathcal{C}(\alpha, [r]) \in \mathbb{G}$ be a homomorphic commitment to $\alpha \in \mathbb{Z}_p$, where r is an optional randomness parameter and $\mathbb{G}$ is a (multiplicative) group. For such a homomorphic commitment, given $C_1 = \mathcal{C}(\alpha_1, [r_1])$ and $C_2 = \mathcal{C}(\alpha_2, [r_2])$, we have $C_1 \cdot C_2 = \mathcal{C}(\alpha_1 + \alpha_2, [r])$.

VSS protocols utilize two forms of commitments. Let g and h be two random generators of $\mathbb{G}$. Feldman, for his VSS protocol [19], used the DLog commitment scheme of the form $\mathcal{C}_{\langle g \rangle}(\alpha) = g^\alpha$ with computational security under the DLog assumption and unconditional share integrity. Pedersen [42] presented another commitment of the form $\mathcal{C}_{\langle g,h \rangle}(\alpha, r) = g^\alpha h^r$ with unconditional security but computational integrity under the DLog assumption. In PKC based on computational assumptions, with adversarial access to the public key, unconditional security of the secret (private key or master key) is impossible. Further, in VSS schemes based on Pedersen commitments, in order to randomly select the generator h, an additional round of communication is required during bootstrapping. Consequently, in our scheme, we use simple and efficient DLog commitments, except during a special case described in Section 5.3.

It is also possible to prove equality of values committed using Pedersen commitments and DLog commitments using non-interactive zero-knowledge (NIZK) proofs [20]. Here, given a DLog commitment $\mathcal{C}_{\langle g \rangle}(s) = g^s$ and a Pedersen commitment $\mathcal{C}_{\langle g,h \rangle}(s, r) = g^s h^r$ to the same value s for generators $g, h \in \mathbb{G}$ and $s, r \in \mathbb{Z}_p$, a prover proves that she knows s and r such that $\mathcal{C}_{\langle g \rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h \rangle}(s, r) = g^s h^r$. We denote this by

$$\text{NIZKPK}_{\equiv Com}(s, r, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r)) = \pi_{\equiv Com} \tag{1}$$

The proof is equivalent to zero-knowledge proofs of knowledge used by Canetti et al. [12] in their adaptive secure DKG. It is generated as follows:

- Pick $v_1, v_2 \in_R \mathbb{Z}_p$, and let $t_1 = g^{v_1}$ and $t_2 = h^{v_2}$.
- Compute hash $c = \text{H}_{\equiv Com}(g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r),\ t_1, t_2)$, where $\text{H}_{\equiv Com} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash function.

- Let $u_1 = v_1 - c \cdot s$ and $u_2 = v_2 - c \cdot r$.
- Send the proof $\pi_{\equiv Com} = (c, u_1, u_2)$ along with $\mathcal{C}_{\langle g \rangle}(s)$ and $\mathcal{C}_{\langle g,h \rangle}(s, r)$.

The verifier checks this proof (given $\pi_{\equiv Com}$, g, h, $\mathcal{C}_{\langle g \rangle}(s)$, $\mathcal{C}_{\langle g,h \rangle}(s, r)$) as follows:

- Let $t'_1 = g^{u_1} \mathcal{C}_{\langle g \rangle}(s)^c$ and $t'_2 = h^{u_2} (\frac{\mathcal{C}_{\langle g,h \rangle}(s,r)}{\mathcal{C}_{\langle g \rangle}(s)})^c$.
- Accept the proof as valid if $c = \mathrm{H}_{\equiv Com}(g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r), t'_1, t'_2)$.

3.3 Distributed Key Generation

Pedersen [42] introduced the concept of distributed key generation (DKG) and developed a DKG scheme. Here, each node runs a VSS instance and distributed shares are added at the end to generate a combined shared secret *without a dealer*. Unlike VSS, DKG requires no trusted party.

Definition 3.3. *An* (n, t)-DKG *scheme consists of two phases: the* sharing *(*Sh*) phase and the* reconstruction *(*Rec*) phase.*

Sh phase. *Every node P_i distributes a secret $z_i \in \mathcal{K}$ among n nodes, where $\mathcal{K}$ is a sufficiently large additive cyclic group. At the end of the* Sh *phase, each honest node P_i holds a share s_i of the distributed secret s, which is a pre-decided linear combination of the shared z_i values.*

Rec phase. *Each node P_i broadcasts its secret share s'_i and a reconstruction function is applied in order to compute the secret $s = \mathsf{Rec}(s'_1, s'_2, \ldots, s'_n)$. For honest nodes $s'_i = s_i$, while for malicious nodes s'_i may be different from s_i or even absent.*

As discussed in the introduction, DKG has numerous application in cryptography. In the paper, we concentrate on threshold cryptosystems in the DLog setting. In this setting, having a cyclic group $\mathbb{G}$ of prime order p, a DKG protocol generates secret sharing of a secret $s \in \mathbb{Z}_p$, and publishes $Y = (g, g^s)$ for $g \in_R \mathbb{G}$ as the corresponding public key. Gennaro et al. [24, Section 4.1] suggested the following secrecy and correctness requirements for an (n, t)-DKG protocol in the DLog setting:

Correctness (DKG-C).

1. There is an efficient algorithm that on input shares from $2t + 1$ nodes and the public information produced by the DKG protocol outputs the same unique value s, even if up to t shares are submitted by malicious nodes.[1]
2. At the end of Sh phase, all honest nodes have the same value of the public key $Y = g^s$, where s is the unique secret guaranteed above.
3. s and Y are uniformly distributed in $\mathbb{Z}_n$ and $\mathbb{G}$, respectively.

Secrecy (DKG-S). No information about the secret s can be learned by the adversary except for what is implied by $Y = g^s$.

Gennaro et al. [22] found that DKGs based on the Feldman VSS methodology do not guarantee uniform randomness of the shared secret key or DKG-S. Further, they observed that the use of digital signatures or other slight modifications do not provide any additional security to the Pedersen DKG [42] and presented a simplification using just the original Feldman VSS called the Joint

[1] Note that this is the stronger version, which they define in the latter part of [24, Section 4.1].

Feldman DKG (JF-DKG) as the representative scheme. They then defined a new DKG protocol combining discrete logarithm and Pedersen commitments [42], which guarantees the uniform randomness property by increasing the *latency* (number of communication rounds) of the DKG protocol by one.

In [23], the same set of authors observed that the Pedersen DKG and JF-DKG produce hard instances of the DLog problem, which may be sufficient for the security of some threshold cryptographic schemes. Although a reduction in their JF-DKG security proof is not tight, as they discussed in [24], an elliptic-curve implementation of JF-DKG with appropriately increased key sizes is still faster than the modification they suggested in [22]. In this weaker version of DKG, the third correctness property is absent and the secrecy requirement weakens to the following:

Weak Correctness (DKG-wC).

1. There is an efficient algorithm that on input shares from $2t + 1$ nodes and the public information produced by the DKG protocol outputs the same unique value s, even if up to t shares are submitted by malicious nodes.
2. At the end of Sh phase, all honest nodes have the same value of public key $Y = g^s$, where s is the unique secret guaranteed above.

Weak Secrecy (DKG-wS). The adversary with t shares and the public key $Y = g^s$ cannot compute the secret s.

Knowing this efficiency and secrecy tradeoff, we define two versions of our DKG constructions: an efficient DKG with weak correctness and weak secrecy, and a DKG with uniform randomness of the shared secret and strong secrecy.

It is important to discuss the relationship between the various secrecy and correctness notions that we present in the above two sections. For secrecy, VSS-wS and VSS-S are equivalent to DKG-wS and DKG-S respectively. For correctness, VSS-SC is included in both DKG-wS and DKG-S; the weaker VSS-C is a property common only in the unconditional VSS protocols.

Our protocols and most of the distributed cryptographic protocols in the literature are not considered secure against an *adaptive* adversary that may choose its t compromisable nodes as a protocol is getting executed. As elaborated in [24, Section 4.4], this is only because their (simulation-based) security proofs do not go through when the adversary can corrupt nodes adaptively. In [19, Section 9.3], Feldman claimed that his VSS protocol is also secure against adaptive adversaries even though his simulation-based security proof did not work out. Canetti *et al.* [12] presented a distributed protocol methodology that is provably secure against adaptive adversaries using interactive zero-knowledge proofs and a proof technique that may rewind the adversary a polynomial number of times. However, their methodology in general adds at least two more communication rounds to a protocol, which can severely deteriorate the system performance.

On the other hand, all of the protocols in the literature that are proven secure only against a static adversary have remained unattacked by an adaptive adversary for the last 25 years. Gaining some confidence from this fact and giving importance to efficiency, we stick to protocols provably secure only against a static adversary in our work.

4 VSS for the Hybrid Model—HybridVSS

VSS is the fundamental building block for DKG. Our VSS protocol modifies the AVSS protocol [9] for our hybrid model. We include recovery messages similar to those from the reliable broad-

cast protocol by Backes and Cachin [4]. We achieve a constant-factor reduction in the protocol complexities using symmetric bivariate polynomials.

4.1 Construction

As usual, our VSS protocol is composed of a sharing protocol (Sh) and a reconstruction protocol (Rec). In protocol Sh, a dealer P_d upon receiving an input message $(P_d, \tau, \mathsf{in}, \mathsf{share}, s)$, shares the secret s, where a counter τ and the dealer identity P_d forms a unique session identifier. Node P_i finishes the Sh protocol by outputting a $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}, s_i)$ message, where $\mathcal{C}$ is the commitment and s_i is its secret share. Any time after that, upon receiving an input message $(P_d, \tau, \mathsf{in}, \mathsf{reconstruct})$, P_i starts the Rec protocol. The Rec protocol terminates for a node P_i by outputting a message $(P_d, \tau, \mathsf{out}, \mathsf{reconstructed}, z_i)$, where z_i is P_i's reconstructed value of the secret s.

Note that, for the simplicity of discussion, we use DLog commitments instead of the Pedersen commitments used in the original AVSS protocol and achieve VSS-wS secrecy (as defined in Section 3.1). It is easily possible to use Pedersen commitments instead and achieve VSS-S secrecy.

Definition 4.1. *In session (P_d, τ), protocol VSS in our hybrid model (HybridVSS) having an asynchronous network of $n \geq 3t + 2f + 1$ nodes with a t-limited Byzantine adversary and f-limited crashes and network failures satisfies the following conditions:*

Liveness. *If the dealer P_d is honest and finally up in the sharing stage of session (P_d, τ), then all honest finally up nodes complete protocol Sh.*

Agreement. *If some honest node completes protocol Sh of session (P_d, τ), then all honest finally up nodes will eventually complete protocol Sh in session (P_d, τ). If all honest finally up nodes subsequently start protocol Rec for session (P_d, τ), then all honest finally up nodes will finish protocol Rec in session (P_d, τ).*

Correctness. *Once $t + 1$ honest nodes complete protocol Sh of session (P_d, τ), then there exists a fixed value z such that*

- *if the dealer is honest and has shared secret s in session (P_d, τ), then $z = s$, and*
- *if an honest node P_i reconstructs z_i in session (P_d, τ), then $z_i = z$.*

This is equivalent to the strong correctness (VSS-SC) property defined in Section 3.1.

Secrecy. *If an honest dealer has shared secret s in session (P_d, τ) and no honest node has started the Rec protocol, then, except with negligible probability, the adversary cannot compute the shared secret s. This is equivalent to the VSS-wS secrecy property defined in Section 3.1.*

Efficiency. *The bit complexity for any instance of HybridVSS is d-uniformly bounded.*

Figure 1 describes the Sh and the Rec phases of HybridVSS. We use pseudo-code notation and include a special condition **upon** to define actions based on messages received from other nodes or system events. $\mathcal{C}$ is a matrix of commitment entries and $e_{\mathcal{C}}$ and $r_{\mathcal{C}}$ are P_i's associated counters for echo and ready messages, respectively. In order to facilitate recovery of the crashed nodes, each node P_i stores all outgoing messages along with their intended recipients in a set $\mathcal{B}$. $\mathcal{B}_\ell$ indicates the subset of $\mathcal{B}$ intended for the node P_ℓ. Counters cnt and cnt_ℓ keep track of the numbers of overall help requests and help requests sent by each node P_ℓ respectively. In the presence of crash-recoveries and malicious nodes, a node may receive a message identified by P_d, τ and $\mathcal{C}$ multiple times. According to the protocol definition, it processes the message only the first time it receives it, and ignores subsequent receipts. We use the following predicates in our protocol.

Sh protocol for node P_i and session (P_d, τ)

upon initialization:
  for all $\mathcal{C}$ **do**
    $A_{\mathcal{C}} \leftarrow \emptyset$; $e_{\mathcal{C}} \leftarrow 0$; $r_{\mathcal{C}} \leftarrow 0$
    $cnt \leftarrow 0$; $cnt_\ell \leftarrow 0$ for all $\ell \in [1, n]$

upon a message $(P_d, \tau, \mathsf{in}, \mathsf{share}, s)$: /* only P_d */
  choose a symmetric bivariate polynomial $\phi(x, y) = \sum_{j,\ell=0}^{t} \phi_{j\ell} x^j y^\ell \in_R \mathbb{Z}_p[x, y]$ and $\phi_{00} = s$
  $\mathcal{C} \leftarrow \{\mathcal{C}_{j\ell}\}_{j,\ell=0}^{t}$ where $\mathcal{C}_{j\ell} = g^{\phi_{j\ell}}$ for $j, \ell \in [0, t]$
  for all $j \in [1, n]$ **do**
    $a_j(y) \leftarrow \phi(j, y)$; send the message $(P_d, \tau, \mathsf{send}, \mathcal{C}, a_j)$ to P_j

upon a message $(P_d, \tau, \mathsf{send}, \mathcal{C}, a)$ from P_d (first time):
  if $\mathsf{verify\text{-}poly}(\mathcal{C}, i, a)$ **then**
    for all $j \in [1, n]$ **do**
      send the message $(P_d, \tau, \mathsf{echo}, \mathcal{C}, a(j))$ to P_j

upon a message $(P_d, \tau, \mathsf{echo}, \mathcal{C}, \alpha)$ from P_m (first time):
  if $\mathsf{verify\text{-}point}(\mathcal{C}, i, m, \alpha)$ **then**
    $A_{\mathcal{C}} \leftarrow A_{\mathcal{C}} \cup \{(m, \alpha)\}$; $e_{\mathcal{C}} \leftarrow e_{\mathcal{C}} + 1$
    if $e_{\mathcal{C}} = \lceil \frac{n+t+1}{2} \rceil$ **and** $r_{\mathcal{C}} < t + 1$ **then**
      Lagrange-interpolate $\overline{a}$ from $A_{\mathcal{C}}$
      for all $j \in [1, n]$ **do**
        send the message $(P_d, \tau, \mathsf{ready}, \mathcal{C}, \overline{a}(j))$ to P_j

upon a message $(P_d, \tau, \mathsf{ready}, \mathcal{C}, \alpha)$ from P_m (first time):
  if $\mathsf{verify\text{-}point}(\mathcal{C}, i, m, \alpha)$ **then**
    $A_{\mathcal{C}} \leftarrow A_{\mathcal{C}} \cup \{(m, \alpha)\}$; $r_{\mathcal{C}} \leftarrow r_{\mathcal{C}} + 1$
    if $r_{\mathcal{C}} = t + 1$ **and** $e_{\mathcal{C}} < \lceil \frac{n+t+1}{2} \rceil$ **then**
      Lagrange-interpolate $\overline{a}$ from $A_{\mathcal{C}}$
      for all $j \in [1, n]$ **do**
        send the message $(P_d, \tau, \mathsf{ready}, \mathcal{C}, \overline{a}(j))$ to P_j
    else if $r_{\mathcal{C}} = n - t - f$ **then**
      $s_i \leftarrow \overline{a}(0)$; output $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}, s_i)$

upon a message $(P_d, \tau, \mathsf{in}, \mathsf{recover})$:
  send the message $(P_d, \tau, \mathsf{help})$ to all the nodes
  send all messages in $\mathcal{B}$

upon a message $(P_d, \tau, \mathsf{help})$ from P_ℓ:
  if $cnt_\ell \leq d(\kappa)$ **and** $cnt \leq (t + 1)d(\kappa)$ **then**
    $cnt_\ell \leftarrow cnt_\ell + 1$; $cnt \leftarrow cnt + 1$
    send all messages of $\mathcal{B}_\ell$

Rec protocol for node P_i and session (P_d, τ)

upon a message $(P_d, \tau, \mathsf{in}, \mathsf{reconstruct})$:
  $c \leftarrow 0$; $S \leftarrow \emptyset$
  for all $j \in [1, n]$ **do**
    send the message $(P_d, \tau, \mathsf{reconstruct\text{-}share}, s_i)$ to P_j

upon a message $(P_d, \tau, \mathsf{reconstruct\text{-}share}, \sigma)$ from P_m:
  if $(g^\sigma = \prod_{j=0}^{t} (\mathcal{C}_{j0})^{m^j})$ **then**
    $S \leftarrow S \cup \{(m, \sigma)\}$; $c \leftarrow c + 1$
    if $c = t + 1$ **then**
      interpolate constant term (z_i) from S
      output $(P_d, \tau, \mathsf{out}, \mathsf{reconstructed}, z_i)$

Figure 1: Protocol HybridVSS

verify-poly$(\mathcal{C}, i, a)$ verifies that the given polynomial a of P_i is consistent with the commitment $\mathcal{C}$. Here, $a(y) = \sum_{\ell=0}^{t} a_\ell y^\ell$ is a degree-t polynomial. The predicate is true if and only if $g^{a_\ell} = \prod_{j=0}^{t} (\mathcal{C}_{j\ell})^{i^j}$ for all $\ell \in [0, t]$.

verify-point$(\mathcal{C}, i, m, \alpha)$ verifies that the given value α corresponds to the polynomial evaluation $\phi(m, i)$. It is true if and only if $g^\alpha = \prod_{j,\ell=0}^{t} (\mathcal{C}_{j\ell})^{m^j i^\ell}$.

Note that the AVSS and our HybridVSS schemes use bivariate polynomials, as they guarantee that the interpolated polynomials $\overline{a}$ are of degree t or less. If the univariate polynomials with the constant term equal to the secret s are instead used by dealers in the send messages and the univariate polynomials with the constant term equal to their shares s_i are instead used by nodes in the echo and ready messages, then degrees of the interpolated polynomials $\overline{a}$ will be greater than t with overwhelming probability.

4.2 Analysis

The main theorem for HybridVSS is as follows.

Theorem 4.1. *With the DLog assumption, protocol HybridVSS implements asynchronous VSS in the hybrid model for $n \geq 3t + 2f + 1$.*

Proof. We need to show liveness, agreement, correctness, secrecy, and efficiency. We combine proof strategies from AVSS [9, Sec. 3.3] and reliable broadcast [4, Sec. 3.3] to achieve this. We start by referring to two lemmas.

Lemma 4.1 (Lemma 1 [4])**.** *Let P_i be a finally up party during session (P_d, τ). Then every distinct message sent to P_i by another finally up party P_j during session (P_d, τ) will be received by P_i in a non-crashed state, provided all associated messages are delivered.*

Lemma 4.2 (Lemma 2 [9])**.** *Suppose an honest node P_i sends a ready message containing commitment $\mathcal{C}_i$ and a distinct honest node P_j sends a ready message containing commitment $\mathcal{C}_j$. Then $\mathcal{C}_i = \mathcal{C}_j$.*

Liveness. Here, we prove that if the dealer P_d is honest and finally up during the sharing stage of session (P_d, τ), then all honest finally up nodes complete protocol Sh.

We assume that the dealer P_d is honest and finally up. According to Lemma 4.1, send messages of the form $(P_d, \tau, \mathsf{send}, \mathcal{C}, a_i)$ sent by P_d to each finally up node P_i will eventually be received and verified by each such P_i. Each of these honest and finally up nodes (at least $n - t - f$) will send an echo message of the form $(P_d, \tau, \mathsf{echo}, \mathcal{C}, a_i(j))$ to each system node P_j. Using Lemma 4.1, every finally up honest node will thus receive at least $n - t - f$ valid echo messages. A valid echo and ready message is one that satisfies verify-point. As $n - t - f \geq \lceil \frac{n+t+1}{2} \rceil$ for $n \geq 3t + 2f + 1$, every honest finally up node P_j will send ready message $(P_d, \tau, \mathsf{ready}, \mathcal{C}, \overline{a}_j(m))$ to every system node P_m as either the received echo messages are greater than required bound ($\lceil \frac{n+t+1}{2} \rceil$) or it has already received $t + 1$ ready messages. As all ready messages will be eventually received by the finally up nodes according to Lemma 4.1, each finally up honest node will receive at least $n - t - f$ verifiably correct ready messages. Consequently, each honest finally up node will complete the protocol Sh by outputting $(P_d, \tau, \mathsf{shared})$ messages.

Agreement. We first show that if some honest node completes protocol Sh of (P_d, τ), then all honest finally up nodes will eventually complete protocol Sh during session (P_d, τ). An honest node completes the sharing when it receives $n - t - f$ valid ready messages. At least $t + f + 1$ of those have been sent by honest nodes. Using the definitions of verify-poly and verify-point, the

honest node sends only valid ready messages. Further, when sending, an honest node sends ready messages to all system nodes. Thus, using Lemma 4.2, every honest finally up node receives at least $t+f+1$ valid ready messages with the same commitment $\mathcal{C}$ and sends a ready message containing $\mathcal{C}$. Consequently, every honest finally up node receives $n-t-f$ valid ready messages with commitment $\mathcal{C}$ and completes the Sh protocol.

For protocol Rec, we show that if all honest finally up nodes subsequently begin protocol Rec for session (P_d, τ), then all honest finally up nodes will finish protocol Rec during session (P_d, τ) by reconstructing s'_i. As discussed above, at the end of the sharing step, every honest finally up node P_i computes the same commitment $\mathcal{C}$. Moreover, P_i has received enough valid echo or ready messages with commitment $\mathcal{C}$ and it computes valid ready messages and a valid share s_i with respect to $\mathcal{C}$ (s_i such that $(g^{s_i} = \prod_{j=0}^{t}(\mathcal{C}_{j0})^{m^j})$ holds). Thus, if all honest servers subsequently start the reconstruction stage, then every server receives enough valid shares to reconstruct some value, provided the adversary delivers all associated messages.

Correctness (VSS-SC). Suppose an honest dealer has shared a degree-t symmetric bivariate polynomial $\phi(x, y)$ with constant term equal to the shared secret s. As the dealer is honest, an echo message that an honest node P_i receives from another honest node P_j contains $\mathcal{C}, \phi(j, i)$. As the required number of echo messages before interpolating the final univariate polynomial at a node is equal to $\lceil \frac{n+t+1}{2} \rceil$, it is impossible for faulty nodes to make a node accept commitment $\mathcal{C}'$ different from commitment $\mathcal{C}$ suggested by the dealer. Subsequently, such an honest node P_i, after verification with verify-point, interpolates a polynomial $\overline{a}(y)$ such that $\overline{a}(y) = \phi(i, y)$. Assume an honest node receives $t+1$ ready messages before obtaining $\lceil \frac{n+t+1}{2} \rceil$ commitment $\mathcal{C}$ echo messages. Using Lemma 4.2 all these ready messages have the same commitment and with at least of one of them from an honest node, it is equal to $\mathcal{C}$. The honest node will interpolate the same $\overline{a}(y)$ as in the case of the echo messages. Using the agreement property, if a node completes the protocol Sh, then all honest nodes will eventually finish it. Let $\mathcal{S}$ be any set of $t+1$ honest nodes (P_j) that have finished the sharing. Let $s_{j,d}$ represent the share for node P_j such that $s_{j,d} = \overline{a}(0) = \phi(j, 0)$. Let $\lambda_j^{\mathcal{S}}$ be Lagrange interpolation coefficients for the set $\mathcal{S}$ and position 0. We have

$$z = \sum_{P_j \in \mathcal{S}} \lambda_j^{\mathcal{S},0} s_{j,d} = \sum_{P_j \in \mathcal{S}} \lambda_j^{\mathcal{S},0} \phi(j, 0) = s$$

and if the dealer is honest and has shared secret s during session (P_d, τ), then $z = s$.

To prove the second part, assume that two distinct honest servers P_i and P_j reconstruct values z_i and z_j by interpolating two distinct sets $S_i = \{\ell, s_\ell^{(i)}\}$ and $S_j = \{\ell, s_\ell^{(j)}\}$ of $t+1$ shares each, which are valid with respect to the unique commitment $\mathcal{C}$ using Lemma 4.2. As the shares in S_i and S_j are verified against commitment $\mathcal{C}$ and they are valid, it is easy to see that $g^{z_i} = \mathcal{C}_{00} = g^{z_j}$. As g is a generator for a prime order group, $z_i = z_j$.

Secrecy (VSS-wS). To prove the secrecy property, we use the DLog assumption. The adversary's view consists of polynomials $\phi(i, y)$ for these Byzantine nodes i, and the commitment matrix $\mathcal{C}$ and the generator g provided by the dealer. Assume that there is an adversary algorithm $\mathcal{A}$ that can compute the shared secret s given g, $\mathcal{C}$ and t degree-t univariate polynomials consistent with $\mathcal{C}$. We prove that a challenger $\mathcal{B}$ with an oracle access to such an adversary algorithm $\mathcal{A}$ can solve any DLog instance (g, g^α).

Given a DLog instance (g, g^α), the challenger $\mathcal{B}$ generates t degree-t polynomials $r_i(y) \in \mathbb{Z}_p[y]$ and associates them with non-zero indices i. It then computes $g^{r_i(0)}$ for each index i, sets $g^{r_0(0)} = g^\alpha$ and computes $\mathcal{C}_{0,k} = \mathcal{C}_{k,0}$ for $k \in [0, t]$ by interpolation. Proceeding similarly, for each $0 < \ell \leq t$, it uses $\mathcal{C}_{0,\ell}$ and $g^{r_i(\ell)}$ for each index i and computes $\mathcal{C}_{\ell,k} = \mathcal{C}_{k,\ell}$ and completes the symmetric commitment matrix $\mathcal{C}$ which is consistent with g^α as $\mathcal{C}_{0,0}$ and polynomials $r_i(y)$. $\mathcal{B}$ can then present

this matrix $\mathcal{C}$ along with polynomials r_i to the adversary algorithm $\mathcal{A}$ and return the output $s = \alpha$ as the DLog value for tuple (g, g^α). As this is not possible, except with negligible probability, we have proven that if an honest dealer has shared secret s during session (P_d, τ) and no honest node has started the Rec protocol, then the adversary cannot compute the secret s except with negligible probability.

Note that, using Pedersen commitments instead of DLog commitments, we can easily achieve and prove the VSS-S property that the adversary has no information about the shared secret s.

Efficiency. We first discuss complexities when there are no crashes. An execution without any crashes has $O(n^2)$ message complexity and $O(\kappa n^4)$ bit complexity where the size of the message is dominated by the commitment matrix $\mathcal{C}$ having $t(t+1)/2 = O(n^2)$ entries.

Using a collision-resistant hash function, Cachin *et al.* [9, Sec. 3.4] suggest a way to reduce the bit complexity to $O(\kappa n^3)$. In this approach, commitments are generated using the exponentiated form of polynomial evaluations ($A_j^{(i)} = g^{\phi(i,j)}$). Let $A^{(j)} = \langle A_0^{(j)}, A_1^{(j)}, \ldots, A_n^{(j)} \rangle$. In this case, the bit complexity gets reduced by a linear factor using the $A^{(0)}$ vector and a vector $h = \langle h_1, \ldots, h_n \rangle$, where H is collision-resistant hash function and $h_j = H(A^{(j)})$.

Now, assume there are crashes and there are subsequent recoveries. As defined earlier, $d(\kappa)$ bounds the number of possible crashes in the system. In addition, each of the Byzantine nodes may produce unlimited false help messages, out of which first $d(\kappa)$ will be answered by honest nodes. Therefore, each honest node will in total answer up to $(t+1)d(\kappa)$ help messages. The recovery mechanism requires $O(n^2)$ messages from the recovering node and $O(n)$ messages from each helper node. Therefore, the total message and bit complexity of HybridVSS are $O(tdn^2)$ and $O(\kappa tdn^3)$ respectively and we obtain a uniform polynomial bound on the bit complexity. □

5 DKG for the Hybrid Model—HybridDKG

HybridVSS requires a dealer (P_d) to select a secret and to initiate a sharing. DKG, going one step further, generates a secret in a completely distributed fashion, such that none of the system nodes knows the secret, while any $t+1$ nodes can combine their shares to determine it. Although it seems that a DKG is just a system with n nodes running their VSSs in parallel and summing all the received shares together at the end, it is not that simple in an asynchronous setting. Agreeing on $t+1$ or more VSS instances such that all of them will finish for all the honest nodes (the *agreement on a set* problem [5]), and the difficulty of differentiating between a slow node and a faulty node in the asynchronous setting are the primary sources of the added complexity.

In our hybrid system model, with no timing assumption, the node cannot wait for more than $n-t-f$ VSSs to finish. The adversary can certainly make agreeing on a subset of size $t+1$ among those nodes impossible, by appropriately delaying its messages. Cachin *et al.* [9] solve a similar agreement problem in their proactive refresh protocol using a multi-valued validated Byzantine agreement (MVBA) protocol. Known (expected) constant-round MVBA protocols [10] require threshold signature and threshold coin-tossing primitives [11]. The algorithms suggested for both of these primitives in [11] require either a dealer or a DKG. As we aim to avoid the former (dealer) in this work and the latter (DKG) is our aim itself, we cannot use their MVBA protocol.

In more technical terms, randomization in the form of distributed coin tossing or equivalent randomization functionality is mandatory for an expected constant-round Byzantine agreement such as MVBA [10]; that thwarts the attack possible with an adversary knowing the pre-defined node selection order by making completely random selections. However, an efficient algorithm for dealerless distributed coin tossing without a DKG is difficult to achieve. Canetti and Rabin [13] define a dealerless distributed coin tossing protocol without DKG; however, their protocol requires n^2

VSSs for each coin toss and is consequently inefficient. Therefore, we refrain from using randomized agreement.

We follow a much simpler approach with the same bit complexity as MVBA protocols. We use a leader-initiated reliable broadcast system with a faulty-leader change facility, inspired by Castro and Liskov's view-change protocol [14]. We choose this (optimistic phase + pessimistic phase) approach, as we expect the Byzantine failures to be infrequent in practice. The probability that the current leader of the system is not behaving correctly is small and it is not worth spending more time and bandwidth by broadcasting proposals by all the nodes during every MVBA. With this background, we now define and analyze HybridDKG.

5.1 Construction

In HybridDKG, for session τ and leader $\mathcal{L}$, each node P_d selects a secret value s_d and shares it among the group using protocol Sh of HybridVSS for session (P_d, τ). Each node finishes the HybridDKG protocol by outputting a $(\overline{\mathcal{L}}, \tau, \mathsf{DKG\text{-}completed}, \mathcal{C}, s_i)$ message, where s_i and $\mathcal{C}$ are its share and the commitment respectively and $\overline{\mathcal{L}}$ is the finally agreed upon leader.

Definition 5.1. *In session τ, protocol HybridDKG in our hybrid model having an asynchronous network of $n \geq 3t + 2f + 1$ nodes with a t-limited Byzantine adversary and f-limited crashes and network failures satisfies the following conditions:*

Liveness. *All honest finally up nodes complete protocol HybridDKG in session τ, except with negligible probability.*

Agreement. *If an honest node completes protocol HybridDKG in session τ, then, except with negligible probability, all honest finally up nodes will eventually complete protocol HybridDKG in session τ.*

Correctness. *Once an honest node completes the HybridDKG protocol for session τ, then there exists a fixed value s such that, if an honest node P_i reconstructs z_i in session τ, then $z_i = s$. This is equivalent to the weak correctness (DKG-wC) property defined in Section 3.3.*

Secrecy. *If no honest node has started the Rec protocol, then, except with negligible probability, the adversary cannot compute the shared secret s. This is equivalent to the DKG-wS secrecy property defined in Section 3.3.*

Efficiency. *The bit complexity for any instance of HybridDKG is d-uniformly bounded.*

We first describe the optimistic phase of our HybridDKG protocol. For each session τ, one among n nodes works as a leader. The leader $\mathcal{L}$, once it finishes the VSS proposal by $t + 1$ nodes with $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}_d, s_{i,d})$, broadcasts the $n - t - f$ ready messages (set $\widehat{\mathcal{R}}$) it received for each of those $t+1$ finished VSSs (set $\widehat{\mathcal{Q}}$). Nodes include signatures with ready messages to enable the leader to provide a validity proof for its proposal. In this *extended HybridVSS* protocol, shared messages look like $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}_d, s_{i,d}, \mathcal{R}_d)$, where a set $\mathcal{R}_d$ includes $n-t-f$ signed ready messages for session (P_d, τ). Once this broadcast completes, each node knows $t + 1$ VSS instances to wait for. Once a node P_i finishes those, it sums the shares $s_{i,d}$ to obtain its final share s_i.

If the leader is faulty or slow and does not proceed with the protocol or sends arbitrary messages, the protocol enters into a pessimistic phase. Here, other nodes use a *leader-change* mechanism to change their leader with a pre-defined cyclic permutation and provide liveness without damaging system safety. Without loss of generality, we assume that the permutation is a linear sorted order of node indices. Every unsatisfied node sends a signed leader-change (lead-ch) request to all the

Optimistic phase for node P_i: session (τ) and leader $\mathcal{L}$

upon initialization:

$e_{\mathcal{L},\mathcal{Q}} \leftarrow 0$ and $r_{\mathcal{L},\mathcal{Q}} \leftarrow 0$ **for** every $\mathcal{Q}$; $\overline{\mathcal{Q}} \leftarrow \emptyset$; $\widehat{\mathcal{Q}} \leftarrow \emptyset$
$\overline{\mathcal{M}} \leftarrow \widehat{\mathcal{R}} \leftarrow n - t - f$ signed lead-ch messages for leader $\mathcal{L}$
$cnt \leftarrow 0$; $cnt_\ell \leftarrow 0$ **for all** $\ell \in [1, n]$;
$lc_{\mathcal{L}} \leftarrow 0$ for each leader $\mathcal{L}$; $lc_{flag} \leftarrow$ **false**
$\mathcal{L}_{next} \leftarrow \mathcal{L} + n - 1$
for all $d \in [1, n]$ **do**
  initialize extended-HybridVSS Sh protocol (P_d, τ)

upon $(P_d, \tau, \mathsf{out}, \mathsf{shared}, \mathcal{C}_d, s_{i,d}, \mathcal{R}_d)$ (first time):

$\widehat{\mathcal{Q}} \leftarrow \{P_d\}$; $\widehat{\mathcal{R}} \leftarrow \{\mathcal{R}_d\}$
if $|\widehat{\mathcal{Q}}| = t + 1$ **and** $\overline{\mathcal{Q}} = \emptyset$ **then**
  if $P_i = \mathcal{L}$ **then**
    send the message $(\mathcal{L}, \tau, \mathsf{send}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to **each** P_j
  else
    $delay \leftarrow delay(T)$; **start timer**$(delay)$

upon a message $(\mathcal{L}, \tau, \mathsf{send}, \mathcal{Q}, \mathcal{R}/\mathcal{M})$ from $\mathcal{L}$ (first time):

if verify-signature$(\mathcal{Q}, \mathcal{R}/\mathcal{M})$ **and** $(\overline{\mathcal{Q}} = \emptyset$ **or** $\overline{\mathcal{Q}} = \mathcal{Q})$ **then**
  send the message $(\mathcal{L}, \tau, \mathsf{echo}, \mathcal{Q})_{\mathsf{sign}}$ to **each** P_j

upon a message $(\mathcal{L}, \tau, \mathsf{echo}, \mathcal{Q})_{\mathsf{sign}}$ from P_m (first time):

$e_{\mathcal{L},\mathcal{Q}} \leftarrow e_{\mathcal{L},\mathcal{Q}} + 1$
if $e_{\mathcal{L},\mathcal{Q}} = \lceil \frac{n+t+1}{2} \rceil$ **and** $r_{\mathcal{L},\mathcal{Q}} < t + 1$ **then**
  $\overline{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\overline{\mathcal{M}} \leftarrow \lceil \frac{n+t+1}{2} \rceil$ signed echo messages for $\mathcal{Q}$
  send the message $(\mathcal{L}, \tau, \mathsf{ready}, \mathcal{Q})_{\mathsf{sign}}$ to **each** P_j

upon a message $(\mathcal{L}, \tau, \mathsf{ready}, \mathcal{Q})_{\mathsf{sign}}$ from P_m(first time):

$r_{\mathcal{L},\mathcal{Q}} \leftarrow r_{\mathcal{L},\mathcal{Q}} + 1$
if $r_{\mathcal{L},\mathcal{Q}} = t + 1$ **and** $e_{\mathcal{L},\mathcal{Q}} < \lceil \frac{n+t+1}{2} \rceil$ **then**
  $\overline{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\overline{\mathcal{M}} \leftarrow t + 1$ signed ready messages for $\mathcal{Q}$
  send the message $(\mathcal{L}, \tau, \mathsf{ready}, \mathcal{Q})_{\mathsf{sign}}$ to **each** P_j
else if $r_{\mathcal{L},\mathcal{Q}} = n - t - f$ **then**
  stop timer, if any
  wait for shared output-messages for each $P_d \in \mathcal{Q}$
  $s_i \leftarrow \sum_{P_d \in \mathcal{Q}} s_{i,d}$; $\forall_{p,q} : \mathcal{C}_{p,q} \leftarrow \prod_{P_d \in \mathcal{Q}} (\mathcal{C}_d)_{p,q}$
  output $(\mathcal{L}, \tau, \mathsf{DKG\text{-}completed}, \mathcal{C}, s_i)$

upon timeout:

if $lc_{flag} =$ **false then**
  if $\overline{\mathcal{Q}} = \emptyset$ **then**
    $lc_{flag} \leftarrow$ **true**; send msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L} + 1, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})_{\mathsf{sign}}$ to **each** P_j
  else
    $lc_{flag} \leftarrow$ **true**; send msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L} + 1, \overline{\mathcal{Q}}, \overline{\mathcal{M}})_{\mathsf{sign}}$ to **each** P_j

upon $(\mathcal{L}, \tau, \mathsf{in}, \mathsf{recover})$:

send the message $(\mathcal{L}, \tau, \mathsf{help})$ to all the nodes;
send all messages in $B_{\mathcal{L},\tau}$

upon a message $(\mathcal{L}, \tau, \mathsf{help})$ from P_ℓ:

if $cnt_\ell \leq d(\kappa)$ **and** $cnt \leq (t + 1)d(\kappa)$ **then**
  $cnt_\ell \leftarrow cnt_\ell + 1$; $cnt \leftarrow cnt + 1$
  send all messages of $B_{\ell(\mathcal{L},\tau)}$

Figure 2: HybridDKG protocol (Optimistic phase)

Leader-change for node P_i: session (τ) and leader $\mathcal{L}$
upon a msg $(\tau, \mathsf{lead\text{-}ch}, \overline{\mathcal{L}}, \mathcal{Q}, \mathcal{R}/\mathcal{M})_{\mathsf{sign}}$ from P_j(first time):
  if $\overline{\mathcal{L}} > \mathcal{L}$ **and** verify-signature$(\mathcal{Q}, \mathcal{R}/\mathcal{M})$ **then**
    $lc_{\overline{\mathcal{L}}} \leftarrow lc_{\overline{\mathcal{L}}} + 1$
    $\mathcal{L}_{next} \leftarrow \min(\mathcal{L}_{next}, \overline{\mathcal{L}})$
    if $\mathcal{R}/\mathcal{M} = \mathcal{R}$ **then**
      $\widehat{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\widehat{\mathcal{R}} \leftarrow \mathcal{R}$
    else
      $\overline{\mathcal{Q}} \leftarrow \mathcal{Q}$; $\overline{\mathcal{M}} \leftarrow \mathcal{M}$
    if $(\sum lc_{\mathcal{L}} = t + f + 1$ **and** $lc_{flag} =$ **false**$)$ **then**
      if $\overline{\mathcal{Q}} = \emptyset$ **then**
        send the msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}_{next}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to **each** P_j
      else
        send the msg $(\tau, \mathsf{lead\text{-}ch}, \mathcal{L}_{next}, \overline{\mathcal{Q}}, \overline{\mathcal{M}})$ to **each** P_j
    else if $(lc_{\overline{\mathcal{L}}} = n - t - f)$ **then**
      $\overline{\mathcal{M}} \leftarrow \widehat{\mathcal{R}} \leftarrow n - t - f$ signed lead-ch messages for $\overline{\mathcal{L}}$
      $\mathcal{L} \leftarrow \overline{\mathcal{L}}$; $\mathcal{L}_{next} \leftarrow \mathcal{L} - 1$
      $lc_{\mathcal{L}} \leftarrow 0$; $lc_{flag} =$ **false**
      if $P_i = \mathcal{L}$ **then**
        if $\overline{\mathcal{Q}} = \emptyset$ **then**
          send the message $(\mathcal{L}, \tau, \mathsf{send}, \widehat{\mathcal{Q}}, \widehat{\mathcal{R}})$ to **each** P_j
        else
          send the message $(\mathcal{L}, \tau, \mathsf{send}, \overline{\mathcal{Q}}, \overline{\mathcal{M}})$ to **each** P_j
      else
        $delay \leftarrow delay(T)$
        start timer$(delay)$

Figure 3: HybridDKG protocol (Pessimistic phase)

nodes for the next leader $\mathcal{L} + 1$ if it receives an invalid message from the existing leader $\mathcal{L}$ or if its timer timed out. Timeouts are based on the function $delay(T)$ described in Section 2.1. When a node collects $t + 1$ lead-ch messages for leaders $\mathcal{L} + \delta$ for small positive integers δ, it is confirmed that at least one honest node is unsatisfied and it sends a lead-ch message to all the nodes for the smallest leader among those requested, if it has not done that yet. Once a node receives $n - t - f$ lead-ch requests for a leader $\overline{\mathcal{L}} > \mathcal{L}$, it accepts $\overline{\mathcal{L}}$ as the new leader and enters into the optimistic phase.

The new leader enters into the optimistic phase by sending a send message for set $\overline{\mathcal{Q}}$ if it is non-empty or else for set $\widehat{\mathcal{Q}}$. Set $\widehat{\mathcal{Q}}$ contains the indices of nodes whose VSS instances have completed at one more nodes, while set $\overline{\mathcal{Q}}$ is a set of nodes broadcast by the current or a previous leader such that an honest node might have completed that broadcast. In case of leader change, set $\overline{\mathcal{Q}}$ avoids two honest nodes finishing with two different VSSs sets with two different leaders. Once a node receives $\lceil \frac{n+t+1}{2} \rceil$ echo messages or $t + 1$ ready messages, $\widehat{\mathcal{Q}}$ is assigned to $\overline{\mathcal{Q}}$ as some or all honest nodes might complete the broadcast even if others time out. $\overline{\mathcal{Q}}$ ensures that the new leader broadcasts the same set $\widehat{\mathcal{Q}}$ and all honest nodes delivers the same set in the agreement. Set $\overline{\mathcal{M}}$ attached to $\overline{\mathcal{Q}}$ contains $\lceil \frac{n+t+1}{2} \rceil$ signed echo messages or $t + 1$ signed ready messages for $\overline{\mathcal{Q}}$. Along with condition ($|\widehat{\mathcal{Q}}| = t + 1$ and $\overline{\mathcal{Q}} = \emptyset$), set $\overline{\mathcal{M}}$ avoids wrong broadcast sets from the dishonest nodes. While sending its proposal, $\mathcal{L}$ also includes lead-ch signatures received from $n - t - f$ nodes to prove its validity to the nodes who have not received enough lead-ch messages. As in HybridVSS, the set $\mathcal{B}$ contains all outgoing messages at a node along with their intended recipients and $\mathcal{B}_\ell$ represents the subset of messages destined for node P_ℓ. Counters cnt and cnt_ℓ keep track of the

numbers of overall help requests and help requests sent by each node P_ℓ respectively. Figures 2 and 3 present the optimistic and the pessimistic phases of the HybridDKG protocol. Protocol Rec for HybridDKG remains exactly the same as HybridVSS Rec protocol from Fig. 1.

5.2 Analysis

The main theorem for our HybridDKG is as follows.

Theorem 5.1. *With the DLog assumption, protocol HybridDKG provides an asynchronous distributed key generation mechanism in the hybrid model for $n \geq 3t + 2f + 1$.*

Proof. We need to show the liveness, agreement, correctness, secrecy, and efficiency of our HybridDKG protocol.

Liveness. In HybridVSS, if the dealer is honest and finally up, then all honest finally up nodes complete the sharing initiated by it. With $n - t - f$ honest finally up nodes in the system, each honest finally up node will eventually complete $t + 1$ HybridVSS sharings, as required. Each such node will start a timer upon completing these $t+1$ HybridVSS instances. If the leader is honest and uncrashed, it also completes $t + 1$ HybridVSS instances, and broadcasts its proposal; based on the liveness property of reliable broadcast [4], each honest finally up node delivers the same verifiable proposal. Honest finally up nodes stop their timers when they complete this reliable broadcast. To finish, according to the HybridVSS agreement properties, all honest finally up nodes complete protocol Sh for nodes in this proposal.

If the leader is compromised, crashed or does not finish its broadcast before a timeout at an honest node, then a signed lead-ch request is broadcast by that honest node (pessimistic phase). After receiving $n-t-f$ lead-ch requests, the new leader takes over, each honest node starts a timer for the proposal from the new leader, and the protocol reenters the optimistic phase. As the number of crashes is polynomially bounded and the network eventually gets repaired resulting in message delays becoming eventually bounded by $delay(T)$, an honest finally up leader will eventually reliably broadcast a proposal and protocol HybridDKG will complete.

The requirement of $n - t - f$ lead-ch requests for a leader replacement makes sure that nodes do not complete the leader change too soon. An honest node sends a signed lead-ch message for the smallest leader (among the received set) if it receives $t + f + 1$ lead-ch messages, even if it has not observed any fault, as this indicates that at least one honest node has observed some fault and the node does not want to start the leader change too late.

Agreement. An honest node completes a HybridDKG execution when it completes a reliable broadcast of the current leader's sharing proposal, finishes HybridVSS sharing by $t + 1$ nodes in that proposal, and computes its final share as a summation of the shares obtained from these $t + 1$ sharings. According to the agreement property of the reliable broadcast, if one honest node completes the protocol, then all honest finally up nodes will eventually complete the protocol. Further, when only some (but not all) nodes complete the reliable broadcast before a leader change, sets $\overline{\mathcal{Q}}$ and $\overline{\mathcal{M}}$ ensure that all nodes complete a reliable broadcast for the same $\overline{\mathcal{Q}}$ after the leader change. According to the agreement property of the HybridVSS, once an honest node completes a set of $t + 1$ sharings, then all honest finally up nodes will eventually complete all of these $t + 1$ sharings. Consequently, once an honest node completes HybridDKG then all honest finally up nodes will eventually complete the HybridDKG protocol.

Correctness (DKG-wC). According to the above discussed agreement property, once an honest node completes the HybridDKG for session (τ), then all $(n - t - f)$ honest finally up nodes will eventually complete the HybridDKG protocol. According to the correctness property of the reliable broadcast protocol, each of these nodes will decide the same set of $t + 1$ sharings. Further, when

only some (but not all) nodes complete the reliable broadcast before a leader change, sets $\overline{\mathcal{Q}}$ and $\overline{\mathcal{M}}$ make sure that all nodes complete a reliable broadcast for the same $\overline{\mathcal{Q}}$ after the leader change. For each of the completed sharings, if run individually, each node P_i will reconstruct the same shared secret $z_{i,d}$ where P_d is the dealer for the sharing. Let $z_i = \sum_{P_d \in \mathcal{Q}} z_{i,d}$. As nodes add their shares for the completed $t+1$ sharings as the HybridDKG execution finishes and as Lagrange-interpolation is homomorphic over addition, on reconstruction after the HybridDKG protocol each node will output the same $z_i = s$.

Secrecy (DKG-wS). In HybridDKG sharings by $t+1$ nodes are used, where at least one of those shared secrets was proposed by an honest party. In a reliable broadcast, two honest nodes always finish the protocol with the same message; therefore, the same $t+1$ sharings are completed by all the honest nodes. For a HybridVSS execution, if the dealer P_d is honest then until the reconstruction protocol starts, the adversary cannot compute the shared secret s_d. Therefore, at the end of HybridDKG protocol, the adversary does not know at least one of the $t+1$ shared secrets. As the system's secret $s = \sum_{P_d \in \mathcal{Q}} s_d$, the adversary cannot compute the shared secret s.

Efficiency. The message and bit complexities of HybridVSS are $O(tdn^2)$ and $O(\kappa tdn^3)$ respectively. In the HybridDKG protocol with the asynchronous communication assumption, the system may complete all n VSS executions, even though the required execution count is just $t+1$; thus, the message and bit complexities of the possible n HybridVSS Sh protocols in HybridDKG are $O(tdn^3)$ and $O(\kappa tdn^4)$ respectively. If the HybridDKG protocol completes without entering into the pessimistic phase, then the system only needs one reliable broadcast of message of size $O(\kappa n)$, message complexity $O(tdn^2)$ and bit complexity $O(\kappa tdn^3)$. As a result, the optimal message and bit complexities for the HybridDKG protocol are $O(tdn^3)$ and $O(\kappa tdn^4)$ respectively.

In the pessimistic case, the total number of leader changes is bounded by $O(d)$. Each leader change operation involves $O(tdn^2)$ messages and $O(\kappa tdn^3)$ communication bits. For each faulty leader, $O(tdn^2)$ messages and $O(\kappa tdn^3)$ bits are communicated during its administration. Therefore, in the worst case, $O(td^2n^2)$ messages and $O(\kappa td^2n^3)$ bits are communicated before the HybridDKG completes and worst case message and bit complexities of the HybridDKG protocol are $O(tdn^2(n+d))$ and $O(\kappa tdn^3(n+d))$ respectively. Note that considering just a t-limited Byzantine adversary (and not also crashes and link failures), the above complexities become $O(n^3)$ and $O(\kappa n^4)$ respectively. These are same as the complexities of the share refresh protocol for AVSS [9]. □

5.3 Uniform Randomness of the Shared Secret

The shared secret in the above HybridDKG may not be *uniformly* random; this is a direct effect of using only the discrete logarithm commitments (see [24, Section 3] for details). In many cases, we do not need a uniformly random secret key; the security of these cases relies on the assumption that the adversary cannot compute the secret. However, a uniformly random shared secret may be required in some protocols. In that case, we use Pedersen commitments, but we do not employ the methodology defined by Gennaro *et al.* [24], which increases the latency in the system. We observe instead that with the random oracle assumption, the communicationally demanding technique by Gennaro *et al.* can be replaced with the much simpler NIZK proofs described in Equation 1 in Section 3.2.

We denote HybridDKG schemes using DLog commitments and Pedersen commitments as $\mathsf{HybridDKG}_{\mathrm{DLog}}$ and $\mathsf{HybridDKG}_{\mathrm{Ped}}$ respectively. For node P_i, the corresponding HybridDKG-Sh and HybridDKG-Rec

schemes are defined as follows.

$$\begin{aligned}
\left(\mathcal{C}^{(s,s')}_{\langle g,h\rangle}, [\mathcal{C}^{(s)}_{\langle g\rangle}, \text{NIZKPK}_{\equiv Com}], s_i, s'_i\right) &= \mathsf{HybridDKG\text{-}Sh}_{\text{Ped}}(n,t,f,t',g,h,\alpha_i,\alpha'_i) \\
\left(\mathcal{C}^{(s)}_{\langle g\rangle}, s_i\right) &= \mathsf{HybridDKG\text{-}Sh}_{\text{DLog}}(n,t,f,\tilde{t},g,\alpha_i) \\
s &= \mathsf{HybridDKG\text{-}Rec}_{\text{Ped}}(t, \mathcal{C}^{(s,s')}_{\langle g,h\rangle}, s_i, s'_i) \\
s &= \mathsf{HybridDKG\text{-}Rec}_{\text{DLog}}(t, \mathcal{C}^{(s)}_{\langle g\rangle}, s_i)
\end{aligned}$$

Here, t' is the number of VSS instances to be chosen ($t < t' \leq 2t+1$), $\alpha_i, \alpha'_i \in \mathbb{Z}_p$ are respectively a secret and randomness shared by P_i, and $\mathcal{C}^{(s)}_{\langle g\rangle} = [g^s, g^{\phi(1)}, \cdots, g^{\phi(n)}]$ and $\mathcal{C}^{(s,s')}_{\langle g,h\rangle} = [g^s h^{s'}, g^{\phi(1)}h^{\phi'(1)}, \cdots, g^{\phi(n)}h^{\phi'(n)}]$ are respectively the discrete logarithm and Pedersen commitment vectors for $\phi, \phi' \in \mathbb{Z}_p[x]$ of degree t with $\phi(0) = s$ and $\phi'(0) = s'$. The optional $\text{NIZKPK}_{\equiv Com}$ is a vector of zero-knowledge proofs of knowledge that the corresponding entries of $\mathcal{C}^{(s)}_{\langle g\rangle}$ and $\mathcal{C}^{(s,s')}_{\langle g,h\rangle}$ commit to the same values. This vector is useful in applications such as distributed PKG for IBC [29], where Pedersen commitments have to be replaced by DLog commitments.

To achieve uniform randomness of the secret, each node adds a DLog commitment of its share and the corresponding $\text{NIZKPK}_{\equiv Com}$ in its DKG-completed message at the end of the $\mathsf{HybridDKG}_{\text{Ped}}$ protocol, and sends this DKG-completed message without the share to every other node. $\mathsf{HybridDKG}_{\text{Ped}}$ achieves the same liveness and agreement guarantees as those of HybridDKG, while for correctness and secrecy it respectively achieves the DKG-C and DKG-S properties instead of their weaker versions in $\mathsf{HybridDKG}_{\text{DLog}}$. Maintaining the system-oriented flow of the article, we postpone the cryptographic analysis of $\mathsf{HybridDKG}_{\text{Ped}}$ with uniform randomness of the shared secret to Appendix A.

6 DKG Implementation over PlanetLab

We have implemented our HybridDKG protocol from Section 5 and analyzed its performance over the PlanetLab platform [43]. In this section, we discuss our implementation and experiments along with other system aspects of DKG. We observed HybridDKG to be practical for use over the Internet, which is also further illustrated by applications such as [29, 48].

We first briefly describe our design and implementation of HybridDKG. We then analyze the results from our experiments over PlanetLab and discuss resilience against denial-of-service (DoS) attacks and Sybil attacks. Finally, we propose some system-level optimizations for the HybridDKG protocol based on our analysis. These optimizations improve the performance of the protocols, without hampering its liveness or safety, when a leader behaves honestly and delays in the system remain within reasonable limits.

6.1 Software Design and Implementation

We design our DKG nodes as state machines (using the state machine replication approach [33,44]), where nodes move from one state to another based on the messages received. These messages are categorized into three types: operator messages, network messages and timer messages. The operator messages define interactions between nodes and their operators and are of two types: in and out. In an in message, an operator provides some input to the protocol, while an out message presents the protocol results to the operator. The network messages realize the protocol flow between nodes. Almost all messages in the HybridVSS and HybridDKG pseudocode in Figures 1, 2 and 3 are of this type. Finally, the timer messages implement the weak synchrony assumption

described in Section 2.1 in the form of start timer, stop timer and timeout. Our node will be in one of the following seven states:

> leader_unconfirmed, under_recovery, functional, agreement_started, agreement_completed, leader_change_started, and dkg_completed.

leader_unconfirmed is a starting phase, which indicates that a node does not have a sufficient number of lead-ch signatures to confirm a new leader. We also differentiate between agreement_completed and dkg_completed as a node may complete the broadcast by the leader before it completes the associated VSS instances. The rest of the states (under_recovery, functional, agreement_started, dkg_completed, leader_change_started) have their apparent meanings.

A primary application for our protocols is to construct distributed PKGs [29, 48]. Therefore, we consider a HybridDKG implementation over pairing-friendly elliptic curves. We develop our object-oriented C++ implementation over the PBC library [35] for the underlying elliptic-curve and finite-field operations, and a PKI infrastructure with the DSA signatures based on GnuTLS [36] for confidentiality and message authentication. However, our DKG code is generic and can easily be modified to work with any other C/C++ number theoretic library. When using a different library, a C++ interface layer will have to be developed over that library, which will provide a cyclic group interface in the form of a C++ class as required for our polynomials, commitments and shares.

Our implementation [2] replicates our HybridVSS and HybridDKG pseudocode and therefore has an event- (or message-) driven architecture. The similarity between the code and the pseudocode is intentional; it helped identify several errors in the code and omissions in the pseudocode. DKG nodes are multithreaded and the code is structured as a set of event handlers. This set contains a handler for each operator and network message, and a handler for each timer. Each handler corresponds to an input action and there are also methods that correspond to the internal actions in the system. The event handling loop works as follows: nodes wait in a select call for a network message to arrive, for an operator instruction or for a timer deadline to be reached and then they call the appropriate handler. The handler performs computations similar to the corresponding action in the pseudocode and then it invokes any methods corresponding to internal actions whose preconditions have become true. In most of the cases, it results in sending a message to an operator or over the network. Each message has a 3-byte generic header, which contains a tag that identifies the message type (1 byte) and the total size of the rest of the message (2 bytes). The structure of the message bodies varies from message type to message type.

6.2 Performance Analysis

Experimental Setup and Testing. In order to examine its realistic performance, we test our DKG implementation on the PlanetLab platform.

A typical PlanetLab machine configuration is 4×2.4 GHz cores with 4 GB RAM and 500 GB hard disk [1]. In terms of network capacity, the average bandwidth between PlanetLab nodes is 64 Mbps [34]. For our experiments, we chose the required number of PlanetLab nodes randomly from nodes having near-average configuration and bandwidth, and a reasonable liveness history. In terms of the geographical distribution, although our selections were biased towards nodes in Europe and America, we had a significant number (around 20 percent) of nodes chosen from the other continents. Taking advantage of the uniform operating system distribution and configuration over all PlanetLab nodes, we compiled and statically linked our code on a single node and replicated the executable over the rest. Note that we did not consider the loads of machines while selecting our nodes; those loads were unpredictable and varied a lot during our experiments. In order to

Table 1: Median values of DKG completion time and CPU time per node for various system sizes.

n	t	f	**Time** (seconds)	**CPU Seconds/Node**
10	1	3	3.43 ± 0.91	0.64 ± 0.08
20	2	6	6.71 ± 1.18	3.11 ± 0.77
30	3	10	17.16 ± 2.25	9.62 ± 1.03
40	4	13	40.98 ± 1.65	25.67 ± 0.74
50	5	17	90.69 ± 13.87	53.14 ± 5.77
60	6	20	188.32 ± 31.09	85.15 ± 4.66
70	7	24	325.31 ± 45.67	173.85 ± 28.85

determine an average performance, we ran the experiments at least ten times for each parameter set.

We test the DKG implementation for systems of up to 70 nodes and present median completion times and median CPU usage in Table 1 along with 95% two-sided confidence intervals. Our experiments are terminating and conducted via the method of independent replications. A single replication consists of $n - t - f$ or more individual observations each corresponding to the time required for a participating peer in the quorum to complete the DKG protocol; there are 10 replications for each n value. Using independent replications, an unbiased sample point estimator for variance is calculated and used to obtain our two-sided confidence intervals using the t-distribution. Our DKG experiments are set up so that correctness is guaranteed so long as at most 30% of the peers may crash and 10% of the peers may be Byzantine. While we can tolerate any fraction of Byzantine peers less than 1/3, we use these numbers since in many practical scenarios we expect the fraction of Byzantine faults to be less than 10% and modest compared to the fraction of crash failures.

During our tests, we studied the following aspects: possible sizes of the system, the average completion time of the protocol and the applicability of the weak synchrony assumption from Section 2.1 that we make for HybridDKG. Our HybridDKG protocol handles Byzantine attacks and the performance of the implementation against these malicious attacks should have been verified during the testing. However, in our HybridDKG analyses in Section 5.2, we observe that a t-limited Byzantine adversary cannot launch any attack other than delaying the network messages; this is a direct effect of working in the computational security setting. The verification mechanism in our HybridDKG and HybridVSS protocols can easily catch any modification to network messages, commitments or shares and messages that fail verification are dropped by the honest nodes without any further processing. As a result, we do not include any special Byzantine adversary code in our tests. Note that we still need to follow the security threshold t for the shared polynomials as otherwise the adversary can break the secrecy of the protocol.

Evaluation. We make the following observations based on our experiments over the PlanetLab platform:

- We test the performance of our DKG implementation for systems of up to 70 nodes and measure the average HybridDKG completion time (see Table 1). We observe an approximately cubic growth in the average completion time. Even with the cubic complexity, we observe that DKGs for reasonably large distributed systems ($n \leq 70$) are practical for the Internet.

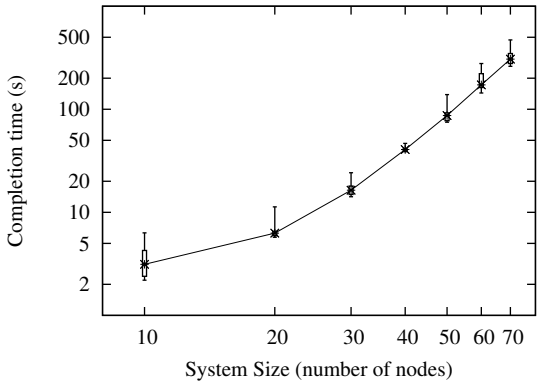

Figure 4: Quartile plot of DKG Completion Times for system sizes ranging from 10 to 70 nodes. Note the log-log scale.

- In Fig. 4, we compare minimum and maximum completion times for a set of experiments. The large gaps between those values demonstrate the robustness of the DKG system against the Internet's asynchronous nature and varied resource levels of the PlanetLab nodes that we chose.

- To check the applicability of the weak synchrony assumption [14] that we use in HybridDKG, we also tested the system with multiple crashed leaders. In such scenarios, the DKG protocol successfully completed after a few leader changes. However, we observe that the average completion time of a system critically varies with the choice of $delay(T)$ function. For $delay(T) = T$, the leader change takes significantly more time than that required to reliably broadcast the chosen instances. As a result, when a leader is crashed, most of the other nodes end up waiting for long periods even though the next possible leader is available. Further, this waiting period grows significantly in the case of multiple crashes.

  We observed that $delay(T) = T/\delta$ for $\delta = 2$ or 3 is a better choice in terms of a compromise between allowing an honest leader enough time to complete its broadcast and reducing the waiting period between leader changes. However, an appropriate $delay(T)$ function may change as the system parameters vary and we suggest that $delay(T)$ should only be finalized for a system after some rigorous testing.

- Further, we compared the (CPU) execution time for nodes against the time they spend on network transmission or waiting for other nodes. We observe that the the protocol execution time per node is significantly smaller than their completion periods (Table 1). This proves that the completion time periods are larger not because of the required computation; they are high rather due to network delays and will drop significantly if a more reliable network with better bandwidth (*e.g.*, an internal network in an organization or a cloud computing environment) is provided.

DoS and Sybil Attacks. The distributed nature of HybridDKG provides an inherent protection against DoS attacks, and the inclusion of the network-failure and crashed-node assumptions makes DoS attacks less feasible. Although leaders might become primary targets, we mitigate this issue with an efficient leader-changing mechanism. Further, as all valid communication is done over TLS links, nodes can easily reject messages arriving from non-system entities. Sybil attacks [17] are not a major concern, as ad-hoc additions of nodes is not a feature of our system. Nodes are added using

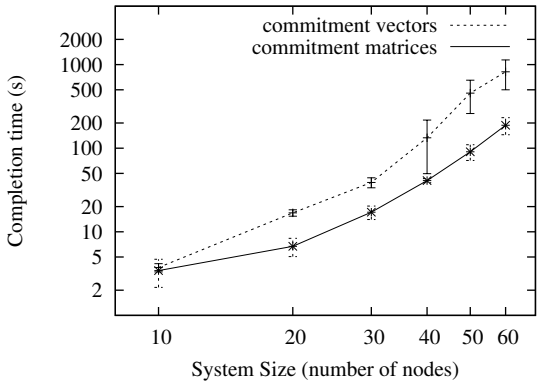

Figure 5: Completion time vs system size (log-log plot) for commitment matrices and commitment vectors

the group modification agreement protocol [28, Sec. 6], which involves administrative interaction at each node.

6.3 System-Level Optimizations

The completion time values of our HybridDKG implementation that we observe in Fig. 4 are practical for applications such as PB-OR [30, 31], where DKG phase sizes are in days. However, these values may not be ideal for many other practical systems such as our robust communication protocols for DHTs [48, 49]. During our experiments we observed that some system-level optimizations can significantly reduce the completion time values and make HybridDKG practical enough for these applications. We next discuss these optimizations. Note that these optimizations make HybridDKG more practical in the normal course of operation, when a leader behaves honestly and its messages flow without any significant delay. They may not be the best options in the worst-case scenarios having multiple leader-change operations. However, they never hamper the safety or liveness properties of the protocol.

shared Messages. With the PKI infrastructure in place, digital signatures are readily available in our system. Our HybridVSS scheme does not make use of these signatures. In the HybridDKG protocol, it is important for the leader to be one of first nodes to complete any HybridVSS instance. That helps the leader to send its DKG send message before the timer timeouts at fast nodes.

To help the leader, we add a new shared network message that a node having $2t + f + 1$ signed VSS ready messages for a completed HybridVSS instance sends to a leader. The leader can then include this HybridVSS instance in its DKG send without completion of the VSS instance at its own machine. Note that $2t + f + 1$ signed ready messages confirms that the corresponding HybridVSS instance will complete at all honest finally up nodes.

Commitment Matrices versus Commitment Vectors.
In theory, linear-size commitment vectors which use collision-resistant hash functions, as introduced in [9] for AVSS, provide linear speedup over quadratic-size commitment matrices. However, measuring precisely, the commitment size is $(t+1)^2$ for matrices and (approximately) $2(3t + 2f + 1)$ for vectors. Even if $f = 0$, the required elements and computations for the

matrix commitments are less than those for the vector commitments for $t < 6$. For $f > 0$, this upper limit will only increase.

Our experiments confirm this computation. We observed that the commitment matrices, although asymptotically inefficient, are more efficient in systems of the sizes we measured (see Fig. 5). We suggest the usage of commitment matrices instead of commitment vectors for systems of reasonable size.

Running only $2t + f + 1$ VSS Instances. We observed that the VSS instances are more resource consuming than the agreement required at the end. Generally, we only need $t+1$ VSS instances to succeed. Assuming $t + f$ VSS instances might fail during a DKG, it is sufficient to start HybridVSS instances at just $2t + f + 1$ nodes instead of at all n nodes. Nodes that do not start a VSS initially may utilize the weak synchrony assumption to determine when to start a VSS instance if required.

7 Conclusion and Future Work

While working towards a realistic DKG architecture, we first investigated the differences between the partially synchronous and asynchronous communication models and observed that only the asynchronous communication model realistically fits the existing Internet. We also incorporated crash-recoveries and network failures in the system model along with the traditional Byzantine adversary.

We defined a VSS scheme (HybridVSS) that works in our hybrid communication model. We then observed the requirement of a Byzantine agreement while implementing DKG in the asynchronous communication setting and presented a leader-based system to achieve that in our HybridDKG protocol. We also implemented our DKG protocol and tested its practicality and efficiency over the PlanetLab platform. Our results are certainly encouraging toward the use of the HybridDKG protocol for multiparty computation over the Internet.

In future, we plan to make our HybridDKG protocol secure against adaptive adversaries by introducing the rewinding adversary similar to the one in [12]. Further, we use the random oracle assumption to achieve uniform randomness of the shared secret in HybridDKG. It would be interesting to obtain uniform randomness without random oracles using some distributed commitment techniques or the common reference string model.

References

[1] PlanetLab Hardware Requirements. `http://www.planet-lab.org/hardware`, 2007. Accessed July 2011.

[2] DKG Implementation Webpage. `http://crysp.uwaterloo.ca/software/DKG`, 2012.

[3] I. Abraham, D. Dolev, and J. Y. Halpern. An Almost-surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience. In *PODC'08*, pages 405–414, 2008.

[4] M. Backes and C. Cachin. Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries. In *DSN'03*, pages 37–46, 2003.

[5] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In *STOC'93*, pages 52–61, 1993.

[6] G. R. Blakley. Safeguarding Cryptographic Keys. In *the National Computer Conference*, pages 313–317, 1979.

[7] D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO'01*, pages 213–229, 2001.

[8] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *PODC'84*, pages 154–162, 1984.

[9] C. Cachin, K. Kursawe, A.Lysyanskaya, and R. Strobl. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In *CCS'02*, pages 88–97, 2002.

[10] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and Efficient Asynchronous Broadcast Protocols. In *CRYPTO'01*, pages 524–541, 2001.

[11] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography. In *PODC'00*, pages 123–132, 2000.

[12] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive Security for Threshold Cryptosystems. In *CRYPTO'99*, pages 98–115, 1999.

[13] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC'93*, pages 42–51, 1993.

[14] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[15] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *FOCS'85*, pages 383–395, 1985.

[16] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Advances in Cryptology—CRYPTO'89*, pages 307–315, 1989.

[17] J. R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 251–260, 2002.

[18] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the Presence of Partial Synchrony. *Journal of ACM*, 35(2):288–323, 1988.

[19] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *FOCS'87*, pages 427–437, 1987.

[20] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86*, pages 186–194, 1986.

[21] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32(2):374–382, 1985.

[22] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT'99*, pages 295–310, 1999.

[23] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Applications of Pedersen's Distributed Key Generation Protocol. In *CT-RSA*, pages 373–390, 2003.

[24] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[25] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[26] Shafi Goldwasser. Multi-Party Computations: Past and Present. In *ACM PODC'97*, pages 1–6, 1997.

[27] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *IEEE Global Communication Conference (GLOBALCOM'87)*, pages 99–102, 1987.

[28] A. Kate and I. Goldberg. Distributed Key Generation for the Internet. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 119–128, 2009.

[29] A. Kate and I. Goldberg. Distributed Private-Key Generators for Identity-Based Cryptography. In *7th Conference on Security and Cryptography for Networks (SCN 2010)*, pages 436–453, September 2010.

[30] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing. In *7th Privacy Enhancing Technologies Symposium (PET)*, pages 95–112, 2007.

[31] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing with Improved Forward Secrecy. *ACM Trans. Inf. Syst. Secur.*, 13(4):29, 2010.

[32] A. Khalili, J. Katz, and W. Arbaugh. Toward Secure Key Distribution in Truly Ad-Hoc Networks. In *IEEE Workshop on Security and Assurance in Ad-Hoc Networks 2003*, pages 342–346, 2003.

[33] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[34] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca. Measuring bandwidth between planetlab nodes. In *PAM'05*, pages 292–305, 2005.

[35] B. Lynn. PBC Library. `http://crypto.stanford.edu/pbc/`, 2009. Accessed April 2009.

[36] N. Mavroyanopoulos, F. Fiorina, T. Schulz, A. McDonald, and S. Josefsson. The GNU Transport Layer Security Library. `http://www.gnu.org/software/gnutls/`, 2009. Accessed August 2009.

[37] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 1997.

[38] M. Naor, B. Pinkas, and O. Reingold. Distributed Pseudo-random Functions and KDCs. In *EUROCRYPT'99*, pages 327–346, 1999.

[39] J. B. Nielsen. A Threshold Pseudorandom Function Construction and Its Applications. In *CRYPTO'02*, pages 401–416, 2002.

[40] A. Patra, A. Choudhary, and C. Pandu Rangan. Efficient Asynchronous Verifiable Secret Sharing and Byzantine Agreement with Optimal Resilience. Cryptology ePrint: 2008/424, 2008.

[41] T. P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *Eurocrypt'91*, pages 522–526. Springer-Verlag, 1991.

[42] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91*, pages 129–140, 1991.

[43] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[44] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[45] D. A. Schultz, B. Liskov, and M. Liskov. MPSS: Mobile Proactive Secret Sharing. *ACM Trans. Inf. Syst. Secur.*, 13(4):34, 2010.

[46] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[47] D. R. Stinson. *Cryptography: Theory and Practice*. CRC Press, Inc., 3rd edition, 2005.

[48] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. In *30th International Conference on Distributed Computing Systems (ICDCS 2010)*, pages 263–272, June 2010.

[49] M. Young, A. Kate, I. Goldberg, and M. Karsten. Towards Practical Communication in Byzantine-Resistant DHTs. To appear in IEEE/ACM Transactions on Networking (ToN), 2012.

[50] L. Zhou, F. B. Schneider, and R. van Renesse. APSS: Proactive Secret Sharing in Asynchronous Systems. *ACM Trans. Inf. Syst. Secur.*, 8(3):259–286, 2005.

A Analysis of $\mathsf{HybridDKG}_{\mathbf{Ped}}$ with Uniform Randomness of the Shared Secret

Here, we analyze the $\mathsf{HybridDKG}_{\mathrm{Ped}}$ protocol that achieves uniform randomness of the shared secret. The liveness and agreement proofs are the same as those of $\mathsf{HybridDKG}_{\mathrm{DLog}}$in Section 5.2.

Correctness (DKG-C). For correctness, we need to prove the following three properties.

1. There is an efficient algorithm that on input shares from $2t+1$ nodes and the public information produced by the $\mathsf{HybridDKG}$ protocol, output the same unique value s, even if up to t shares are submitted by malicious nodes.

2. At the end of Sh phase of $\mathsf{HybridDKG}_{\mathrm{Ped}}$, all honest nodes have the same value of public key $Y = g^s$, where s the unique secret guaranteed above.

3. s and Y are uniformly distributed in $\mathbb{Z}_p$ and $\mathbb{G}$ respectively.

The first two properties are the same as those in protocol $\mathsf{HybridDKG}_{\mathrm{DLog}}$ in Section 5 and we only need to prove the third property.

Here, $s = \sum_{P_i \in \overline{\mathcal{Q}}} \alpha_i$. As long as there is one value α_i in this sum that is chosen at random and independently from the other values in the sum, the uniform distribution of s is guaranteed. All α_i values are only available in the form of Pedersen commitments until set $\overline{\mathcal{Q}}$ is finalized. From Theorem 4.4 of [42], in VSS using Pedersen commitments, the view of the t-limited adversary is

independent of the shared secret. Therefore, with at least one VSS from the honest nodes in the $t+1$ chosen VSSs, s is uniformly distributed and so is $Y = g^s$.

Secrecy (DKG-S). We need to prove that no information about s can be learned by the adversary except for what is implied by $Y = g^s$.

More formally, we prove that for every PPT adversary $\mathcal{A}$ that has up to t nodes, there exists a PPT simulator $\mathcal{S}$ that on input $Y \in \mathbb{G}$ produces an output distribution which is polynomially indistinguishable from $\mathcal{A}$'s view of a run of the HybridDKG protocol that ends with Y as its public key. Our proof is based on the proof of secrecy in [24, Section 4.3].

In Fig. 6, we describe the simulator $\mathcal{S}$ for our HybridDKG protocol. An informal description is as follows. $\mathcal{S}$ runs a HybridDKG instance on behalf of all honest nodes. For most of the protocol (until message DKG-completed is to be sent), it follows the protocol HybridDKG as instructed. For DKG-completed messages, it changes the public key shares $Y_i = g^{s_i}$ to "hit" the desired public key Y. $\mathcal{S}$ knows all g^{s_j} and $g^{s'_j}$ values for all $P_j \in \mathcal{B}$, as it chooses $\phi^{(j)}(x,y)$ for good nodes and has received enough shares from bad nodes to reconstruct the bivariate polynomials shared by them. For $i \in [t+1, n]$, $\mathcal{S}$ sets $g^{s_i^*}$ as interpolation (in the exponent) of $(0, Y)$ and (j, g^{s_j}) for $j \in [1, t]$. It creates the corresponding NIZKPK$_{\equiv Com}$ using the random oracle hash table.

We show that the view of the adversary $\mathcal{A}$ that interacts with $\mathcal{S}$ on input Y is the same as the view of $\mathcal{A}$ that interacts with the honest nodes in a regular run of the protocol that outputs the given Y as the public key.

Algorithm for Simulator $\mathcal{S}$

Let $\mathcal{B}$ be the set of parties controlled by the adversary, and $\mathcal{G}$ be the set of honest parties (run by the simulator). Without loss of generality, $\mathcal{B} = [P_1, P_{t'}]$ and $\mathcal{G} = [P_{t'+1}, P_n]$, where $t' \leq t$. Let $Y \in \mathbb{G}$ be the input public key and $\mathrm{H}_{\equiv Com} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash table for NIZKPK$_{\equiv Com}$.

1. Perform all steps on behalf of the uncorrupted parties $P_{t'+1}, \ldots, P_n$ exactly as in the HybridDKG protocol until the DKG-completed message. Once a node is ready to send the DKG-completed message, the following holds:
   - Set $\overline{\mathcal{Q}}$ is well defined with at least one honest node in it.
   - The adversary's view consists of polynomials $\phi^{(j)}(x,y)$ for $j \in \mathcal{B}$, the share polynomials $a_j^{(i)}y = \phi^{(i)}(j,y)$ for $P_i \in \overline{\mathcal{Q}}$, $P_j \in \mathcal{B}$, and commitments $\mathcal{C}_i$ for $P_i \in \mathcal{Q}$.
   - $\mathcal{S}$ knows all polynomials $\phi^{(i)}(x,y)$ for $P_i \in \overline{\mathcal{Q}}$ as it knows $n - t'$ shares for each of those.
2. Perform the following computations for each $i \in [t+1, n]$ before starting Step 6:
   (a) Compute s'_j for $P_j \in [1, n]$ and s_j for $P_j \in \mathcal{B}$. Interpolate (in the exponent) $(0, Y)$ and (j, g^{s_j}) for $j \in [1, t]$ to compute $\mathcal{C}_{\langle g \rangle}(s_i^*) = g^{s_i^*}$.
   (b) Compute the corresponding NIZKPK$_{\equiv Com}$ by generating random challenge $c_i \in_R \mathbb{Z}_p$ and responses $u_{i,1}, u_{i,2} \in_R \mathbb{Z}_p$, computing the commitments $t_{i,1} = (g^{s_i^*})^{c_i} g^{u_{i,1}}$ and $t_{i,2} = \frac{\mathcal{C}_{\langle g,h \rangle}(s_i, r_i)}{\mathcal{C}_{\langle g \rangle}(s_i^*)}^{c_i} h^{u_{i,2}}$ and include entry $\langle (g, h, \mathcal{C}_{\langle g \rangle}(s_i^*), \mathcal{C}_{\langle g,h \rangle}(s_i, r_i), t_{i,1}, t_{i,2}), c_i \rangle$ in the hash table $\mathrm{H}_{\equiv Com}$ so that $\pi_{\equiv Com_n} = (c_i, u_{i,1}, u_{i,2})$.
3. In the end, $s = \sum_{P_i \in \overline{\mathcal{Q}}} \alpha_i$ such that $Y = g^s$.

Figure 6: Simulator for HybridDKG with the uniform randomness property

In a regular run of protocol HybridDKG, $\mathcal{A}$ sees the following probability distribution of data produced by the honest nodes:

- Values $\phi_i(j,y)$, $\phi'_i(j,y)$ for $i \in \mathcal{G}, j \in \mathcal{B}$, uniformly chosen in $\mathbb{Z}_p$

- Values $\mathcal{C}_i$ and g^{s_i} for $P_i \in \mathcal{G}$, that correspond to randomly chosen polynomials.

As we are interested in runs of HybridDKG that end with Y as the public key, we note that the above distribution of values is induced by the choice (of the good parties) of polynomials $\phi_i(x,y)$, ${\phi'}_i(x,y)$ for $P_i \in \overline{\mathcal{Q}}$, uniformly distributed in the family of degree-t polynomials over $\mathbb{Z}_p$ such that $\prod_{P_i \in \overline{\mathcal{Q}}} g^{\phi_i(0,0)} = Y$. Without loss of generality, assume $P_n \in \mathcal{G}$ belongs to $\overline{\mathcal{Q}}$. The above distribution is characterized by the choice of polynomials $\phi_i(x,y)$, $\phi_i^*(x,y)$ for $P_i \in (\mathcal{G} \cap \overline{\mathcal{Q}}) - \{P_n\}$ as random independent degree-t bivariate polynomials over $\mathbb{Z}_p$ and of $\phi_n(x,y)$ as a uniformly chosen polynomial from the family of degree-t bivariate polynomials over $\mathbb{Z}_p$ that satisfy the constraint $\phi_n(0,0) = s - \sum_{P_i \in \overline{\mathcal{Q}} \setminus \{n\}} \phi_i(0,0)$.

We show that the distribution of outputs of the simulator $\mathcal{S}$ is *identical* to the above distribution. Note that the above distribution depends on the set $\overline{\mathcal{Q}}$. Since all actions of the simulator until $\overline{\mathcal{Q}}$ is (eventually) delivered to all nodes are identical to the actions of honest parties interacting with $\mathcal{A}$ in a real run of the protocol, we are assured that the set $\overline{\mathcal{Q}}$ defined in this simulation is identical to its value in the real protocol.

We now describe the output distribution of $\mathcal{S}$ in terms of degree-t bivariate polynomials ϕ_i^* corresponding to the choices of the simulator. For $P_i \in (\overline{\mathcal{Q}} - \mathcal{B} - \{P_n\})$, set ϕ_i^* to ϕ_i and ${\phi'}_i^*$ to ${\phi'}_i$. Define ϕ_n^* such that the values $\phi_n^*(0,0) = \log_g(\frac{Y}{\prod_{j \in (\overline{\mathcal{Q}} - \mathcal{B} - \{P_n\})} g^{\alpha_j^*}})$ and $\phi_n^*(j,y) = \phi_n(j,y)$ for $j \in [1,t]$. Finally, define ${\phi'}_n^*(x,y)$ such that $\phi_n^*(x,y) + \Lambda {\phi'}_n^*(x,y) = \phi_n(x,y) + \Lambda {\phi'}_n(x,y)$, where $\Lambda = \log_g(h)$. It can be seen by this definition that the univariate polynomial evaluations of these polynomials evaluated at the indices for $P_j \in \mathcal{B}$ coincide with the values $\phi_i(j,y)$ which are seen by the corrupted parties in the protocol. Note that the above DLog values $\phi_n^*(0,0)$ and ${\phi'}_n^*(0,0)$ are unknown to the simulator. Also, the commitments of these polynomials agree with $\mathcal{C}_i$ published by the simulated honest parties in the protocol as well as with the exponentials $g^{s_i^*}$ for $P_i \in \mathcal{G}$ published by the simulator at the end on behalf of the honest parties. Thus, these values pass the verifications in the real protocol.

It remains to be shown that polynomials ϕ_i^* and ${\phi'}_i^*$ belong to the right distribution. Indeed, for $\overline{\mathcal{Q}} - \mathcal{G} - \{P_n\}$ this is immediate since they are defined identically to ϕ_i which are chosen according to the uniform distribution. For ϕ_n^* we see that this polynomial evaluates in points $j = [1,t]$ to random values $(\phi_n(j,y))$ while at 0 it evaluates $\log_g(g^{\alpha_n^*})$ as required to hit Y. Finally, ${\phi'}_n^*$ is defined as ${\phi'}_n^*(x,y) = \Lambda^{-1}(\phi_n(x,y) - \phi_n^*(x,y) + {\phi'}_n(x,y))$ and since ${\phi'}_n^*(x,y)$ is random and independent then so is ${\phi'}_n^*(x,y)$.