Adaptively Secure, Universally Composable Distributed Generation of Discrete-Logarithm Based Keys

Hanna Ek[1], Kelsey Melissaris[2], and Lawrence Roy[3]

[1] Chalmers University of Technology & University of Gothenburg, Göteborg, Sweden
[2] Université Paris Cité, CNRS, IRIF, Paris, France
[3] Aarhus University, Aarhus, Denmark

Abstract. Distributed key generation (DKG) protocols enable a set of parties to distributively generate a threshold-shared key pair $(\mathsf{pk}, \mathsf{sk})$, such that at least t parties must participate to reconstruct the secret. We introduce the first DKG protocol for discrete-logarithm based keys that are both *universally composable* and *adaptively secure* without erasure, inconsistent players, interactive assumptions, or oracle-aided simulation.

Our contributions are as follows:

1. an adaptively secure and universally composable DKG that achieves *guaranteed output delivery* in three rounds assuming an honest majority,
2. an adaptively secure and universally composable *committed* DKG that realizes our novel committed DKG functionality, tolerates a full corruption threshold, and achieves *identifiable abort* in two rounds,
3. an adaptively secure and universally composable *committed* DKG that achieves *guaranteed output* in three rounds assuming an honest majority,
4. as an application, an incredibly simple threshold Schnorr protocol in the committed DKG-hybrid model–implying an adaptively secure and universally composable threshold Schnorr protocol tolerating a full corruption threshold with *identifiable abort* in three rounds, and an adaptively secure and universally composable threshold Schnorr protocol for an honest majority with *guaranteed output* in four rounds.

Most importantly, our DKG constructions are secure in the random oracle model under the DDH assumption. Our output guarantees are proven under the assumption of synchrony. All existing synchronous DKG protocols for discrete-logarithm based keys satisfy weaker security notions or require stronger assumptions.

Table of Contents

Adaptively Secure, Universally Composable Distributed Generation of Discrete-Logarithm Based Keys 1
Hanna Ek, Kelsey Melissaris, and Lawrence Roy
1 Introduction . . . 3
1.1 The Challenge to Universally Composable Adaptive DKGs . . . 3
1.2 Contributions . . . 4
1.3 Related Work . . . 5
1.3.1 Approaches to Adaptive Security . . . 5
1.3.2 The Most Recent Related Works. . . . 6
Table 1: Synchronous DKG Protocols . . . 7
1.4 Technical Overview . . . 7
1.4.1 Intuition: Three Round Protocol . . . 8
Figure 1: Overview of 3-Round DKG Protocol $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ with Guaranteed Output . . . 8
1.4.2 Intuition: Committed DKG Protocol . . . 11
Figure 2: Overview of 2-Round Committed DKG $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ with Identifiable Abort . . . 12
1.4.3 Intuition: Threshold Schnorr . . . 13
Figure 3: Overview of Threshold Schnorr Protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ . . . 13
2 Preliminaries . . . 13
2.1 Model . . . 14
2.2 Secret Sharing . . . 16
3 Adaptive Zero Knowledge . . . 17
3.1 Adaptive Non-Interactive Zero Knowledge . . . 18
Figure 4: $(\mathsf{Prv}, \mathsf{Ver}, \mathcal{S}^{\mathsf{R}})$, Adaptive NIZKs for Relations $\mathcal{R}_{\mathsf{pd}}$ and $\mathcal{R}_{\mathsf{eq}}$ . . . 20
3.2 Universally Composable Adaptive Zero Knowledge . . . 21
Figure 5: $\mathcal{F}_{\mathsf{azk}}$ Adaptive Zero Knowledge Functionality . . . 21
Figure 7: Π_{azk} Adaptive Zero Knowledge Protocol . . . 22
Theorem 1: Adaptive UC Security of Adaptive Zero Knowledge Protocol . . . 22
4 Distributed Key Generation with Guaranteed Output Delivery . . . 23
4.1 The Distributed Key Generation Functionality . . . 23
Figure 8: $\mathcal{F}_{\mathsf{dkg}}$ DKG Functionality . . . 23
4.2 Three Round DKG with Guaranteed Output Delivery . . . 24
Figure 9: $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ DKG Protocol with Guaranteed Output Delivery . . . 25
Theorem 2: Adaptive UC-Security of DKG Protocol $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ . . . 27
5 Committed Distributed Key Generation with Identifiable Abort . . . 42
5.1 The Committed DKG Functionality . . . 42
Figure 14: $\mathcal{F}_{\mathsf{cdkg}}$ Committed DKG Functionality . . . 42
5.2 Two Round Committed DKG Protocol with Identifiable Abort . . . 43
Figure 15: $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Committed DKG Protocol with Identifiable Abort in Two Rounds . . . 44
Theorem 3: Adaptive UC-Security of Committed DKG Protocol $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ . . . 46
5.3 Three Round Committed DKG Protocol with Guaranteed Output . . . 55
Figure 17: $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$ Committed DKG Protocol with Guaranteed Output in Three Rounds 55
Theorem 4: Adaptive UC-Security of Committed DKG Protocol $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$ . . . 57
6 Adaptively Secure Threshold Schnorr . . . 57
6.1 The Ideal Threshold Schnorr Functionality . . . 57
Figure 18: $\mathcal{F}_{\mathsf{sch}}$ Schnorr Signing Functionality . . . 57
6.2 Three Round Threshold Schnorr with Identifiable Abort . . . 58
Figure 19: $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ Schnorr Protocol with Identifiable Abort . . . 58
Theorem 5: Adaptive UC-Security of Schnorr Protocol . . . 59

1 Introduction

Distributed key generation (DKG) enables a set of n parties to jointly compute a public key pk with a (t, n)-threshold shared secret key sk. DKG protocols are fundamental to threshold cryptography, underpinning, e.g., threshold signatures [Des88; DF90], blockchain concensus mechanisms, and distributed systems [Gur+21; KMS20; Syt+17], increasingly popular areas that have received overwhelming attention in recent years including formal standardization efforts by NIST [BP25]. Of particular interest is the generation of discrete-logarithm based keys [GJKR07] in which the public key $\mathsf{pk} = \mathsf{sk}G$ is an element of a cyclic group, and the secret key is its' discrete logarithm. Many widely adopted digital signature schemes, such as BLS [BBS04], Schnorr [Sch91], ECDSA [GJKR96], and their thresholdized variants [Bol03; SS01], have discrete logarithm based keys. DKG protocols [Ped92; CGGMP20] are primarily proven secure in the presence of static adversaries that corrupt parties prior to protocol execution. Security against more realistic, yet significantly stronger, *adaptive* adversaries that corrupt parties throughout the lifetime of a system has been proven in weaker models, via reductions to interactive assumptions, or incurr a security loss from not tight reductions (see Section 1.3 and Table 1).

1.1 The Challenge to Universally Composable Adaptive DKGs

We study adaptively secure DKGs in one of the strongest possible model(s) for multiparty computation (MPC): the universal composition framework (UC) [Can01], without erasures or relaxed composition theorems.[4] The UC framework (see Section 2.1) has become the "gold standard" for MPC security in many areas, as it provides strong guarantees for arbitrary composition from the requirement of straight-line simulation. Many existing statically secure DKGs inhabit a sort-of gray-area; there is seemingly no way to effectively simulate the protocol, but adaptive corruption does not appear to enable recovery of the secret key. This begs the following question: *is there a natural way to describe the extra power adaptive corruption gives the adversary in such protocols?*

We argue that MPC-in-the-head proofs of knowledge [IKOS07] provide such a description. In the MPC-in-the-head framework a prover runs an MPC protocol that is required to be statically secure against less than t semi-honest corruptions *in its head*, i.e., by playing the part of every party in the execution. The prover then sends the protocol output and commitments to each party's view in the execution to the verifier. The verifier samples a random subset of $t-1$ parties to (*adaptively*) corrupt, and the prover opens the commitments to the views of these parties for the verifier. A successfully cheating prover must correctly guess which subset of parties the verifier would choose to corrupt. Similarly, an adaptive adversary against an MPC protocol can first observe the protocol outputs then choose a subset of parties to adaptively corrupt. It seems that MPC-in-the-head can be built from protocols that cannot be simulated adaptively, as such a simulator could forge the corresponding MPC-in-the-head proof of knowledge. The parallel to MPC-in-the-head illuminates exactly what adaptive corruption provides to the adversary: the ability to prove knowledge of the secret key.

Specifying to DKG protocols, the MPC-in-the-head commitment to each party's view is essentially the view of the adversary in the DKG execution, including the public transcript of the the protocol and any private communication given to the adversary. As an example, a statically secure DKG might use Feldman VSS [Fel87] to threshold-share a secret key. Feldman VSS publicly reveals perfectly binding commitments $\mathsf{sk} \cdot G$ to each party's key share. If in this example the key is additively shared between two parties as $\mathsf{sk} = x + y$, then $x \cdot G$ and $y \cdot G$ are published, where $\mathsf{pk} = xG + yG$ is the public key. These commitments are sufficient to prove soundness of the corresponding MPC-in-head proof of knowledge of sk. In fact, in the two-party case this proof-of-knowledge is equivalent to that in Schnorr's identification protocol with a 1-bit challenge space: x is the nonce, and based on the challenge the prover reveals either x or $k - x$. Adaptive corruption of real parties in an actual DKG execution can be used to convince a verifier (with soundness $\frac{1}{2}$) that the adversary knows k, even though the adversary has only corrupted a single party and therefore should know *nothing* about k.

[4] Our protocols do use programmable random oracles.

Crites and Stewart [CS25] show that this process can be used to attack real protocols if a certain computational problem can be solved efficiently. Roughly, they show a generalization of this correspondence between Feldman VSS key shares and Schnorr proofs can be used to forge a signature by mapping the Schnorr challenge into a subset of parties to corrupt that will reveal the response to this challenge. This attack, on its own, is sufficient motivation for fully adaptive DKG protocols—a model in which such bad interaction between protocols cannot occur.

1.2 Contributions

We present protocols for DKG with UC adaptive security assuming broadcast, a public key infrastructure (PKI), a programmable random oracle, and Decisional Diffie-Hellman (DDH).[5] Ours are the first DKG protocols for discrete-logarithm based keys that are UC-secure against adaptive adversaries without secure erasure, high security loss, or interactive assumptions (see Section 1.3 and Table 1).[6]

Our contributions are as follows:

- ⋆ **Protocol $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$.** A 3-round adaptive UC DKG that achieves guaranteed output delivery in the presence of an honest majority ($t < \frac{n}{2}$) from DDH and NIZKs that satisfy *zero knowledge against adaptive corruptions* in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model. Our protocol requires a one-time setup, after which parties can execute many three round distributed key generation subsessions. The protocol appears in Figure 9.
- ⋆ **Protocol $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$.** A 2-round adaptive UC committed DKG that achieves identifiable abort against a dishonest majority ($t < n$) from DDH and NIZKs that satisfy *zero knowledge against adaptive corruptions* in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model. The protocol similarly requires a one time setup followed by many two round distributed key generation subsessions, and realizes our novel *committed* DKG functionality $\mathcal{F}_{\mathsf{cdkg}}$ that outputs perfectly hiding commitments to the secret key share of every party. The protocol appears in Figure 15.
- ⋆ **Protocol $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$.** A 3-round adaptive UC committed DKG that achieves guaranteed output for an honest majority ($t < n/2$) from DDH and NIZKs that satisfy *zero knowledge against adaptive corruptions* in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model. The protocol similarly requires a one time setup followed by many two round distributed key generation subsessions, and realizes our novel *committed* DKG functionality $\mathcal{F}_{\mathsf{cdkg}}$ that outputs perfectly hiding commitments to the secret key share of every party. The protocol appears in Figure 17.
- ⋆ **Protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$.** An adaptive UC threshold Schnorr protocol in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$-hybrid model with security from the DL assumption that preserves the output guarantees of whichever protocol realizes $\mathcal{F}_{\mathsf{cdkg}}$; for $t < n$ protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ has identifiable abort when $\mathcal{F}_{\mathsf{cdkg}}$ is realized with identifiable abort, and for $t < \frac{n}{2}$ protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ has guaranteed output delivery when $\mathcal{F}_{\mathsf{cdkg}}$ is realized with guaranteed output delivery. The protocol appears in Figure 19.

After a one-time setup we achieve a three-round DKG with guaranteed output delivery in the honest majority setting, and a two-round committed DKG with identifiable abort that tolerates the full corruption threshold. With the second protocol, we strengthen our results to realize a DKG functionality with *committed output*, where parties receive a fresh Pedersen commitments to every party's key shares.

Our DKG naturally lends itself to an almost trivial 3-round adaptive threshold Schnorr signing protocol (with identifiable abort): parties execute the DKG to generate a threshold-shared nonce R, hash to compute the challenge $e \leftarrow \mathsf{H}_{\mathsf{sch}}(\mathsf{pk}, R, m)$, then open a linear combination of the key shares and the nonce shares: $\sigma_i := \mathsf{sk}_i + e \cdot r_i$. We stress that $\mathsf{H}_{\mathsf{sch}}$ is not assumed to be a random oracle; our protocols are adaptively secure without programming $\mathsf{H}_{\mathsf{sch}}$. We also stress that our protocol realizes a threshold *signing* functionality

[5] We need to hash into the group, which means that we need a reverse sampling algorithm for it. Reverse sampling seems to be necessary to avoid non-black-box evaluation of group operations inside MPC; without the reverse sampling algorithm the simulator would instead have to program a representation of the group element into the random oracle or honest party messages, and decoding that representation would likely be non-black-box.

[6] Excluding, of course, those constructions based on evaluating non-black-box group operations inside generic adaptively secure MPC.

(like [DKLs24]), and not a threshold signature functionality (like [CGGMP20]); we realize the stronger functionality that computes and outputs an honest Schnorr signature. Threshold signing functionalities are stronger, in that they guarantee the signatures output by the protocol behave in all respects exactly like honestly generated signatures, as opposed to exclusively modeling unforgeability.

There is one respect in which our DKG functionalities are potentially weaker: in order to save a round of communication in our identifiable abort protocol, we allow the adversary to choose their key shares *after* seeing the public key. This also means that we do not require the corrupted party key shares to be fully uniform, only the public key itself, as is relatively standard for DKG functionalities (see [Kat24]). We note that our protocols could easily upgrade to fully random key shares by running a coin flip at the end to rerandomize the shares with a sharing of zero. Our application to Schnorr demonstrates that this functionality is still useful.

1.3 Related Work

In Section 1.3.1 we provide a broad overview of various tactics to prove adaptive security. We focus on the generation or use of discrete logarithm-based keys with adaptive security proven from standard assumptions without these tactics. In Section 1.3.2 we focus on prior work that is closely related to ours—coincidentally, a series of works on threshold signature schemes. For a thorough review on DKG, please see this recent SoK [BK25], from which many entries of Table 1 were retrieved. We note that Table 1 supports our claim that all known synchronous distributed generation of discrete-logarithm based keys protocols that are adaptively secure rely on either secure erasure, or the SIP framework.

1.3.1 Approaches to Adaptive Security We discuss weakened functionalities, secure erasure, inconsistent players, and oracle-aided simulation.

Weakened Functionalities. One common approach is to prove adaptive security against a weakened DKG functionality that outputs commitments to each party's key share (e.g., $\mathsf{sk}G$) the simulator cannot otherwise simulate. See [Kat24] for a recent work in this line. This approach allows for relatively simple protocols and straightforward simulators, but leakage of this share-related information can introduce problems to any higher-level protocol built on top, e.g., Schnorr threshold signatures become vulnerable to the Crites–Stewart attack briefly discussed above.

Secure Erasure. Another natural approach to adaptive security is to require that honest parties erase any portions of their secret state that cannot be simulated. This way, any commitments produced as part of a protocol simulation that cannot be opened by the simulator in an ideal-world adaptive corruption similarly cannot be opened in a real-world adaptive corruption. Unfortunately, only short-term secrets and ephemeral random values can be erased; the secret key share of a party is required for meaningful use of the shared key, e.g., subsequent threshold signatures, and therefore cannot be erased. Unfortunately, this secret key share is exactly the value that is difficult to simulate. Thus, even assuming secure erasure, a DKG that reveals $\mathsf{pk}_i := \mathsf{sk}_i \cdot G$ to all parties (for all shares sk_i) is not trivially adaptively secure. Despite this, erasure has been used to prove adaptive security of Schnorr threshold signatures [Mak22]. The assumption of secure erasure is reflected in the **Assumptions** column of Table 1.

Furthermore, secure erasure [CGJKR99] is a seemingly strong assumption. It can be difficult to ensure that information is truly erased on modern systems, which include many levels of abstraction that are often not built for secure erasure. In general, compilers, operating systems, virtual machines, and even CPU architectures are designed and optimized for correctness, not to forget information whenever it's no longer needed.

The Single Inconsistent Player. The single inconsistent player (SIP) technique, first proposed by [CGJKR99; FMY99a; FMY99b] and subsequently adapted to the (SIP-)UC framework [AF04], weakens simulation by allowing the simulator to fail if the adversary corrupts a single, randomly selected, honest party called

the single inconsistent player. This technique is common for proving adaptive security in conjunction with various computational assumptions in a variety of security models [JL00; LP01; ADN06; Bra+24; GCRS25] (see Table 1). If the adversary's view is (computationally) independent of the identifier of the inconsistent player until its corruption then simulation at most reduces the adversary's advantage by $(n-t+1)/n$ (multiplicatively). This solution partially solves the problem and puts a polynomial bound on the impact of the MPC-in-the-head attack discussed in Section 1.1: it is only possible to forge proofs of knowledge that have soundness error at least $(n-t+1)/n$. Proofs such as Schnorr with sufficiently low soundness error are likely not prohibitively less secure in this model, and for this reason SIP is often used in adaptively secure Schnorr threshold protocols. One technique that is not available in the UC framework to reduce security loss is rewinding. If the adversary issues corruption instructions that cannot be simulated then the simulator can rewind the adversary and try again with new choices. Note that rewinding is not compatible, in general, with even parallel composition.

The SIP-UC technique restricts composition; an SIP-UC secure protocol can only be composed with protocols executed by the *same set of parties with the same inconsistent player*. Multiple protocols running across distinct sets of parties would require the simulator to guess multiple inconsistent parties, incurring an exponential reduction in the probability of successful simulation. Referring back to our MPC-in-the-head parallel, this blow up potentially occurs because parallel repetition can be used to amplify soundness in Section 1.1; independent protocols, each allowing the forgery of some low soundness proof of knowledge (such as our $kG = xG + yG$ example above), could be combined to achieve negligible soundness error in the ideal, yet forging each of the individual proofs results in a forgery of the overall scheme. The SIP framework is reflected in the **Sec. Model** column of Table 1; SIP-SA and SIP-UC standing for the stand-alone and UC variants, respectively.

One-More Oracles, and Oracle-Aided Simulation. Another frequent approach to proving adaptive security is to give the simulator limited access to a discrete logarithm oracle, or an oracle that otherwise aids the simulator in producing the contents of the state of a newly corrupted party [BL22]. With $t-1$ queries to a discrete logarithm oracle, a simulator can simulate commitments to key shares pk_i such that these commitments interpolate to the targeted public key pk, and upon an adaptive corruption of a party query the oracle with $\mathsf{pk}_i = \mathsf{sk}_i \cdot G$ to open the commitment, receiving the key share sk_i. Oracle-aided simulatability is reflected as OA in the **Model** column of Table 1.

The One-More Discrete Logarithm (OMDL) assumption [BNPS03] is an interactive, unfalsifiable variant of the Discrete Logarithm assumption (Def 1) which states that no PPT adversary can compute the discrete logarithm of q group element challenges given access to an (inefficient) oracle for at most $q-1$ discrete logarithms. The weaker Algebraic OMDL assumption (AOMDL) [NRS21] requires the adversary to submit an algebraic representation of each query to the oracle, and therefore is falsifiable. Note that such an oracle significantly weakens the security of any protocol built on top, limiting composition. However, one can construct adaptively secure threshold signatures on top of a DKG that is secure under a one-more assumption [BLTWZ24; Che25; BLSW24].

1.3.2 The Most Recent Related Works.

In this section we focus exclusively on adaptively secure protocols that generate or use discrete logarithm-based keys with security proven without the tactics outlined above in Section 1.3.1. Referring to Table 1 and [BK25], it is clear that there are no such DKG protocols straightforwardly. Instead, the most related recent works present unforgeable threshold signatures with adaptive security. In this section we discuss a series of works with a non-trivial intersection of authors [DR24; BDLR25b; BDLR25a] that present and gradually improve upon a specific tactic for adaptive security. Each paper in the line of work focuses on *game-based* security of *threshold signatures*, as opposed to UC-security of DKG. We stress that an adaptively secure threshold signature *does not* imply an adaptively secure DKG under the same assumptions, whereas our adaptive DKG *does* imply an adaptively secure threshold signature as demonstrated in Section 6.

First, Das and Ren [DR24] presented an adaptively secure existentially unforgeable (game-based EUF-CMA) threshold BLS signature scheme from the DDH and co-CDH in asymmetric pairing groups. The

Protocol	Model	t	RC	Assumptions
[CGJKR99]	SIP-SA	$n/2$	$5 \cdot \mathsf{BC}_n$	SE, ZKP, DL
[JL00]	SIP-SA	$n/2$	$10 \cdot \mathsf{BC}_n$	aZKP, DDH
[AF04]	SIP-UC	$n/2$	$5 \cdot \mathsf{BC}_n$	DDH
[FMT24]	OA-SA	$n/2$	$2 \cdot \mathsf{BC}_n$	SE, NIZK, PKI, RO, DDH
This Work: $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$	UC	$n/2$	$3 \cdot \mathsf{BC}_n$	aNIZK, PKI, RO, DDH
This Work: $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$	UC	$n/2$	$3 \cdot \mathsf{BC}_n$	aNIZK, PKI, RO, DDH
This Work: $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$	UC	n	$2 \cdot \mathsf{BC}_n$	aNIZK, PKI, RO, DDH

Table 1: Comparison of existing adaptively secure synchronous DKG protocols [BK25] to ours. Protocol $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ is the DKG protocol in Figure 9 of Section 4, while $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ and $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$ are the committed DKG protocols of Figures 15 and 17 of Section 5. The **Model** column indicates the model: SA is stand-alone simulation, UC is universal composition, SIP is the addition of the single inconsistent player framework, and OA is oracle-aided simulation. The t column indicates the corruption threshold. The **RC** column indicates the round complexity of the protocol, where BC_n is the round complexity of broadcast to n parties. The **Assumptions** columns indicate the assumptions: (a)ZKP denote (adaptive) interactive distributed verifier zero knowledge, (a)NIZK denote (adaptive) non-interactive zero knowledge, PKI is a public key infrastructure, RO is the random oracle model, DL is discrete-logarithm, DDH is decisional Diffie-Hellman, SE is secure erasure.

assume an adaptively secure DKG, and present an adaptively secure DKG using SIP-based simulation (see SIP, above), which may make it seem unrelated at first glance. In an approach similar to ours, their DKG protocol uses a generalized Pedersen VSS—parties threshold share a random secret and two independent sharings of zero—but is proven adaptively secure using SIP-based simulation, as honest parties are bound to their random secrets in the first round. Their threshold BLS simulator programs a random oracle to output group elements that depend on the "target" signature—the target signature being the required output of the simulated protocol—in a manner that is similar to our DKG simulators.

Next, Bacho, Das, Loss and Ren presented *Glacius* [BDLR25b], an adaptive EUF-CMA threshold Schnorr scheme. The construction is proven to satisfy a novel game-based definition of identifiable abort in five rounds and tolerate a full corruption threshold ($t < n$), and can be compared to our protocol with identifiable abort. Subsequently the same authors presented *Gargos* [BDLR25a], a three round threshold Schnorr protocol with the same properties under the same assumptions.[7] Both protocols are secure under the DDH assumption in the ROM, similarly to our protocols, and are proven secure via a simulation technique that programs a random oracle to output target-related group elements, similarly to our DKG simulator.

With respect to threshold signing, as an application of our DKG we present a threshold Schnorr signature protocol that is adaptively secure, requires 3 rounds, achieves identifiable abort and tolerates a full corruption threshold under the same assumptions (DDH + ROM). This scheme is UC-secure and realizes the threshold signing functionality, therefore achieves a stronger notion of security than the works discussed above. We note that our protocols are built atop a our committed DKG protocol that in turn relies on broadcast and a setup, two assumptions that are avoided in the above line of work.

1.4 Technical Overview

Our setting is the following: the n parties in $\mathcal{P}$ generate a public key $\mathsf{pk} = \mathsf{sk}G$ in a group $\mathcal{G} = (\mathbb{G}, G, q)$. In Section 1.4.1, towards building intuition, we focus on the construction of our three round protocol with guaranteed output delivery $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ that is overviewed in Figure 1. In Section 1.4.2 we discuss our two-round committed DKG $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ that is presented in Figure 15.

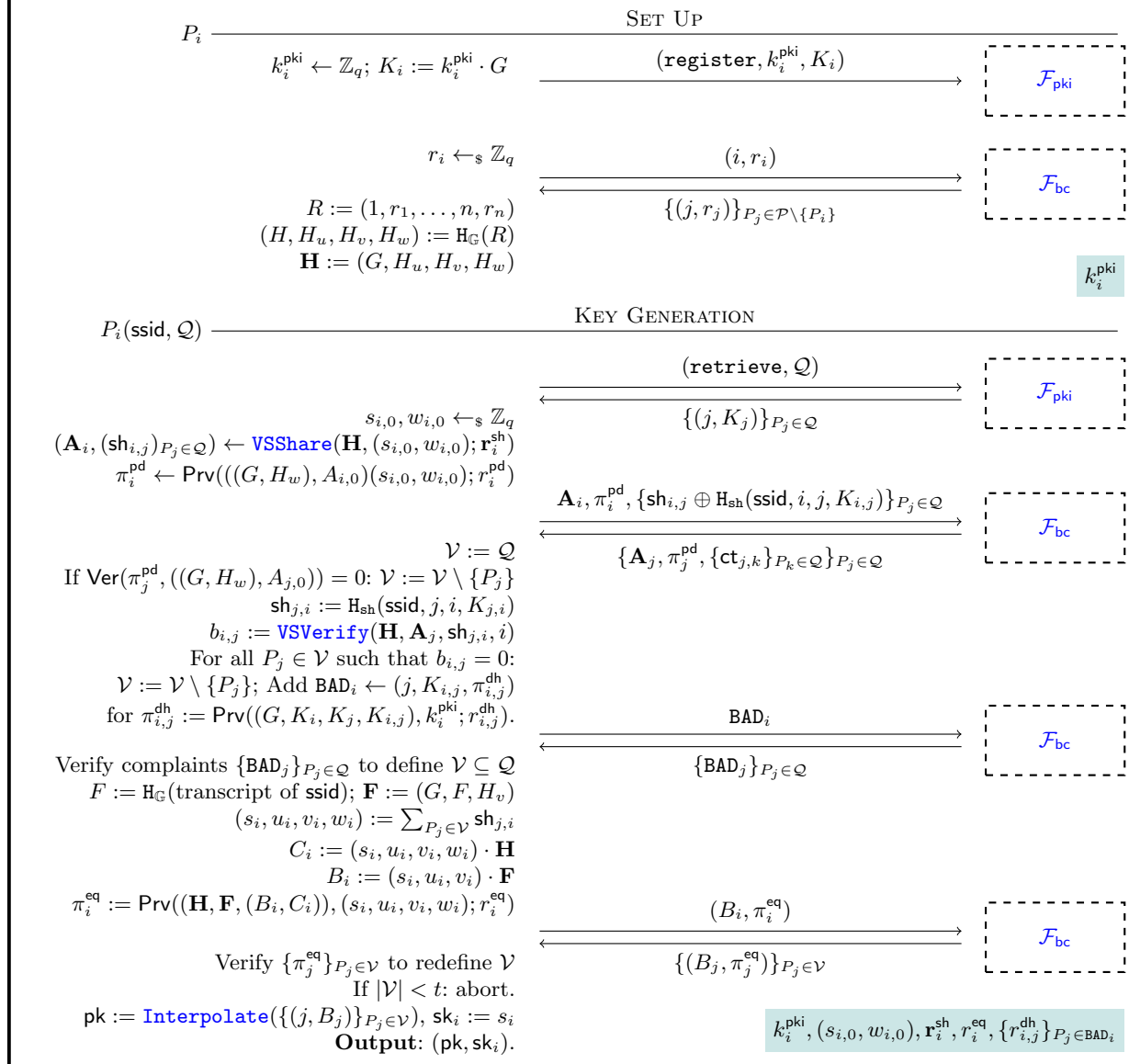
[7] While the authors conjecture that the scheme satisfies game-based identifiable abort and do not directly prove it, there is no reason to believe that the property is not satisfied.

1.4.1 Intuition: Three Round Protocol

We begin with our DKG protcol of Section 4.

Figure 1: Overview of $\Pi^{\mathsf{go}}_{\mathsf{dkg}}(\mathcal{G}, t)$.
Three Round DKG with Guaranteed Output if $t < n/2$, $\mathcal{Q} = \mathcal{P}$, and $\mathcal{F}_{\mathsf{bc}}$ has bounded delay.

The protocol is executed by the n parties in $\mathcal{P}$ in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{ro}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model, and is parameterized by $\mathcal{G} = (\mathbb{G}, G, q)$ a group $\mathbb{G}$ with generator G, and a number $t < n/2$. The pair $(\mathtt{H}_{\mathbb{G}}, \mathtt{H}_{\mathtt{sh}})$ are random oracle instances. The pair $(\mathtt{VSShare}, \mathtt{VSVerify})$ is a VSS (Def 5). Proofs $\pi_i^{\mathsf{pd}}, \pi_i^{\mathsf{eq}}, \pi_{i,j}^{\mathsf{dh}}$ are NIZKs (Sec 3.1) for relations $\mathcal{R}_{\mathsf{pd}}, \mathcal{R}_{\mathsf{eq}}, \mathcal{R}_{\mathsf{dh}}$. Party state contents are highlighted.



Start with Feldman Verifiable Secret Sharing (VSS). Consider the following (insecure) protocol that constructs a DKG by requiring that all n concurrently execute a Feldman VSS [Fel87]. Each party $P_i \in \mathcal{P}$

samples a uniform secret $s_i \in \mathbb{Z}_q$ and sets the secret as the free term of a degree $t-1$ sharing polynomial $S(X) = s_0 + \sum_{k=1}^{t-1} s_k X^k$ over $\mathbb{Z}_q$, as in Shamir threshold secret sharing. Then, P_i commits to the coefficients of the sharing polynomial $C_k := s_k G$, broadcasts the commitments $(C_k)_{k\in[t-1]}$, and privately distributes the threshold shares $S(j)$ as the evaluation of the polynomial on the identifier of the party $P_j \in \mathcal{P}$. Once the shares $S_j(i)$ are received from each $P_j \in \mathcal{P}$, the received shares can be verified against the broadcast commitments by evaluating the committed polynomial:

$$S_j(i)G = \sum_{k\in[t-1]} (i^k) \cdot C_{j,k}$$

If verification does not hold then the receiver P_i can broadcast a complaint against the dealing party P_j, and P_j responds to any complaints by broadcasting a verifying share for the complainant. Any party that was the subject of a complaint and did not respond with a verifying share is disqualified. At the end of the complaint phase parties define their secret key as the sum of the shares received from the remaining qualified parties $\mathsf{sk} := \sum_j S_j(i)$, and the resulting public key to be the sum of the commitments to the zero coefficients of those parties $\mathsf{pk} := \sum_j C_{j,0}$. This protocol [Ped91] has been proven to allow adversarial bias over pk, which is often undesirable for public keys [GJKR99]. Furthermore, as discussed above, the broadcast commitments are perfectly binding to the secrets exchanged between any pair of parties; under the discrete logarithm assumption it is not possible for a simulator to open the commitment of every simulated party that the adversary may choose to corrupt.

Upgrading to Pedersen VSS. Now consider, as an alternative, a variant that uses Pedersen VSS [Ped92] in place of Feldman VSS, thereby replacing the perfectly binding commitments with perfectly hiding Pedersen commitments as in the first half of [GJKR99]. In this protocol parties sample uniform $s_{i,0}, w_{i,0} \in \mathbb{Z}_q$ and set these values as the free terms of two degree $t-1$ sharing polynomials $S_i(X) = s_{i,0} + \sum_{k=1}^{t-1} s_{i,k} X^k$ and $W_i(X) = w_{i,0} + \sum_{k=1}^{t-1} w_{i,k} X^k$. Then, P_i computes pedersen commitments to the coefficients $B_{i,k} := s_{i,k}G + w_{i,k}H$, broadcasts $(B_{i,k})_{k\in[t-1]}$ and privately distributes the shares $S_i(j), W_i(j)$ to each $P_j \in \mathcal{P}$. Received shares can be verified by evaluating the committed polynomial as above, and verification can similarly be followed by complaints and disqualification.

As described this variant cannot yet be used to generate a discrete-logarithm based public key because the resulting aggregate $\mathsf{pk} = \sum_j B_{j,0} = \mathsf{sk}G + w'H$ has a non-zero randomness component $w' = \sum_j w_{j,0}$. The randomness component is generally solved by following the above with a round of Feldman VSS [GJKR03]; Pedersen VSS is used to eliminate bias over pk but must be followed by a round in which parties "reveal" the public key by removing this randomness w' in some way. For Pedersen VSS to be useful for us this opening phase must also be adaptively secure, precluding the standard Feldman VSS second round due to the discussion above.

If the Pedersen commitment randomness $w_{j,0}$ of each party P_j is sampled such that the aggregate $w' = 0$, i.e., if the randomness of the VSS commitments are additive shares of 0, then the resulting protocol could potentially achieve some form of adaptive security. Straightforwardly, this does not immediately work; setting the randomness of each individual party to $0 = w_{j,0}$ binds the parties to their secret shares $s_{j,0}$ *and* re-introduces bias over pk. Instead, this is used as our "reveal" phase in the following protocol.

Three Generator VSS First. To eliminate the possibility of bias over the key we consider Pedersen VSS with correlated commitment randomness, as immediately above, as the second round in which the public key pk is revealed. In the first phase, each $P_i \in \mathcal{P}$ samples uniform $s_{i,0}, w_{i,0} \in \mathbb{Z}_q$ and sets these secrets as the free terms of two degree $t-1$ sharing polynomials $S_i(X) = s_{i,0} + \sum_{k=1}^{t-1} s_{i,k} X^k$ and $W_i(X) = w_{i,0} + \sum_{k=1}^{t-1} w_{i,k} X^k$. In addition, parties compute a degree $t-1$ sharing of 0 as $U_i(X) = \sum_{k=1}^{t-1} u_{i,k} X^k$. Parties then compute pedersen commitments to the coefficients $A_{i,k} := s_{i,k}G + u_{i,k}H_u + w_{i,k}H_w$, broadcast the commitments $(A_k)_{k\in[t-1]}$ and privately distribute the shares $S_i(j), U_i(j), W_i(j)$ to $P_j \in \mathcal{P}$. Received shares are verified by evaluation, as above, and verification is followed by complaints and disqualifications.

Then, parties execute the Pedersen VSS as above. If after this initial phase the parties were to open their commitments $A_{i,k}$ to commitments $(B_{i,k} := s_{i,k}G + u_{i,k}H_u)_{k\in[t-1]}$ the resulting zero commitments $B_{i,0} = s_{i,0}G$ are still binding.

A Crucial Observation. The problem with effectively simulating the above protocol is not that the commitments are binding, but instead the commitments bind the simulator $\mathcal{S}$ to values that it cannot produce. If we adjust how these generators are computed then we solve the real problem.

Observe that at the point that the public key pk must be revealed in the protocol (let's say after the first 3-generator VSS above) the simulator $\mathcal{S}$ has received pk from the ideal functionality $\mathcal{F}$. Therefore if, at this point, the second commitments are opened with respect to a new set of group elements (G, F) that depend on pk as below then $\mathcal{S}$ can compute the Pedersen opening for any potentially corrupted party. Between the first and the second phases above—the three and two generator VSS—parties query a random oracle on the transcript of the protocol thus far to receive a group element $F \in \mathbb{G}$ that is *unpredictable* until $\mathcal{S}$ receives pk from $\mathcal{F}$. In our security proof our simulator $\mathcal{S}$ programs F such that $\mathsf{pk} = sG + \alpha F$ for uniform $s, \alpha \in \mathbb{G}$, then simulates the second round of the protocol such that the parties actually share (s, α) as the scalars of G, F instead of $(\mathsf{sk}, 0)$ as the scalars of G, H_u.

Security is apparent once we note that on the t-th corruption, the corruption at which an adversary would notice that the collection of t corrupted party shares reconstruct the pair (s, α) and not $(\mathsf{sk}, 0)$, is the corruption at which $\mathcal{S}$ receives sk from $\mathcal{F}$. This then allows $\mathcal{S}$ to compute $\mathsf{log}_G(F)$ and equivocate the honest party commitments in order to simulate; to the adversary it looks like sk was shared in the exponent of G and 0 was shared in the exponent of F throughout the protocol execution.

A Stronger Functionality. We realize a DKG functionality that outputs secret key shares sk_i to each of the parties in each DKG subsession. As we are in the UC framework, this means that the environment $\mathcal{Z}$ sees the inputs and outputs of all of the parties, even the honest parties. The above protocol does not yet have enough degrees of freedom for $\mathcal{S}$ to ensure that the openings match the outputs already seen by $\mathcal{Z}$; $\mathcal{S}$ can explain the output of any newly corrupted party, but not in a manner that is consistent with the secret key share sk_i already output to the environment. To this end, we add another group generator.

Four Generators: $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ at a High Level. In this version, parties sample uniform $s_{i,0}, w_{i,0} \in \mathbb{Z}_q$ and set these secrets to be the free terms of two degree $t-1$ sharing polynomials $S_i(X) = s_{i,0} + \sum_{k=1}^{t-1} s_{i,k}X^k$ and $W(X) = w_{i,0} + \sum_{k=1}^{t-1} w_{i,k}X^k$, then compute two degree $t-1$ sharings of 0 as $U_i(X) = \sum_{k=1}^{t-1} u_{i,k}X^k$ and $V_i(X) = \sum_{k=1}^{t-1} v_{i,k}X^k$. Parties compute pedersen commitments to the coefficients $A_{i,k} := s_{i,k}G + u_{i,k}H_u + v_{i,k}H_v + w_{i,k}H_w$, broadcast $(A_{i,k})_{k\in[t-1]}$ and privately distribute the shares $S_i(j), U_i(j), V_i(j), W_i(j)$ to each $P_j \in \mathcal{P}$. Received shares are verified by evaluation, as above, and verification is followed by complaints and disqualifications. After complaints, parties query a random oracle instance on the transcript of the subsession to determine the generator F. In a third round parties broadcast the single aggregate commitment $B_i := s_iG + u_iF + v_iH_v$ for the aggregate $s_i := \sum_j S_j(i), u_i := \sum_j U_j(i), v_i := \sum_j V_j(i)$. The set of B_i interpolate to $\mathsf{sk}G + 0F + 0H_v$, but can be opened by our simulator $\mathcal{S}$ upon an adaptive corruption.

Adaptive Zero Knowledge. This basis change requires parties to prove in zero knowledge that their commitment B_i matches the aggregate commitment to the shares received in the first round $C_i := \sum_j \sum_{k\in[t-1]}(i^k) \cdot A_{j,k}$. To this end, we discuss non-interactive zero knowledge proofs that satisfy the additional requirement of *security against adaptive corruption*; there must exist a simulator that can produce the randomness used to produce any simulated proofs for security against adaptive corruptions. In fact, the relations that are relevant to our protocols can be proven with Fiat-Shamir transformed Σ-protocols, and therefore (almost trivially) satisfy this style of security against adaptive corruption of a simulated prover. We discuss this in Section 3, and present a simple adaptive zero knowledge functionality $\mathcal{F}_{\mathsf{azk}}$ to model straightline extractable NIZKPoKs.

Share Distribution. In our protocol, pairs of parties exchange shares in every subsession. To achieve this with adaptive security we again leverage the random oracle model. Each pair of parties $P_i, P_j \in \mathcal{P}$ share a secret $K_{i,j}$ and, in every subsession ssid, use the one-time-pad to encrypt these shares:

$$\mathsf{ct}_{i,j} := (S_i(j), [\cdots]) \oplus \mathtt{H}_{\mathtt{sh}}(\mathsf{ssid}, i, j, K_{i,j}).$$

Given that $\mathtt{H}_{\mathtt{sh}}$ is a random oracle, the simulator can simply program the random oracle given that it can compute the shares exchanged between a pair of honest parties prior to an adaptive corruption. There are many options for how parties can exchange their pairwise secrets $K_{i,j}$, most notably either (1) each party locally stores $n-1$ shared secrets for share distribution, or (2) parties derive the $n-1$ shared secrets on the fly in each subsession given access to a public key infrastructure $\mathcal{F}_{\mathsf{pki}}$. We choose (2) which allows us to simultaneously simplify our complaints. Therefore, we introduce a Set Up.

Set Up & Verifiable Complaints. During setup each party $P_i \in \mathcal{P}$ registers a discrete-logarithm based public key $K_i = k_i^{\mathsf{pki}} \cdot G$ with $\mathcal{F}_{\mathsf{pki}}$ and in every subsession P_i retrieves the relevant public keys and computes, for every P_j the shared secret $K_{i,j} := k_i^{\mathsf{pki}} \cdot K_j$ as a Diffie-Hellman key. This then allows each party P_i to issue *verifying* complaints by broadcasting $K_{i,j}$ along with a proof of correctness of the shared key whenever the other party P_j distributed a share that does not verify. We thereby eliminate the need for a response round. We note that it is not known how to eliminate the complaint round with adaptive security, as a publicly verifiable VSS is not yet known. This is exactly where the security of our protocols reduces to DDH.[8]

Random Oracle Instead of a CRS. For the generation of group elements $H_u, H_v, H_w \in \mathbb{G}$, we have two options (1) use a CRS functionality $\mathcal{F}_{\mathsf{crs}}$, or (2) have parties generate these elements as the output of an instance of $\mathcal{F}_{\mathsf{ro}}$ during Set Up. We opt for (2) but note that security holds in both cases. To this end, parties broadcast random strings r_i after registering their public key during Set Up. The randomness is used as input to $\mathtt{H}_{\mathbb{G}}$ to compute the group elements that otherwise would be contained within the CRS. Therefore, the adversary can rejection sample the VSS commitment keys.

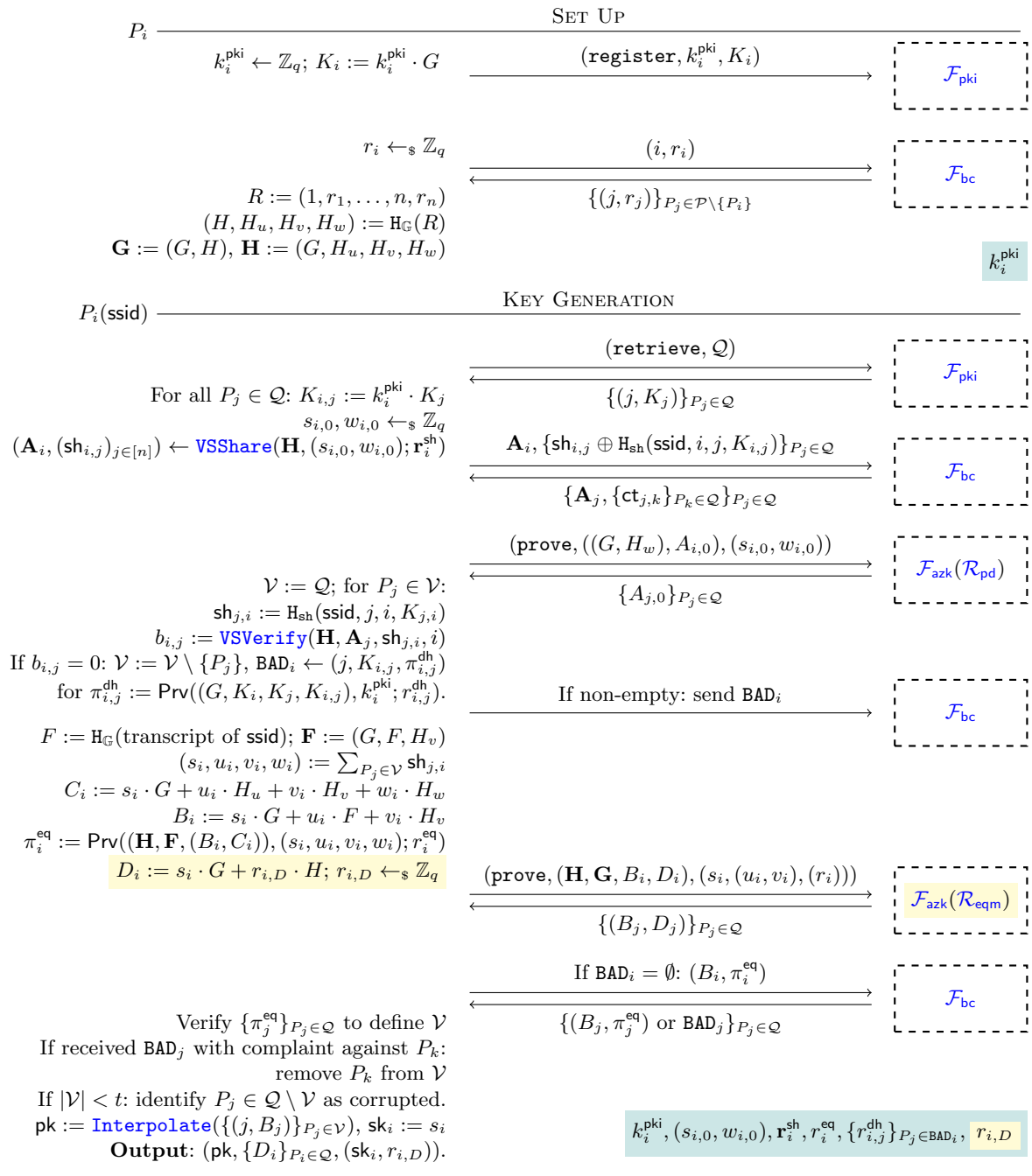
1.4.2 Intuition: Committed DKG Protocol Our committed DKG functionality $\mathcal{F}_{\mathsf{cdkg}}$ is essentially the same as our functionality $\mathcal{F}_{\mathsf{dkg}}$, except that the functionality additionally outputs independent random Pedersen commitments to each secret key share. This is to ensure that our DKG functionalities are useful in higher-level protocols; given commitments to the secret key shares parties can prove, e.g., the correctness of partial signatures, against these commitments.

Essentially, the protocol executes the second and third rounds of the three-round variant concurrently; parties *either* broadcast a complaint *or* their third round message. As this protocol is secure against a dishonest majority, i.e., has a full corruption threshold $t < n$, we must incorporate straightline extractable proofs via $\mathcal{F}_{\mathsf{azk}}$ to aid in simulation. In $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ our simulator can reconstruct the secrets of the corrupted party from the honest majority of shares in the VSS, but this is not possible for $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$.

[8] Given a DL-based adaptively secure approach to share distribution, security of our protocols would no longer have to rely on the DDH assumption.

Figure 2: Overview of $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}(\mathcal{G}, t)$.
Two Round Committed DKG with Identifiable Abort if $t < n$, $\mathcal{Q} = \mathcal{P}$, and $\mathcal{F}_{\mathsf{bc}}$ has bounded delay.

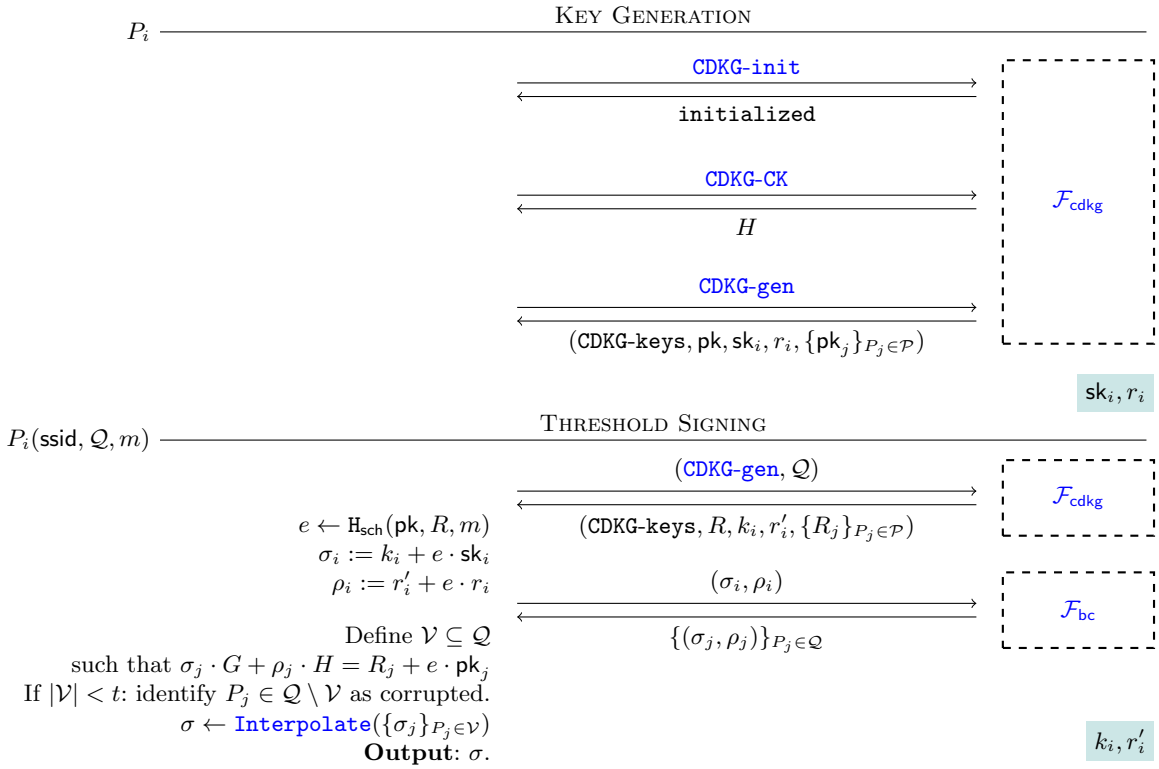
The protocol is executed by the n parties in $\mathcal{P}$ in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{ro}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model (Sec 2.1). Parameters are a group $\mathcal{G} = (\mathbb{G}, G, q)$, and the threshold $t < n$. The pair $(\mathtt{H}_{\mathbb{G}}, \mathtt{H}_{\mathtt{sh}})$ are random oracle instances. The pair $(\mathtt{VSShare}, \mathtt{VSVerify})$ is a VSS (Def 5). Proofs $\pi^{\mathsf{eq}}_i, \pi^{\mathsf{dh}}_{i,j}$ are NIZKs (Sec 3.1) for relations $\mathcal{R}_{\mathsf{eq}}, \mathcal{R}_{\mathsf{dh}}$. Functionality $\mathcal{F}_{\mathsf{azk}}$ proves statements include $\mathcal{R}_{\mathsf{eq}}, \mathcal{R}_{\mathsf{eqm}}$. Modifications for the committed property are highlighted. Party state contents are highlighted.



1.4.3 Intuition: Threshold Schnorr Our threshold Schnorr protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ is incredibly simple, and demonstrates the usefulness of $\mathcal{F}_{\mathsf{cdkg}}$. An overview of our protocol appears in Figure 3.

Figure 3: Overview of $\Pi^{\mathsf{ia}}_{\mathsf{sch}}(\mathcal{G}, t, \mathtt{H}_{\mathsf{sch}})$.
Three Round Threshold Schnorr Signing.

The protocol is executed by the n parties in $\mathcal{P}$ in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$-hybrid model, and is parameterized by $\mathcal{G} = (\mathbb{G}, G, q)$ a group $\mathbb{G}$ with generator G, an integer $t < n$, and the Schnorr Hash function $\mathtt{H}_{\mathsf{sch}} : \{0,1\}^* \to \mathbb{Z}_q$. Party state contents are highlighted.



2 Preliminaries

Notation. Protocols are executed by a set of parties $\mathcal{P}$. For ease, we say that if $\mathcal{P}$ is of size n then the identifiers of these parties cover the interval $[1, n]$. The protocol transcript is denoted as trans, and within each round the set of messages are ordered based on the sender identifiers instead of the order of receipt. The sets $\mathcal{C}$ and $\mathcal{H}$ denote the set of corrupted and honest parties, respectively.

Vectors $\mathbf{v} = (v_1, \ldots, v_k)$ are bold. We denote an inclusive range as $[a, b] := \{a, \ldots, b\}$. Assigning the variable x to take the value y is be written both left to right $x := y$ and right to left $y =: x$. Uniformly sampling x from a finite set is denoted $x \leftarrow_{\$} \mathcal{S}$. Computing x as the output of a randomized process is denoted $x \leftarrow \mathtt{ALGO}(\mathsf{inputs})$.

Groups are written additively. For a vector of scalars $\mathbf{e} = (e_1, \ldots, e_k) \in \mathbb{Z}_q^k$ and a vector of group elements $\mathbf{G} = (G_1, \ldots, G_k) \in \mathbb{G}^k$, we denote multi-multiplication by $\mathbf{e} \cdot \mathbf{G} = (e_1 G_1, \ldots, e_k G_k)$. We assume a generic group generation algorithm $\mathcal{G}(\lambda)$ that takes as input the security parameter and outputs public parameters $\mathcal{G} = (\mathbb{G}, G, q)$ that describe a cyclic group $\mathbb{G}$ of order q (with $||q|| = \lambda$) with generator $G \in \mathbb{G}$.

Helper Functions. Our constructions switch between equivalent representations of polynomials $P(X) \in \mathbb{Z}_q[X]$. The function `Coeff` takes as input a vector of polynomials and outputs vectors of coefficients (in a manner that are compatible with the description of our VSS algorithm, presented in Definition 5).

$$\texttt{Coeff}\left(\left(s_i + \sum_{k\in[1,t-1]} r_{i,k}X^k\right)_{i\in[\ell]}\right) \to \begin{pmatrix} \mathbf{s} = (s_i)_{i\in[\ell]}; \\ \mathbf{r} = (r_{i,1}, \ldots, r_{i,t-1})_{i\in[\ell]} \end{pmatrix}$$

Computational Assumptions. We recall the Discrete Logarithm and Decisional Diffie-Hellman Assumptions in Definitions 1 and Definition 2, respectively.

Definition 1 (Discrete Logarithm (DL) Assumption). *Let* GG *be a group sampling algorithm. The discrete logarithm assumption states that for all* $\lambda \in \mathbb{N}$*, for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[H = e \cdot G \;\middle|\; \begin{array}{r} \mathcal{G} := (\mathbb{G}, q) \leftarrow \mathsf{GG}(1^\lambda) \\ G, H \leftarrow_{\$} \mathbb{G} \\ e \leftarrow \mathcal{A}(\mathcal{G}, (G, H)) \end{array}\right] \leq \mathsf{negl}(\lambda). \tag{1}$$

If the DL assumption holds for GG *then we say that the DL problem is hard relative to* GG*.*

Definition 2 (Decisional Diffie-Hellman (DDH) Assumption). *Let* GG *be a group sampling algorithm. The Decisional Diffie-Hellman assumption is that for all* $\lambda \in \mathbb{N}$ *and for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that:*

$$\left|\frac{1}{2} - \Pr\left[b' = b \;\middle|\; \begin{array}{r} b \leftarrow_{\$} \{0,1\} \\ \mathcal{G} := (\mathbb{G}, G, q) \leftarrow \mathsf{GG} \\ x, y \leftarrow_{\$} \mathbb{Z}_q \\ \textit{if } b = 0 : z \leftarrow_{\$} \mathbb{Z}_q \\ \textit{if } b = 1 : z := x \cdot y \\ b' \leftarrow \mathcal{A}(\mathcal{G}, x \cdot G, y \cdot G, z \cdot G) \end{array}\right]\right| \leq \mathsf{negl}(\lambda).$$

If the DDH assumption holds for GG *then we say that the DDH problem is hard relative to* GG*.*

2.1 Model

Our protocols are written in the UC framework of Canetti [Can01]. Informally, in the UC framework a cryptographic task is modeled as an ideal functionality $\mathcal{F}$ that functions as a trusted third party which performs the computation on behalf of the set of parties with the secrets of the parties. Security is simulation-based; a protocol Π is said to UC-realize an ideal functionality $\mathcal{F}$ if for any real world adversary $\mathcal{A}$ against Π there exists a simulator $\mathcal{S}$ such that any environment $\mathcal{Z}$ cannot distinguish an execution in the ideal world with $(\mathcal{F}, \mathcal{S})$ from an execution in the real world with $(\Pi, \mathcal{A})$. Essentially, a proof of UC security proves that a protocol is *as good as* the ideal functionality and therefore any instaqnce of $\mathcal{F}$ can be replaced with an instance of Π.

Adaptive Corruptions in UC. On the adaptive corruption of an honest party $P_i \in \mathcal{P}$ the environment $\mathcal{Z}$ receives the secret state st_i of P_i. An adaptive corruption can occur at any point, and st_i must *explain* any output of P_i in the view of $\mathcal{Z}$. Specifically, st_i should contain the history of the input, output, work, and random tapes of P_i, and is sent to $\mathcal{Z}$ through $\mathcal{A}$. We informally describe the flow of information during an adaptive corruption, where all communication between $\mathcal{Z}$ and $\mathcal{A}^*$ in the ideal world, or $\mathcal{A}$ in the real world, is passed through a control function. In the real world, the environment $\mathcal{Z}$ issues a corruption instruction $(\texttt{corrupt}, i)$ to an honest party P_i through the real world adversary $\mathcal{A}$. The honest party P_i then outputs the read contents of each tape to $\mathcal{A}$. In the ideal world, the environment $\mathcal{Z}$ issues a corruption instruction $(\texttt{corrupt}, i)$ to an honest party P_i through the copy of $\mathcal{A}$ ran by the ideal world adversary $\mathcal{A}^*$ (the simulator $\mathcal{S}$). The ideal adversary issues the corruption instruction to the ideal functionality $\mathcal{F}$ and in response receives some information sec_i about the secret contents of st_i maintained by $\mathcal{F}$ on behalf of P_i. The remainder of st_i is computed by $\mathcal{A}^*$ before being returned to $\mathcal{Z}$ through $\mathcal{A}$. For simplicity, observe that to prove adaptive security it is sufficient for $\mathcal{S}$ to compute the state contents that are sufficient to recompute the entire state of the party.

There are some additional technicalities about adaptive corruption in UC. Primarily, we require consistency of corruption; it is required that the parties corrupted by $\mathcal{A}^*$ are exactly those corrupted by $\mathcal{Z}$, which can be enforced by, e.g., requiring that $\mathcal{F}$ output $\mathsf{corrupted}$ to a newly corrupted P_i in the ideal world, and adding to the protocol description that P_i similarly output $\mathsf{corrupted}$ once corrupted in the real world. This same issue arises when considering static adversaries. In the static case these machines should all agree on the set of corrupted parties prior to protocol execution. Similarly to the static case we assume that this agreement is enforced in some manner.

Output Guarantees in UC. A protocol satisfies guaranteed output delivery (GOD) if the adversary cannot prevent parties from computing output in a protocol execution. A protocol satisfies identifiable abort (IA) if the honest parties are guaranteed to identify a corrupted party in the event that the protocol does not deliver output. Modeling such output guarantees (GOD or IA) in UC is not straightforward. The UC model consists of a set of interacting Turing machines (ITMs), and is inherently single-threaded; only a single ITM is running at any time, and this ITM it can pass activation on to other parties. When honest parties activate the functionality with their input the functionality activates the simulator (to simulate the corrupt parties' views). The simulator may never return activation to the functionality, in which case the functionality does not provide output to the honest parties. Formally proving, e.g., GOD, requires proving that the simulator always returns activation to the functionality.

Katz, Maurer, Tackmann and Zikas [KMTZ13] provide an alternative approach to modeling synchronous termination guarantees in UC by requiring the functionality to deliver output if queried by the honest parties enough times. Their approach is notationally dense and we believe it could distract the reader from our contributions. To provide output guarantees we take the pedestrian approach; we prove that the simulator always activates the functionality to provide output if the underlying broadcast channel receives enough activations to guarantee that it produces output. While simple and sound, this methodology has one downside: not all properties of our protocols are reflected in the ideal functionality.

Hybrid Functionalities. Each of our protocols assumes that parties have ideal access to a subset of the following functionalities.

$\mathcal{F}_{\mathsf{azk}}$. Our adaptive zero knowledge functionality is detailed in Section 3.2.

$\mathcal{F}_{\mathsf{bc}}$. The broadcast functionality $\mathcal{F}_{\mathsf{bc}}$ has a single interface. The output guarantees of our protocols are proven under the condition that $\mathcal{F}_{\mathsf{bc}}$ guarantees the delivery of all messages, meaning that the channel has a bounded delay (see Output Guarantees in UC), but this is not required for security in general.

- $\texttt{broadcast}$: This query takes as input a message m, leaks m to $\mathcal{A}^*$, then outputs $(\texttt{broadcast}, i, m)$ to $\mathcal{P}$ at the adversary's discretion.

$\mathcal{F}_{\mathsf{pki}}$. The public key infrastructure (PKI) functionality models a PKI in which each party must prove knowledge of their secret key to register. The functionality has two interfaces:

– `PKI-register`: Key registration can be queried a single time by any $P_i \in \mathcal{P}$. The query includes a party index such that $P_i \in \mathcal{P}$ the public key pk_i and the secret key k_i^{pki}. The functionality records (P_i, pk_i).

– `PKI-fetch`: Key retreival can be queried by any $P \in \mathcal{P}$ takes as input a set of parties $\mathcal{Q} \subseteq \mathcal{P}$, and outputs any recorded keys for the parties in the set.

$\mathcal{F}_{\mathsf{ro}}$**.** The programmable random oracle functionality is parameterized by an output space $\mathcal{Y}$ and has a single interface. To ease notation the instance $\mathcal{F}_{\mathsf{ro}}(\mathcal{Y})$ is denoted $\mathtt{H}_{\mathcal{Y}}$.

– `get-RAND`: This query takes as input x and outputs a consistent, uniformly random output y, sampled on the first query of x.

2.2 Secret Sharing

A common and useful tool in DKG protocols is verifiable secret sharing (VSS). We recall the relevant concepts in this section.

Commitments. The pedersen commitment scheme [Ped92] in a cyclic group $\mathbb{G}$ of prime order q for messages $\mathbf{e} \in \mathbb{Z}_q^\ell$ of length ℓ is presented in Definition 3. Pedersen's commitment scheme is both *perfectly hiding*; a computationally unbounded attacker cannot learn anything about the message $\mathbf{e}$ given $(\mathbf{H}, \mathtt{Commit}(\mathbf{H}, \mathbf{e}))$. The scheme is *computationally binding* under the DL assumption in $\mathbb{G}$; any PPT adversary cannot output two distinct messages $\mathbf{e}, \mathbf{e}' \in \mathbb{Z}_q^\ell$ such that $\mathtt{Commit}(\mathbf{H}, \mathbf{e}) = \mathtt{Commit}(\mathbf{H}, \mathbf{e}')$.

Definition 3 (Pedersen Commitment). *The pedersen commitment scheme in a group $\mathbb{G}$ of order q for ℓ messages in $\mathbb{Z}_q$ is described by the pair of algorithms below.*

$\mathtt{Commit}(\mathbf{H}, \mathbf{e}) \to C$**.** *On inputs $\mathbf{H} \in \mathbb{G}^\ell$ and $\mathbf{e} \in \mathbb{Z}_q^\ell$, output $C := \mathbf{e} \cdot \mathbf{H}$.*

$\mathtt{ComVerify}(\mathbf{H}, \mathbf{e}, C) \to \{0,1\}$**.** *On inputs $\mathbf{H} \in \mathbb{G}^\ell$, $\mathbf{e} \in \mathbb{Z}_q^\ell$, and $C \in \mathbb{G}$, output 1 iff $C = \mathbf{e} \cdot \mathbf{H}$.*

Shamir Secret Sharing. Shamir secret sharing [Sha79] to (t, n)-threshold share secrets in $\mathbb{Z}_q$ is presented in Definition 4. The secret sharing scheme is *perfectly secret*, meaning that a computationally unbounded attacker cannot learn anything about the secret s given $t-1$ or fewer shares output by $\mathtt{SShare}(s; \mathbf{r})$.

Definition 4 (Shamir Secret Sharing). *The Shamir secret sharing scheme for secrets $s \in \mathbb{Z}_q$, parameterized by the number of parties $n \in \mathbb{N}$ and threshold $t \leq n$, is described by the pair of algorithms below.*

$\mathtt{SShare}(s; \mathbf{r}) \to \mathsf{sh}$**.** *On inputs $s \in \mathbb{Z}_q$, $(r_1, \ldots, r_{t-1}) = \mathbf{r} \in \mathbb{Z}_q^{t-1}$ define the degree $t-1$ polynomial*

$$S(X) := s + \sum_{k=1}^{t-1} r_k X^k$$

and output $(S(1), \ldots, S(n)) \in \mathbb{Z}_q^n$.

$\mathtt{SReconstruct}((i_1, \mathsf{sh}_{i_1}), \ldots, (i_t, \mathsf{sh}_{i_t})) \to s$**.** *On input $i_j \in [1, n]$ and $\mathsf{sh}_{i,j} \in \mathbb{Z}_q$ for $j \in [1, t]$, compute $S(X) \leftarrow \mathtt{Interpolate}(t-1, \{(i_j, \mathsf{sh}_{i_j})\}_{j \in [t]})$. Output $S(0)$.*

The function `Interpolate` computes Lagrange interpolation. The input is a set of at least t pairs, and the output is a polynomial of degree $t-1$ that interpolates these points. The parameter t is ommitted when clear from context; we do not pass t to the function explicitly.

$$\mathtt{Interpolate}(\{(x_i, y_i)\}_{i \in [d]}) \to P(X) := \sum_{i \in [d]} \left(\prod_{j \in [d] \setminus \{i\}} \frac{x_j - X}{x_j - x_i} \right) y_i.$$

Verifiable Secret Sharing. The Pedersen verifiable secret sharing (VSS) scheme [Ped92] in a group $\mathbb{G}$ of order q to (t, n)-threshold share secrets $\mathbf{s} \in \mathbb{Z}_q^\ell$ is presented in Definition 5. Pedersen VSS satisfies *perfect secrecy*, meaning that a computationally unbounded attacker cannot learn anything about the secret $\mathbf{s}$ given any $t-1$ or fewer secrets and the public commitments $\mathbf{C}$; there exists a simulator $\mathcal{S}$ that, on input C_0 and $t-1$ shares $\{(i_j, \mathsf{sh}_{i_j})\}_{j\in[t-1]}$ can compute $(C_1, \ldots, C_{t-1})$ such that $\mathbf{C} = (C_0, \ldots, C_{t-1})$ has the same distribution as the output of $\mathtt{VSShare}$ and $1 = \mathtt{VSVerify}(\mathbf{H}, \mathbf{C}, \mathsf{sh}_{i_j}, i_j)$ for all $j \in [1, t-1]$.

Definition 5 (Pedersen Verifiable Secret Sharing). *The Pedersen verifiable secret sharing scheme for ℓ secrets $\mathbf{s} \in \mathbb{Z}_q^\ell$, parameterized by the number of parties $n \in \mathbb{N}$ and threshold $t \le n$, is described by the pair of algorithms below.*

$\mathtt{VSShare}(\mathbf{H}, \mathbf{s}; \mathbf{r}) \to (\mathsf{sh}, \mathbf{C})$**.** *On input $\mathbf{H} \in \mathbb{G}^\ell$, $\mathbf{s} \in \mathbb{Z}_q^\ell$, and randomness $((r_{1,1}, \ldots, r_{1,t-1}), \ldots, (r_{\ell,1}, \ldots, r_{\ell,t-1})) = \mathbf{r} \in \mathbb{Z}_q^{\ell\times(t-1)}$:*

- *Commit $C_0 := \mathtt{Commit}(\mathbf{H}, \mathbf{s})$,*
- *Commit $C_k := \mathtt{Commit}(\mathbf{H}, (r_{1,k}, \ldots, r_{\ell,k}))$ for $k \in [1, t-1]$, and*
- *Share $\mathsf{sh}_j \leftarrow \mathtt{SShare}(s_j; (r_{j,1}, \ldots, r_{j,t-1}))$ for $j \in [1, \ell]$.*

Output $\mathsf{sh} := (\mathsf{sh}_1, \ldots, \mathsf{sh}_\ell)$ and $\mathbf{C} := (C_0, C_1, \ldots, C_{t-1})$.

$\mathtt{VSVerify}(\mathbf{H}, \mathbf{C}, \mathsf{sh}, i) \to \{0, 1\}$**.** *On input $\mathbf{H} \in \mathbb{G}^\ell$, $\mathbf{C} \in \mathbb{G}^t$, $\mathsf{sh} \in \mathbb{Z}_q^\ell$, and $i \in [1, n]$ output 1 iff $\mathtt{Commit}(\mathbf{H}, \mathsf{sh}) = C_0 + \sum_{k=1}^{t-1}(i^k) \cdot C_k$.*

The secret s_j at any index $j \in [1, \ell]$ can be reconstructed from any t verifying shares $\{(i_j, \mathsf{sh}_{i_j})\}_{j\in[1,t]}$ via $\mathtt{SReconstruct}$.

Pedersen VSS is also aggregatable; $\mathtt{AggSh}$ below takes as input the vectors $\{\mathsf{sh}_i\}_i$, $\mathsf{sh}_i = (\mathsf{sh}_{i,1}, \ldots, \mathsf{sh}_{i,\ell}) \in \mathbb{Z}_q^\ell$ and outputs the element-wise sum of the share vectors:

$$\mathtt{AggSh}(\mathsf{sh}_1, \ldots, \mathsf{sh}_m) \to \hat{\mathsf{sh}} := (\hat{\mathsf{sh}}_1, \ldots, \hat{\mathsf{sh}}_\ell) \in \mathbb{Z}_q^\ell \text{ where : } \hat{\mathsf{sh}}_i := \sum_{j\in[m]} \mathsf{sh}_{j,i}.$$

3 Adaptive Zero Knowledge

We frame Section 3 as a reinterpretation of existing results on (adaptive) zero knowledge and do not claim novelty. In Section 3.1 we present stand alone definitions for adaptive NIZKPoKs, and demonstrate proof systems that satisfy the required properties in Figure 4. In Section 3.2 we present a simple functionality $\mathcal{F}_{\mathsf{azk}}$ for adaptive zero knowledge, and UC-realize this protocol from an existing functionality $\mathcal{F}_{\mathsf{anizk}}$ conditioned on a "well behaved" ideal adversary for $\mathcal{F}_{\mathsf{anizk}}$ [LR22]. This result implies a UC-secure protocol that realizes $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eq}})$ and $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$.

Group-based Relations of Interest. Our protocols rely on non-interactive zero knowledge proofs for malicious security. The relevant relations are those listed below, all parameterized by an additive group $\mathcal{G} = (\mathbb{G}, q)$:

$$\begin{aligned}
\mathcal{R}_{\mathsf{dh}} &:= \left\{ \left((G, E, H, F), e\right) \;\middle|\; \begin{matrix} E, F \in \mathbb{G} \\ E = e \cdot G \;\wedge\; F = e \cdot H \end{matrix} \right\} \\
\mathcal{R}_{\mathsf{pd}} &:= \left\{ \left((\mathbf{G}, E), \mathbf{e}\right) \;\middle|\; \begin{matrix} (\mathbf{G}, E) \in \mathbb{G}^* \\ E = \mathbf{e} \cdot \mathbf{G} \end{matrix} \right\} \\
\mathcal{R}_{\mathsf{eq}} &:= \left\{ \left((\mathbf{G}, \mathbf{H}, A, B), \mathbf{e}\right) \;\middle|\; \begin{matrix} (\mathbf{G}, A), (\mathbf{H}, B) \in \mathbb{G}^* \\ A = \mathbf{e} \cdot \mathbf{G} \;\wedge\; B = \mathbf{e} \cdot \mathbf{H} \end{matrix} \right\} \\
\mathcal{R}_{\mathsf{eqm}} &:= \left\{ \left(\begin{matrix} ((G, \mathbf{G}), (G, \mathbf{H}), A, B), \\ (s, \mathbf{e}, \mathbf{e}') \end{matrix} \right) \;\middle|\; \begin{matrix} G, (\mathbf{G}, A), (\mathbf{H}, B) \in \mathbb{G}^* \\ A = sG + \mathbf{e} \cdot \mathbf{G} \;\wedge\; B = sG + \mathbf{e}' \cdot \mathbf{H} \end{matrix} \right\}
\end{aligned} \tag{2}$$

Looking ahead, our protocols make use of a standard NIZK (without requiring adaptivity) for $\mathcal{R}_{\mathsf{dh}}$, but require adaptivity for proofs of the relations $\mathcal{R}_{\mathsf{pd}}, \mathcal{R}_{\mathsf{eq}}, \mathcal{R}_{\mathsf{eqm}}$. The notions of adaptivity that we consider are clarified in the subsequent sections. Further, all proofs that are required to be straightline extractable are proven via the adaptive zero knowledge functionality $\mathcal{F}_{\mathsf{azk}}$ in Figure 5.

3.1 Adaptive Non-Interactive Zero Knowledge

We recall the definition of a standard Σ-protocol below. The randomness used to compute protocol messages is explicit, as we are concerned with adaptive security.

Definition 6 (Σ-protocol). *A Σ-protocol for an NP relation $\mathcal{R}$ is a 3-move public coin interactive protocol between a probabilistic prover P_Σ and verifier V_Σ that can be described as a tuple of PT algorithms with the following syntax:*

$\Sigma.\mathtt{Commit}(\phi, w; a) \to (A, \mathsf{st}_P)$. *On input a statement ϕ, a witness w and randomness a outputs a commitment A, and a state st_P.*

$\Sigma.\mathtt{Challenge}(\phi, A; r_V) \to c$. *On input a statement ϕ and a commitment A outputs a challenge c.*

$\Sigma.\mathtt{Respond}(\mathsf{st}_P, c) \to z$. *On input a state st_P and a challenge c outputs a response z.*

$\Sigma.\mathtt{Verify}(\phi, A, c, z) \to \{0,1\}$. *On input a statement ϕ and a transcript (A, c, z) outputs 1 for accept and 0 for reject.*

A Σ-protocol must satisfy the properties of completeness, special honest-verifier zero-knowledge and special soundness below.

Completeness. *For all $(\phi, w) \in \mathcal{R}$ and all $\lambda \in \mathbb{N}$:*

$$\Pr\left[\Sigma.\mathtt{Verify}(\phi,(A,c,z))=1 \;\middle|\; \begin{array}{r} a \leftarrow_{\$} \{0,1\}^\lambda \\ (A,\mathsf{st}) \leftarrow \Sigma.\mathtt{Commit}(\phi,w;a) \\ c \leftarrow \Sigma.\mathtt{Challenge}(\phi,A,c) \\ z \leftarrow \Sigma.\mathtt{Respond}(\mathsf{st},c) \end{array}\right] = 1.$$

Special Honest-Verifier Zero-Knowledge. *There exists a PPT simulator $\mathcal{S}$ such that for all $\lambda \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$, and for all $(\phi, w) \in \mathcal{R}$:*

$$\left|\frac{1}{2} - \Pr\left[b'=b \;\middle|\; \begin{array}{r} b \leftarrow_{\$} \{0,1\} \\ a \leftarrow_{\$} \{0,1\}^\lambda \\ (A,\mathsf{st}) \leftarrow \Sigma.\mathtt{Commit}(\phi,w;a) \\ c \leftarrow \Sigma.\mathtt{Challenge}(\phi,A,c) \\ z \leftarrow \Sigma.\mathtt{Respond}(\mathsf{st},c) \\ \pi_0 \leftarrow \mathcal{S}(1^\lambda,\phi); \pi_1 := (A,c,z) \\ b' \leftarrow \mathcal{A}(1^\lambda,\phi,\pi_b) \end{array}\right]\right| = 1 - \mathsf{negl}(\lambda).$$

Special Soundness. *There exists a PPT extractor $\mathcal{E}$ such that for all $\lambda \in \mathbb{N}$, all statements ϕ and pairs of transcripts $\pi = (A, c, z)$ and $\pi' = (A', c', z')$ such that $c \neq c'$ and $V_\Sigma(\phi, \pi) = V_\Sigma(\phi, \pi') = 1$:*

$$\Pr\left[(\phi,w) \in \mathcal{R} \middle| w \leftarrow \mathcal{E}(1^\lambda, \phi, \pi, \pi')\right] = 1 - \mathsf{negl}(\lambda).$$

A standard Σ-protocol can be compiled into a non-interactive zero knowledge proof (NIZK, Definition 7) via the Fiat-Shamir transform [FS87] in the random oracle model. Essentially, for oracle $\mathtt{H}_{\mathsf{fs}}$ the FS-transform replaces the process $\Sigma.\mathtt{Challenge}$ executed by honest V_Σ with the random oracle: $c \leftarrow \mathtt{H}_{\mathsf{fs}}(\phi, A)$.

Definition 7 (Non-Interactive Zero Knowledge (NIZK)). *A non-interactive zero knowledge proof in the random oracle model for the relation $\mathcal{R}$ is a pair of PT algorithms $(\mathsf{Prv}, \mathsf{Ver})$, the latter deterministic, with the following syntax:*

$\mathsf{Prv}^{\mathtt{H}_{\mathsf{fs}}}(\phi, w; r) \to \pi$. *On input a statement ϕ, a witness w and randomness r outputs a string π.*

$\mathsf{Ver}^{\mathtt{H}_{\mathsf{fs}}}(\phi, \pi) \to \{0,1\}$. *On input a statement ϕ and a string π outputs 1 for accept and 0 for reject.*

Any NIZK must satisfy completeness, along with both zero knowledge and soundness properties. We define the properties of NIZKs individually or modularity; our protocols require different strengths of soundness for different relations. Completeness (Definition 8) is a correctness property that guarantees the verifier will accept proofs output by the prover with overwhelming probability.

Definition 8 (Overwhelming Completeness). *A NIZK for the relation $\mathcal{R}$ satisfies overwhelming completeness if for any oracle $\mathtt{H}$, for all $\lambda \in \mathbb{N}$, for all $(\phi, w) \in \mathcal{R}$, there exists a negligible function negl such that:*

$$\Pr\left[\mathsf{Ver}^{\mathtt{H}}(\phi,\pi) = 1 \,\middle|\, \begin{array}{r} r \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathsf{Prv}^{\mathtt{H}}(\phi, w; r) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

For security against adaptive adversaries the standard zero knowledge property, i.e., the existence of an efficient simulator that, without a witness, can output proofs that are computationally indistinguishable from those output by an honest prover, is insufficient. This is because on the adaptive corruption of a simulated prover the zero knowledge simulator is not guaranteed to be able to *explain* every simulated proof by producing randomness that can be used to recompute the proof with the honest algorithm. In the adaptive corruption setting the zero knowledge simulator must be able to produce this randomness for any witness *after* simulating a proof. We present the standard definition for zero knowledge in the random oracle model in Definition 9, and highlight the additional requirements for security against adaptive corruption of a simulated prover. Note that our definition allows the adversary to adaptively select the pair (ϕ, w) with access to the oracle $\mathtt{H}_b$.

Definition 9 (Zero Knowledge with Adaptive Prover Corruption (ZK, aZK)). *A NIZK $(\mathsf{Prv}, \mathsf{Ver})$ for the relation $\mathcal{R}$ satisfies* **zero knowledge with adaptive prover corruption** *if for all $\lambda \in \mathbb{N}$ there exists a PPT simulator $\mathcal{S}_{\mathsf{azk}} = (\mathcal{S}_{\mathsf{SU}}, \mathcal{S}_{\mathsf{P}}, \mathcal{S}_{\mathsf{R}})$, for all pairs of PPT algorithms $(D_{\mathsf{SU}}, D_{\mathsf{P}})$ there exists a negligible function negl such that:*

$$\left|\frac{1}{2} - \Pr\left[b = b' \,\middle|\, \begin{array}{r} b \leftarrow_{\$} \{0,1\} \\ (\mathtt{H}_0, \mathsf{st}_S) \leftarrow \mathcal{S}_{\mathsf{SU}}(1^\lambda); \mathtt{H}_1 := \mathtt{H} \\ (\phi, w, \mathsf{st}_D) \leftarrow D_{\mathsf{SU}}^{\mathtt{H}_b}(1^\lambda) \\ v := ((\phi, w) \in \mathcal{R}) \\ \pi_1 \leftarrow \mathsf{Prv}^{\mathtt{H}_1}(\phi, w; r_1);\ r_1 \leftarrow_{\$} \{0,1\}^\lambda \\ (\mathsf{st}_P, \mathtt{H}'_0, \pi_0) \leftarrow \mathcal{S}_{\mathsf{P}}(\mathsf{st}_S, \phi, v); \mathtt{H}'_1 := \mathtt{H}_1 \\ r_0 \leftarrow \mathcal{S}_{\mathsf{R}}^{\mathtt{H}'_0}(\mathsf{st}_S, \mathsf{st}_P, w) \\ b' \leftarrow D_{\mathsf{P}}^{\mathtt{H}'_b}(\mathsf{st}_D, \pi_b, r_b) \end{array}\right]\right| = \mathsf{negl}(\lambda),$$

where $\mathtt{H}$ is a random oracle.
A NIZK satisfies **zero knowledge** *if the above holds without the highlighted text.*

For security against adaptive adversaries the standard soundness properties suffice. In Definition 10 we recall the definitions of soundness, simulation soundness and simulation extractability for a NIZK. Note that these definitions are adaptive in the sense that the adversary $\mathcal{A}$ can similarly select the pair (ϕ, π) based on its access to the oracle $\mathtt{H}$.

Definition 10 (Soundness, Simulation Soundness, and Simulation Extractability (S, SS, SE)). *A NIZK* $(\mathsf{Prv}, \mathsf{Ver})$ *for the relation* $\mathcal{R}$ *satisfies* **simulation soundness** *(SS) if there exists an extractor* $\mathcal{E}$ *that, for any security parameter* $\lambda \in \mathbb{N}$ *and any PPT adversary* $\mathcal{A}$ *there exists a negligble function* negl *such that:*

$$\Pr\left[\begin{array}{c|c} (\phi, w) \in \mathcal{R} & \mathcal{Q} := \emptyset \\ \wedge & (\mathtt{H}, \mathsf{st}_S) \leftarrow \mathcal{S}_{\mathsf{SU}}(1^\lambda) \\ (\phi, \pi) \notin \mathcal{Q} & (\phi, \pi) \leftarrow \mathcal{A}^{\mathtt{H}, \mathcal{O}_{\mathcal{S}_\mathsf{P}}(\cdot,\cdot)}(1^\lambda) \\ \wedge & w \leftarrow \mathcal{E}^{\mathtt{H}, \mathcal{A}}(\mathsf{st}_S, \phi, \pi) \\ \mathsf{Ver}(\phi, \pi) = 1 & \end{array}\right] = \mathsf{negl}(\lambda),$$

where $\mathtt{H}$ *is a random oracle programmable by* $\mathcal{E}$*.*

Oracle $\mathcal{O}_{\mathcal{S}_\mathsf{P}}(\phi, w) \to \pi$*: if* $(\phi, w) \notin \mathcal{R}$ *the oracle computes* $(\mathsf{st}_P, \mathtt{H}, \pi) \leftarrow \mathcal{S}_\mathsf{P}(\mathsf{st}_S, \mathsf{st}_P, \phi, 1)$ *adds* $(\phi, \pi) \to \mathcal{Q}$ *and outputs* π*, and otherwise outputs* $\perp$*.*

A NIZK satisfies **soundness** *(S) if* $\mathcal{A}$ *is not given oracle access to simulated proofs via* $\mathcal{O}_{\mathcal{S}_\mathsf{P}}$*.*

A NIZK satisfies **simulation-extractability** *(SE) if* $\mathcal{E}$ *is required to be efficient.*

FS-transformed Σ-protocols have been proven to satisfy completeness, zero knowledge, and each of the soundness notions in Definition 10 [FKMV12]. We present in Figure 4 FS-transformed Σ-protocols $(\mathsf{Prv}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}, \mathsf{Ver}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}})$ and $(\mathsf{Prv}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}, \mathsf{Ver}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}})$ for the relations $\mathcal{R}_{\mathsf{pd}}$ and $\mathcal{R}_{\mathsf{eq}}$, which then satisfy Definitions 8, 10, and zero knowledge as in Definition 9. We present the relevant *randomness simulators* $(\mathcal{S}_{\mathsf{pd}}^{\mathsf{R}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}})$ in this figure as well, thereby demonstrating that these NIZKs satisfy zero knowledge with adaptive prover corruption. As the triple $(\phi, \pi, \mathbf{e})$ uniquely determines the output, Definition 9 is trivially satisfied.

Figure 4: $(\mathsf{Prv}, \mathsf{Ver}, \mathcal{S}^{\mathsf{R}})$ **for Relations** $\mathcal{R}_{\mathsf{pd}}$ **and** $\mathcal{R}_{\mathsf{eq}}$**.**
FS-transformed Σ*-Protocols.*

We demonstrate the adaptive zero knowledge of the following otherwise standard FS-compiled NIZKPoKs for $\mathcal{R}_{\mathsf{pd}}$ and $\mathcal{R}_{\mathsf{eq}}$ as in Eq 2 by construction. In the below:

– $\mathcal{G} = (\mathbb{G}, G, q)$ is a group $\mathbb{G}$ of order q with generator G,

– $\mathtt{H}_{\mathsf{pd}}, \mathtt{H}_{\mathsf{eq}} : \{0,1\} \to \mathbb{Z}_q$ are random oracles; the input/output tables are passed to $\mathcal{S}$.

Pedersen Commitment Opening $\mathcal{R}_{\mathsf{pd}}$**:**

$\underline{\mathsf{Prv}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}(\phi, \mathbf{e}; r_{\mathbf{a}})}$:
$(\mathbf{G}, E) := \phi$
$d := \mathsf{dim}(\mathbf{G})$
$A := \mathbf{a} \cdot \mathbf{G}$; $\mathbf{a} := r_{\mathbf{a}} \in \mathbb{Z}_q^d$
$c := \mathtt{H}_{\mathsf{pd}}((\mathbf{G}, E), A)$
$\mathbf{z} := \mathbf{a} + c \cdot \mathbf{e}$
$\pi := (A, \mathbf{z})$
Return π

$\underline{\mathsf{Ver}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}(\phi, \pi)}$:
$(\mathbf{G}, E) := \phi$
$(A, \mathbf{z}) := \pi$
$c := \mathtt{H}_{\mathsf{pd}}(\phi, A)$
Return 0 **if:**
$\mathbf{z} \cdot \mathbf{G} \neq A + c \cdot E$
Return 1.

$\underline{\mathcal{S}_{\mathsf{pd}}^{\mathsf{R}}(\mathtt{H}_{\mathsf{pd}}, \phi, \pi, \mathbf{e})}$:
Return $\perp$ **if:**
$1 \neq \mathsf{Ver}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}((\mathbf{G}, E), \pi)$
$(\mathbf{G}, E) := \phi$
$(A, \mathbf{z}) := \pi$
$c := \mathtt{H}_{\mathsf{pd}}(\phi, A)$
$\mathbf{a} := \mathbf{z} - c \cdot \mathbf{e}$
Return a.

Equal Pedersen Openings $\mathcal{R}_{\mathsf{eq}}$:

$\underline{\mathsf{Prv}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}}(\phi, \mathbf{e}; r_{\mathbf{a}})}$:

$(\mathbf{G}, \mathbf{H}, E, F) := \phi$
$d := \mathsf{dim}(\mathbf{G})$
$(\mathbf{a}_G, \mathbf{a}_H) := r_{\mathbf{a}} \in \mathbb{Z}_q^{2d}$
$A_E := \mathbf{a}_E \cdot \mathbf{G}$
$A_F := \mathbf{a}_F \cdot \mathbf{H}$
$c := \mathtt{H}_{\mathsf{eq}}(\phi, A_E, A_F)$
$\mathbf{z}_E := \mathbf{a}_E + c \cdot \mathbf{e}$
$\mathbf{z}_F := \mathbf{a}_F + c \cdot \mathbf{e}$
$\pi := (A_E, A_F, \mathbf{z}_E, \mathbf{z}_F)$
Return π.

$\underline{\mathsf{Ver}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}}(\phi, \pi)}$:

$(\mathbf{G}, \mathbf{H}, E, F) := \phi$
$(A_E, A_F, \mathbf{z}_E, \mathbf{z}_F) := \pi$
$c := \mathtt{H}_{\mathsf{eq}}(\phi, A_E, A_F)$
Return 0 **if:**
$\mathbf{z}_E \cdot \mathbf{G} \neq A_E + c \cdot E$
$\mathbf{z}_F \cdot \mathbf{H} \neq A_F + c \cdot F$
Return 1.

$\underline{\mathcal{S}^{\mathsf{eq}}_{\mathsf{R}}(\mathtt{H}_{\mathsf{eq}}, \phi, \pi, \mathbf{e})}$:

Return $\perp$ **if:**
$1 \neq \mathsf{Ver}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}}(\phi, \pi)$
$(\mathbf{G}, \mathbf{H}, E, F) := \phi$
$(A_E, A_F, \mathbf{z}_E, \mathbf{z}_F) := \pi$
$c := \mathtt{H}_{\mathsf{eq}}(\phi, A_E, A_F)$
$\mathbf{a}_E := \mathbf{z}_E - c \cdot \mathbf{e}$
$\mathbf{a}_F := \mathbf{z}_F - c \cdot \mathbf{e}$
Return $(\mathbf{a}_E, \mathbf{a}_F)$.

3.2 Universally Composable Adaptive Zero Knowledge

Adaptive Zero Knowledge Functionality. We present a simple adaptive zero knowledge proof of knowledge functionality $\mathcal{F}_{\mathsf{azk}}$ in Figure 5.

Figure 5: $\mathcal{F}_{\mathsf{azk}}(\mathcal{R})$.
Adaptive Zero Knowledge Functionality.

The adaptive non-interactive zero knowledge proof of knowledge functionality is parameterized by an NP relation $\mathcal{R}$. The functionality interacts with the parties in $\mathcal{P}$ and the ideal adversary $\mathcal{A}^*$ via the queries below, all of which implicitly include the session identifier sid. Upon invocation the functionality receives a set of corrupt parties $\mathcal{C} \subseteq \mathcal{P}$, and initializes an empty table $\mathtt{wits}[][]$.

On $(\mathtt{ZK\text{-}prove}, \mathsf{ssid}, \phi, w)$ **from** $P_i \in \mathcal{P}$**.** Ignore if $(\phi, w) \notin \mathcal{R}$. If $P_i \notin \mathcal{C}$: send $(\mathtt{Prover}, \mathsf{ssid}, P_i, \phi)$ to $\mathcal{A}^*$, and await $(\mathtt{ZK\text{-}deliver}, \mathsf{ssid}, P_i)$ from $\mathcal{A}^*$. Set $\mathtt{wits}[i][\mathsf{ssid}] := (\phi, w)$ and send $(\mathtt{ZK\text{-}proven}, \mathsf{ssid}, P_i, \phi)$ to $\mathcal{P}$.

On $(\mathtt{ZK\text{-}corrupt}, P_i)$ **from** $\mathcal{A}^*$**.** Add $P_i \to \mathcal{C}$. Send $(\mathtt{ZK\text{-}corrupt}, P_i, \mathtt{wits}[i][:])$ to $\mathcal{A}^*$.

Adaptive Non-Interactive Zero Knowledge Functionality. In the UC framework a NIZK *proof of knowledge* (NIZKPoK) must be straight-line extractable. A Σ-protocol can be compiled into a (black-box) straight-line extractable NIZKPoK via the randomized Fischlin transform [Fis05; Ks22]. The work of Lysyanskaya and Rosenbloom [LR22], from which many of our definitions were adapted, demonstrates how to obtain an efficient and adaptive UC-secure NIZKPoK from any Σ-protocol satisfying adaptivity in the global random oracle model. We recall their functionality $\mathcal{F}_{\mathsf{anizk}}$ in Figure 6.

Figure 6: $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
Adaptive NIZK Proof of Knowledge Functionality.

The adaptive non-interactive zero knowledge proof of knowledge functionality is parameterized by an NP relation $\mathcal{R}$. The functionality interacts with the parties in $\mathcal{P}$ and the ideal adversary $\mathcal{A}^*$

via the queries below, all of which implicitly include the session identifier sid. Upon invocation the functionality receives the set of corrupted parties $\mathcal{C} \subseteq \mathcal{P}$, and initializes an empty table $\texttt{proofs}[]$.

On $(\texttt{NIZK-init})$ **from** $P \in \mathcal{P}$**.** Ignore the query if there is a record $(\texttt{init}, [\cdots])$. Send $(\texttt{NIZK-init})$ to $\mathcal{A}^*$. Receive $(\texttt{NIZK-algo}, \mathsf{SetUp}, \mathsf{Prv}, \mathsf{Ver}, \mathcal{S}.\mathsf{SetUp}, \mathcal{S}.\mathsf{Prv}, \mathcal{E})$ from $\mathcal{A}^*$ and record the tuple. Execute $(\mathsf{pp}, \tau_{\mathcal{S}}) \leftarrow \mathcal{S}.\mathsf{SetUp}$ and record $(\texttt{init}, \mathsf{pp}, \tau_{\mathcal{S}})$. Output $\texttt{NIZK-initialized}$ to P.

On $(\texttt{NIZK-prove}, \phi, w)$ **from** $P_i \in \mathcal{P}$**.** Ignore the query if $(\phi, w) \notin \mathcal{R}$ or if there is no record $(\texttt{init}, [\cdots])$. Compute $(\pi, \tau_\pi) \leftarrow \mathcal{S}.\mathsf{Prv}(\phi)$ and output $\texttt{NIZK-fail}$ if $\mathsf{Ver}(\phi, \pi) \neq 1$. Add $\texttt{proofs}[i] \leftarrow (\tau_{\mathcal{S}}, \phi, w, \pi, \tau_\pi)$ and output $(\texttt{NIZK-proof}, \phi, \pi)$ to P_i.

On $(\texttt{NIZK-verify}, \phi, \pi)$ **from** $P_i \in \mathcal{P}$**.** If there is some $P_j \in \mathcal{P}$ such that $(\cdot, \phi, \cdot, \pi, \cdot) \in \texttt{proofs}[j]$ set $b := 1$. Else if $\mathsf{Ver}(\phi, \pi) = 0$ set $b := 0$. Else, compute $w \leftarrow \mathcal{E}(\phi, \pi)$ and if $(\phi, w) \in \mathcal{R}$ set $b := 1$, and otherwise output $\texttt{NIZK-fail}$. Output $(\texttt{NIZK-verify}, \phi, \pi, b)$ to P_i.

On $(\texttt{NIZK-corrupt}, i)$ **from** $\mathcal{A}^*$**.** Add $P_i \to \mathcal{C}$. Output $\texttt{proofs}[i]$ to $\mathcal{A}^*$.

Compiling from $\mathcal{F}_{\mathsf{anizk}}$ *to* $\mathcal{F}_{\mathsf{azk}}$. The work of [LR22] focuses on a class of protocols that include discrete-logarithm based relations. The authors prove that many popular Σ-protocol satisfy a form of adaptivity that is preserved by the Fischlin transform. Relevant to our work, the authors show that PoK of equality of representations of a set of group elements in a set of bases can be transformed into adaptive UC-secure NIZKPoK. Our protocol Π_{azk} in Figure 5 demonstrates that $\mathcal{F}_{\mathsf{anizk}}$ can be used to UC-realize $\mathcal{F}_{\mathsf{azk}}$ above, under certain conditions on the output of their ideal adversary $\mathcal{A}^*_{\mathsf{anizk}}$ as proven with Theorem 1. Referencing [LR22], the first condition can be verified by inspection of the ideal adversary $\mathcal{A}^*_{\mathsf{anizk}}$ and the second condition is satisfied by their definitions.

Figure 7: $\Pi_{\mathsf{azk}}(\mathcal{R})$**.**
Adaptive Zero Knowledge Protocol.

The protocol is parameterized by the relation $\mathcal{R}$ and is executed between the parties in $\mathcal{P}$ with access to hybrid functionalities $\mathcal{F}_{\mathsf{anizk}}$ and $\mathcal{F}_{\mathsf{bc}}$.

On $(\texttt{ZK-prove}, \mathsf{ssid}, \phi, w)$**, Party** $P_i \in \mathcal{P}$**:**

1. Query $(\texttt{NIZK-init})$ to $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
2. Await $\texttt{NIZK-initialized}$ from $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
3. Query $(\texttt{NIZK-prove}, \phi, w)$ to $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
4. Await $(\texttt{NIZK-proof}, \phi, \pi)$ from $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
5. Broadcast $(\texttt{ZK-proof}, \mathsf{ssid}, \phi, \pi)$ to $\mathcal{P}$.

On $(\texttt{ZK-proof}, \mathsf{ssid}, \phi, \pi)$ **from** $P_j \in \mathcal{P}$**, Party** $P_i \in \mathcal{P}$**:**

6. Query $(\texttt{NIZK-verify}, \phi, \pi)$ to $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
7. Await $(\texttt{NIZK-verify}, \phi, \pi, b)$ from $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$.
8. If $b = 1$: output $(\texttt{ZK-proven}, \mathsf{ssid}, P_j, \phi)$.

Theorem 1 (Security of Π_{azk}**).** *For an NP relation $\mathcal{R}$ the protocol $\Pi_{\mathsf{azk}}(\mathcal{R})$ in Figure 7 UC-realizes the functionality $\mathcal{F}_{\mathsf{azk}}(\mathcal{R})$ in Figure 5 in the $(\mathcal{F}_{\mathsf{anizk}}, \mathcal{F}_{\mathsf{bc}})$-hybrid model in the presence of an adaptive malicious adversary $\mathcal{A}$ under the following conditions on the ideal adversary $\mathcal{A}^*_{\mathsf{anizk}}$ to $\mathcal{F}_{\mathsf{anizk}}(\mathcal{R})$:*

– *the tuple* $(\texttt{NIZK-algo}, [\cdots])$ *is immediately sent to* $\mathcal{F}_{\mathsf{anizk}}$ *on* $(\texttt{NIZK-init})$*,*

– Ver *in the tuple is deterministic, and for all* $(\phi, w) \in \mathcal{R}$*:*

$$\Pr\left[\mathsf{Ver}(\phi,\pi)=1 \;\middle|\; \begin{array}{c}(\mathsf{pp},\tau_{\mathcal{S}}) \leftarrow \mathcal{S}.\mathsf{SetUp} \\ (\pi,\tau_\pi) \leftarrow \mathcal{S}.\mathsf{Prv}(\phi)\end{array}\right] = 1-\mathsf{negl}(\lambda).$$

4 Distributed Key Generation with Guaranteed Output Delivery

In this section we present the distributed key generation functionality $\mathcal{F}_{\mathtt{dkg}}$ (Figure 8) and a three-round protocol $\Pi^{\mathsf{go}}_{\mathtt{dkg}}$ (Figures 9, 1)that UC-realizes $\mathcal{F}_{\mathtt{dkg}}$ with guaranteed output in the synchronous setting under the assumption of an honest majority. In Theorem 2, we state the security of $\Pi^{\mathsf{go}}_{\mathtt{dkg}}$.

4.1 The Distributed Key Generation Functionality

The distributed key generation functionality $\mathcal{F}_{\mathtt{dkg}}$ appears in Figure 8. The functionality $\mathcal{F}_{\mathtt{dkg}}(\mathcal{G}, t)$ consists of two phases: *initialization*, and *key generation*. After a one-time setup via the `DKG-init` interface, parties can query `DKG-gen` to distributively generate a key pk with a subset of parties $\mathcal{Q} \subseteq \mathcal{P}$. All queries implicitly include a session identifier and each invocation of key generation explicitly includes a subsession identifier ssid. Towards adaptive corruptions, the functionality accepts corruption queries `DKG-corrupt` at any point. In response to a corruption query the functionality outputs the secret key shares of the newly corrupted party P_i in each subsession ssid such that $P_i \in \mathcal{Q}$ to $\mathcal{A}^*$.

Once the functionality is invoked by all parties in $\mathcal{Q}$ for ssid the functionality samples a secret key and leaks the corresponding public key to the ideal adversary $\mathcal{A}^*$. After this, the functionality receives the corrupted party key shares from $\mathcal{A}^*$ via a query of `DKG-shares` and defines the secret key shares of the honest parties. The functionality delivers output to the parties in the subsession at the adversary's discretion. Specifically, the adversary can abort any subsession with a `DKG-iabort` query, and, as this interface models identifiable abort, must include the identifier of a corrupted party. We note that the functionality does not *eliminate* the identified party; honest parties can choose to exclude a party identified as corrupted, or otherwise processes the identifiable abort output in whichever manner described by some policy that we consider to be outside of the scope of this work.

In terms of record keeping, the functionality maintains multiple tables to record subsession specific variables. The table $\mathtt{keys}[\mathsf{ssid}]$ records the key pair $(\mathsf{pk}, \mathsf{sk})$ for subsession ssid. The table $\mathtt{shares}[i][\mathsf{ssid}]$ records the share of party P_i in subsession ssid. The table $\mathtt{status}[\mathsf{ssid}]$ records the status of subsession ssid, which can be set to: `ready` if the functionality has sampled and is prepared to deliver output for ssid, $\top$ if the functionality has delivered key shares for ssid, or $\perp$ if the subsession ssid has been aborted by $\mathcal{A}^*$.

Figure 8: $\mathcal{F}_{\mathtt{dkg}}(\mathcal{G}, t)$**.**
Distributed Key Generation Functionality

The distributed key generation functionality is parameterized by a group $\mathcal{G} := (\mathbb{G}, G, q)$, and a threshold $t < n$. The functionality interacts with the n parties in $\mathcal{P}$ and the ideal adversary $\mathcal{A}^*$ via the queries below, all of which implicitly include a session identifier sid. Upon invocation the functionality receives the set of corrupted parties $\mathcal{C} \subset \mathcal{P}$ from $\mathcal{A}^*$, and initializes empty tables $\mathtt{keys}[], \mathtt{shares}[][], \mathtt{status}[]$.

On $(\mathtt{DKG\text{-}init}, i)$ **from** $P_i \in \mathcal{P}$**.** If the record does not exist, record and send $(i, \mathcal{P})$ to $\mathcal{A}^*$. Once queried by all $P_j \in \mathcal{P} \setminus \mathcal{C}$, on receipt of $(\mathsf{initialized})$ from $\mathcal{A}^*$ output $(\mathsf{initialized})$ to all parties in $\mathcal{P} \setminus \mathcal{C}$.

On $(\mathtt{DKG\text{-}gen}, \mathsf{ssid}, \mathcal{Q}, i)$ **from** $P_i \in \mathcal{P}$**.** Ignore the query if there exists a record $(\mathsf{ssid}', \mathcal{Q}')$ with $\mathcal{Q}' \neq \mathcal{Q}$, or if $P_i \notin \mathcal{Q} \subseteq \mathcal{P}$. If there does not exist a record $(\mathsf{ssid}, \cdot)$ record and send $(\mathsf{ssid}, \mathcal{Q})$ to $\mathcal{A}^*$. Record and send (ssid, i) to $\mathcal{A}^*$. If (ssid, j) is recorded for all $P_j \in \mathcal{Q}$ sample $\mathsf{sk} \leftarrow_{\$} \mathbb{Z}_q$, define $\mathsf{pk} := \mathsf{sk} \cdot G$, record $\mathtt{keys}[\mathsf{ssid}] := (\mathsf{pk}, \mathsf{sk})$, and send $(\mathtt{DKG\text{-}pk}, \mathsf{ssid}, \mathsf{pk})$ to $\mathcal{A}^*$.

On $(\texttt{DKG-shares}, \mathsf{ssid}, \{(j, \mathsf{sk}_j)\}_{P_j \in \mathcal{C}})$ **from** $\mathcal{A}^*$**.** Ignore the query if there is no record $\texttt{keys}[\mathsf{ssid}] := (\mathsf{pk}, \mathsf{sk})$. Retreive $(\mathsf{ssid}, \mathcal{Q})$. If $\mathsf{sk}_j \notin \mathbb{Z}_q$ for some $P_j \in \mathcal{Q} \cap \mathcal{C}$ define $\mathsf{sk}_j := 0$. Compute a random degree $t-1$ polynomial $S(X)$ that interpolates $(0, \mathsf{sk})$ and (j, sk_j) for all $P_j \in \mathcal{Q} \cap \mathcal{C}$. For each $P_i \in \mathcal{Q}$ record $\texttt{shares}[i][\mathsf{ssid}] := S(i)$. Record $\texttt{status}[\mathsf{ssid}] := \mathsf{ready}$.

On $(\texttt{DKG-deliverkey}, \mathsf{ssid})$ **from** $\mathcal{A}^*$**.** Ignore unless $\texttt{status}[\mathsf{ssid}] = \mathsf{ready}$. Overwrite $\texttt{status}[\mathsf{ssid}] := \top$. Retrieve $(\mathsf{ssid}, \mathcal{Q})$ and $(\mathsf{ssid}, \mathsf{pk}, \cdot)$. Send $(\texttt{DKG-keys}, \mathsf{ssid}, \mathsf{pk}, \texttt{shares}[i][\mathsf{ssid}])$ to every $P_i \in \mathcal{Q} \setminus \mathcal{C}$.

On $(\texttt{DKG-iabort}, \mathsf{ssid}, P_j \in \mathcal{C})$ **from** $\mathcal{A}^*$**.** Ignore unless there is a record $(\mathsf{ssid}, \mathcal{Q})$ with $j \in \mathcal{Q}$, or if $\texttt{status}[\mathsf{ssid}] = \top$. Set $\texttt{status}[\mathsf{ssid}] := \bot$. Record and send $(\texttt{DKG-iabort}, \mathsf{ssid}, j)$ to each $P_i \in \mathcal{Q} \setminus \mathcal{C}$.

On $(\texttt{DKG-corrupt}, P_i \in \mathcal{P})$ **from** $\mathcal{A}^*$**.** Add $P_i \to \mathcal{C}$ and send $(i, \texttt{shares}[i])$ to $\mathcal{A}^*$.

4.2 Three Round DKG with Guaranteed Output Delivery

The three round distributed key generation protocol $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ is detailed in Figure 9 and overviewed in Figure 1.

One Time Set Up Round. On the first input from $\mathcal{Z}$, parties run a non-interactive *set up*, to sample and broadcasts randomness to all parties in $\mathcal{P}$, used to define the group elements of the protocol, and register a discrete-log based public key $k_i^{\mathsf{pki}} \times G = K_i$ with $\mathcal{F}_{\mathsf{pki}}$. These public keys are essentially the ephemeral keys in pairwise Diffie-Hellman Key Exchanges between the parties in $\mathcal{P}$. The protocol waits until all parties have completed setup before proceeding to key generation. Parties query the random oracle $\mathtt{H}_{\mathbb{G}}(\mathsf{id}, \dots)$ for $\mathsf{id} \in \{u, v, w\}$ with the randomness broadcasted by each party in the setup, to compute the group generators H_u, H_v, H_w which define the Pedersen commitment key.

Round 1: Share. Each party takes the role of the dealer in a Verifiable Secret Sharing (VSS, Definition 5). Parties threshold share two random values $s_{i,0}, w_{i,0} \leftarrow_{\$} \mathbb{Z}_q$ as the scalars of group elements G, H_w, and two independent threshold sharings of 0 with the group elements H_u, H_v. To enforce that the point $(0, 0)$ is on the sharing polynomials associated to H_u, H_v the protocol requires that the party proves that the shared secrets are of this form. Specifically, each party proves knowledge of the opening of the commitment $A_{i,0} = s_{i,0}G + w_{i,0}H_w (+0H_u + 0H_v)$ with commitment key G, H_w with π_i^{pd}. In this protocol, the proof system Π_{pd} is a NIZKPoK that satisfies adaptive zero knowledge against adaptive adversaries (AZK, Def 9) and simulation soundness (SS, Def 10) as in Section 3. Shares are encrypted and distributed over the broadcast channel. To encrypt the shares in each subsession ssid parties query the random oracle $\mathtt{H}_{\mathsf{sh}}(\mathsf{ssid}, i, j, K_{i,j})$ to compute a symmetric key, where $K_{i,j}$ is the Diffie-Hellman key computed from the keys registered with $\mathcal{F}_{\mathsf{pki}}$, and use the One Time Pad. The first round message contains: (1) the VSS commitments $\mathbf{A}_i$, (2) the proof π_i^{pd}, and (3) the set of encrypted shares $\{\mathsf{ct}_{i,j}\}_{P_j \in \mathcal{P}}$. Towards guaranteeing output we require all parties participating in the subsession broadcast a (potentially empty) message in the first round prior the start of the second round.

Round 2: Verification & Complaints. Complaints are verifiable and take a single round. VSS shares are encrypted with a Diffie-Hellman key and exchanged over broadcast; complaints can be verified non-interactively. All parties agree on whether any given party actually distributed shares to the remaining parties, and whether the broadcasted proofs π_j^{pd} verify. Any party that did not distribute shares or a verifying proof is disqualified. For all qualified parties P_j, each party P_i decrypts the ciphertext $\mathsf{ct}_{j,i}$ and verifies the decrypted share against the broadcasted commitments $\mathbf{A}_j$.

If a share fails to verify P_i broadcasts a complaint against party P_j. Complaints take the form of a zero knowledge proof of correctness of the shared Diffie-Hellman key. Specifically, a zero knowledge (ZK, Def 9) and sound (S, Def 10) NIZK Π_{dh} of the relation $\mathcal{R}_{\mathsf{dh}}$ as in Section 3 is used to prove that $K_{i,j}$ is the Diffie-Hellman key corresponding to K_i and K_j.Party P_i issues a complaint against P_j that contains: (1) the

shared key $K_{i,j}$, and (2) a proof $\pi^{\mathsf{dh}}_{i,j}$ that $(K_i, K_j, K_{i,j})$ is a Diffie-Hellman tuple. Complaints are verified by verifying $\pi^{\mathsf{dh}}_{i,j}$, decrypting the dealt share $\mathsf{sh}_{j,i}$, and disqualifying P_j if the decrypted share does not verify against $\mathbf{A}_j$. We similarly require all parties in the subsession to broadcast (potentially empty) complaints prior the start of the third round to guarantee that the honest parties will terminate with valid shares of the same secret and agree on the set of qualified parties.

Round 3: Reveal the Public Key. At the start of Round 3 parties define the transcript of the protocol thus far and query $\mathtt{H}_{\mathbb{G}}$ on input the transcript and subsession identifier to define a new group generator $F \in \mathbb{G}$. After verifying the complaints, the parties agree on the subset of qualified parties $\mathcal{V} \subseteq \mathcal{Q}$. Each party aggregates the shares received from these qualified parties $(s_i, u_i, v_i, w_i) := \sum_{P_j \in \mathcal{V}} \mathsf{sh}_{j,i}$ and computes a commitment to the aggregate shares, $B_i = s_i G + u_i F + v_i H_v$ with commitment key G, F, H_v. To enforce this behavior parties prove that the commitment B_i has an equal opening to $A_i := s_i G + u_i H_u + v_i H_v + w_i H_w$ with π^{eq}_i, generated with a NIZK Π_{eq}that satisfies adaptive zero knowledge against adaptive adversaries (AZK, Def 9), and simulation soundness (SS, Def 10) as in Section 3. The third round broadcasted message of party P_i includes the commitment B_i and the proof π^{eq}_i.

Output Reconstruction. Once t parties have broadcast a commitment B_i together with a verifying proof π^{eq}_i the public key can be computed by interpolating t many pairs (j, B_j) using Lagrange interpolation to recover polynomial $B(X)$, and evaluate the polynomial $\mathsf{pk} := B(0)$.

Figure 9: $\Pi^{\mathsf{go}}_{\mathsf{dkg}}(\mathcal{G}, t)$**.**
Distributed Key Generation with Guaranteed Output Delivery.

The DKG protocol parameterized by a group $\mathcal{G} := (\mathbb{G}, G, q)$ and threshold $t < n$ is executed between the n parties in $\mathcal{P}$ with access to:

- Functionalities: $\mathcal{F}_{\mathsf{bc}}$, $\mathcal{F}_{\mathsf{pki}}$, $\mathcal{F}_{\mathsf{ro}}$ instances $\mathtt{H}_{\mathbb{G}}$ and $\mathtt{H}_{\mathtt{sh}}$.
- NIZK $\Pi_{\mathsf{dh}} = (\mathsf{Prv}^{\mathtt{H}_{\mathsf{dh}}}_{\mathsf{dh}}, \mathsf{Ver}^{\mathtt{H}_{\mathsf{dh}}}_{\mathsf{dh}})$ ZK and S for $\mathcal{R}_{\mathsf{dh}}$ (Defs 7, 9 10)
- NIZK $\Pi_{\mathsf{pd}} = (\mathsf{Prv}^{\mathtt{H}_{\mathsf{pd}}}_{\mathsf{pd}}, \mathsf{Ver}^{\mathtt{H}_{\mathsf{pd}}}_{\mathsf{pd}})$ aZK and SS for $\mathcal{R}_{\mathsf{pd}}$ (Defs 7, 9 10)
- NIZK $\Pi_{\mathsf{eq}} = (\mathsf{Prv}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}}, \mathsf{Ver}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}})$ aZK and SS for $\mathcal{R}_{\mathsf{eq}}$ (Defs 7, 9 10)

Below is the code of party $P_i \in \mathcal{P}$:
The contents of the state of P_i are highlighted and flushed right.

Initialization.

On $(\mathtt{DKG\text{-}init}, i)$**:** k^{pki}_i

1. $K_i := k^{\mathsf{pki}}_i \cdot G$; $k^{\mathsf{pki}}_i \leftarrow_{\$} \mathbb{Z}_q$; $r_i \leftarrow_{\$} \{0,1\}^{\lambda}$
2. Query $(\mathtt{PKI\text{-}register}, i, k^{\mathsf{pki}}_i, K_i)$ to $\mathcal{F}_{\mathsf{pki}}$.
3. Broadcast $(\mathtt{DKG\text{-}init}, i, r_i)$ to $\mathcal{P}$.

On (j, r_j) **from all parties in** $\mathcal{P}$:

4. For $\mathsf{id} \in \{u, v, w\}$: $H_{\mathsf{id}} := \mathtt{H}_{\mathbb{G}}(\mathsf{id}, (1, r_1, \ldots, n, r_n))$.
5. Define $\mathbf{H} := (G, H_u, H_v, H_w)$.
6. Output $\mathsf{initialized}$.

Key Generation

On $(\mathtt{DKG\text{-}gen}, \mathsf{ssid}, \mathcal{Q}, i)$: $s_{i,0}, w_{i,0}, \mathbf{r}^{\mathsf{sh}}_i, r^{\mathsf{pd}}_i$

7. Define $\mathtt{BAD}_i := \emptyset$, $\mathcal{V} := \mathcal{Q}$.

8. $((\mathsf{sh}_{i,1},\dots,\mathsf{sh}_{i,n}),\mathbf{A}_i=(A_{i,0},\cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H},(s_{i,0},0,0,w_{i,0});\mathbf{r}_i^{\mathsf{sh}})$
$s_{i,0},w_{i,0} \leftarrow_{\$} \mathbb{Z}_q$, $\mathbf{r}_i^{\mathsf{sh}} \leftarrow_{\$} \mathbb{Z}_q^{4\times(t-1)}$.
9. $\phi_i^{\mathsf{pd}} := ((G,H_w),A_{i,0})$
10. $\pi_i^{\mathsf{pd}} := \mathsf{Prv}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}(\phi_i^{\mathsf{pd}},(s_{i,0},w_{i,0});r_i^{\mathsf{pd}})$; $r_i^{\mathsf{pd}} \leftarrow_{\$} \{0,1\}^{\lambda_P}$
11. For all $P_j \in \mathcal{Q}\setminus\{P_i\}$:
    (a) Retrieve (j,K_j) from $\mathcal{F}_{\mathsf{pki}}$.
    (b) $K_{i,j} := k_i^{\mathsf{pki}} \cdot K_j$
    (c) $\mathsf{ct}_{i,j} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid},i,j),K_{i,j}) \oplus \mathsf{sh}_{i,j}$
12. Broadcast $(\mathsf{ssid},i,\mathbf{A}_i,\pi_i^{\mathsf{pd}},\{(j,\mathsf{ct}_{i,j})\}_{P_j\in\mathcal{Q}})$.

Complaints

On $(\mathsf{ssid},j,\mathbf{A}_j,\pi_j^{\mathsf{pd}},\{(j',\mathsf{ct}_{j,j'})\}_{P_{j'}\in\mathcal{Q}})$ **from** $P_j\in\mathcal{Q}$: $\{\mathsf{sh}_{j,i}\}_{P_j\in\mathcal{Q}},\{r_{i,j}^{\mathsf{dh}}\}_{(j,\dots)\in\mathtt{BAD}_i}$

13. Remove P_j from $\mathcal{V}$ if any of the following hold:
    – $(j',\mathsf{ct}_{j,j'})$ was not broadcast for some $P_{j'}\in\mathcal{Q}$.
    – $1 \neq \mathsf{Ver}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}(A_{j,0},\pi_j^{\mathsf{pd}})$ for $\mathbf{A}_j := (A_{j,0},\dots)$.
14. $\mathsf{sh}_{j,i} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid},j,i),K_{i,j}) \oplus \mathsf{ct}_{j,i}$
15. If $1 \neq \mathtt{VSVerify}(\mathbf{H},\mathbf{A}_j,\mathsf{sh}_{j,i},i)$:
    (a) $\phi_{i,j}^{\mathsf{dh}} := (G,K_i,K_j,K_{i,j})$
    (b) $\pi_{i,j}^{\mathsf{dh}} \leftarrow \mathsf{Prv}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}\left(\phi_{i,j}^{\mathsf{dh}},k_i^{\mathsf{pki}};r_{i,j}^{\mathsf{dh}}\right)$; $r_{i,j}^{\mathsf{dh}} \leftarrow_{\$} \{0,1\}^{\lambda_P}$
    (c) Add $\left(\mathsf{ssid},j,K_{i,j},\pi_{i,j}^{\mathsf{dh}}\right) \rightarrow \mathtt{BAD}_i$.
16. Once received from all $P_j\in\mathcal{Q}$: Broadcast $(\mathsf{ssid},i,\mathtt{BAD}_i)$.

Reveal the Public Key

On $(\mathsf{ssid},j,\mathtt{BAD}_j)$ **from** $P_j\in\mathcal{V}$**:** r_i^{eq}

17. Query $F \leftarrow \mathtt{H}_{\mathbb{G}}(\mathsf{ssid},\mathsf{trans})$ with trans the transcript of ssid thus far.
18. Define $\mathbf{F} := (G,F,H_v)$
19. For all $(\mathsf{ssid},j',K_{j,j'},\pi_{j,j'}^{\mathsf{dh}}) \in \mathtt{BAD}_j$ remove $P_{j'}$ from $\mathcal{V}$ if:
    (a) $1 = \mathsf{Ver}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}\left(\phi_{j,j'}^{\mathsf{dh}},\pi_{j,j'}^{\mathsf{dh}}\right)$ for $\phi_{j,j'}^{\mathsf{dh}} := (G,K_j,K_{j'},K_{j,j'})$, and
    (b) $1 \neq \mathtt{VSVerify}(\mathbf{H},\mathbf{A}_{j'},\mathsf{sh}_{j',j},j)$ for $\mathsf{sh}_{j',j} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid},j',j),K_{j,j'}) \oplus \mathsf{ct}_{j',j}$
20. Once received from all $P_j\in\mathcal{V}$: continue.
21. $(s_i,u_i,v_i,w_i) \leftarrow \mathtt{AggSh}(\{\mathsf{sh}_{j,i}\}_{P_j\in\mathcal{V}})$
22. $C_i := s_i\cdot G + u_i\cdot H_u + v_i\cdot H_v + w_i\cdot H_w$
23. $B_i := s_i\cdot G + u_i\cdot F + v_i\cdot H_v$
24. $\phi_i^{\mathsf{eq}} := (\mathbf{H},\mathbf{F},C_i,B_i)$
25. $\pi_i^{\mathsf{eq}} := \mathsf{Prv}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}(\phi_i^{\mathsf{eq}},(s_i,u_i,v_i,w_i);r_i^{\mathsf{eq}})$; $r_i^{\mathsf{eq}} \leftarrow_{\$} \{0,1\}^{\lambda_P}$
26. Broadcast $(\mathsf{ssid},i,B_i,\pi_i^{\mathsf{eq}})$.

Output Reconstruction

On $\left(\mathsf{ssid},j,B_j,\pi_j^{\mathsf{eq}}\right)$ **with** $P_j\in\mathcal{V}$ **:**

27. $C_j := \sum_{k=0}^{t-1} \sum_{P'_j \in \mathcal{V}} (j^k) A_{j',k}$
28. Remove P_j from $\mathcal{V}$ if $1 \neq \mathsf{Ver}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}\left(((\mathbf{H}, \mathbf{F}), C_j, B_j), \pi_j^{\mathsf{eq}}\right)$.

Once verifying messages received from t parties in $\mathcal{V}$: continue.

29. $B(X) \leftarrow \mathtt{Interpolate}(\{(j, B_j)\}_{j \in \mathcal{V}})$
30. $\mathsf{pk} := B(0)$, $\mathsf{sk}_i := s_i$
31. **Output** $(\mathsf{pk}, \mathsf{sk}_i)$.

Theorem 2 (Security of $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$). *For all $\mathcal{G}$ and all $t < n$ protocol $\Pi_{\mathsf{dkg}}^{\mathsf{go}}(\mathcal{G}, t)$, in Figure 9, UC-realizes functionality $\mathcal{F}_{\mathsf{dkg}}$ in the $(\mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model in the presence of an adaptive malicious adversary under the DDH assumption $\mathcal{G}$, given Π_{dh}, Π_{pd} and Π_{eq} are ZK+S, AZK+SS and AZK+SS NIZKs (Definitions 7, 8, 9, 10), respectively. Moreover, the protocol achieves guaranteed output delivery for an honest majority ($t < n/2$) under the condition that $\mathcal{F}_{\mathsf{bc}}$ guarantees delivery of all messages.*

Proof. The simulator $\mathcal{S}_{\mathtt{DKG}}$ is separated into three parts[9] that share state for ease of presentation: $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}$ in Figure 11 details the emulation of the hybrid functionalities, $\mathcal{S}_{\mathsf{dkg}}^{\Pi}$ in Figure 12 details the simulation of the honest parties in an execution of subsession ssid of $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$ with $\mathcal{A}$, and $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}$ in Figure 13 details the simulation of the honest party states upon adaptive corruption instructions. The "global" simulator, $\mathcal{S}_{\mathtt{DKG}}$ in Figure 10 gives the details of the simulation, simulates set up upon invocation, invokes the parts of the simulator, and routes communication between these parts and any external ITIs.

Global Simulator Summary: $\mathcal{S}_{\mathtt{DKG}}$. We present the simulator modularly; for each ssid there is a *protocol* simulator $\mathcal{S}_{\mathsf{dkg}}^{\Pi}$ that simulates the protocol execution and a *corruption* simulator $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}$ that computes party states on adaptive corruptions, there is a single *hybrids* simulator $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}$ that emulates the hybrid functionailties, and finally a *global* simulator in Figure 10 that can be thought of as a wrapper that routes incoming and outgoing messages to the aforementioned parts. We stress that our simulator is presented this way for ease of understanding.

Figure 10: $\mathcal{S}_{\mathtt{DKG}}(\mathcal{G}, t)$: Global Simulator.
Global Simulator for $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$.

The simulator $\mathcal{S}_{\mathtt{DKG}}$ for $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$ is parameterized by a cyclic group $\mathcal{G} = (\mathbb{G}, G, q)$, the threshold t. The simulator forwards all communication between $\mathcal{A}$ and $\mathcal{Z}$ without modification.

For ease of presentation the simulator has been divided into three parts: $\mathcal{S}_{\mathsf{dkg}}^{\Pi}$, $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}$, and $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}$:

Simulator $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}$ in Figure 11 emulates hybrid functionalities $(\mathcal{F}_{\mathsf{ro}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{bc}})$. We highlight that there is a single instance of each hybrid functionality other than $\mathcal{F}_{\mathsf{ro}}$. We denote by $\mathtt{H}_{\mathbb{G}}$ and $\mathtt{H}_{\mathtt{sh}}$ independent instances of $\mathcal{F}_{\mathsf{ro}}$ with outputs in $\mathbb{G}$, and $\{0,1\}^{4\log(q-1)}$, respectively.

Simulator $\mathcal{S}_{\mathsf{dkg}}^{\Pi}$ in Figure 12 emulates the n parties with identifiers in $\mathcal{P} \setminus \mathcal{C}$. There is an instance of $\mathcal{S}_{\mathsf{dkg}}^{\Pi}$ for each subsession ssid with signing quorum, $\mathcal{Q} \subseteq \mathcal{P}$, the simulator sends protocol messages for the parties in $\mathcal{Q} \setminus \mathcal{C}$.

Simulator $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}$ in Figure 13 manages corruption queries. There is an instance of $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}$ associated to each ssid with $\mathcal{Q} \subseteq \mathcal{P}$, and on a corruption query the simulator generates a part of the secret state on behalf of the newly corrupted party that is consistent with the inputs/outputs of $\mathcal{S}_{\mathsf{dkg}}^{\Pi}(\mathsf{ssid})$.

[9] This presentation is to aid with understanding. We stress that there is no extra machinery or "fancy business" about our simulator in comparison with standard UC simulators.

These parts have access to the same storage. All simulators make black-box use of the following proof systems:

- Π_{pd} for $\mathcal{R}_{\mathsf{pd}}$: $(\mathcal{S}_{\mathsf{P}}^{\mathsf{pd}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{pd}})$ as per the AZK property in Definition 9.
- Π_{eq} for $\mathcal{R}_{\mathsf{eq}}$: $(\mathcal{S}_{\mathsf{P}}^{\mathsf{eq}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}})$ as per the AZK property in Definition 9.

On Invocation.

Statically Corrupted Parties. Receive the set of corrupted parties $\mathcal{C} \subseteq \mathcal{P}$ and maintain $\mathcal{H}$ such that $\mathcal{H} := \mathcal{P} \setminus \mathcal{C}$. Initialize $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}(\mathcal{G}, t)$. Forward all hybrid functionality queries and responses between $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}(\mathcal{G}, t)$ and $\mathcal{A}$.

Record Keeping. Initialize the following tables for record keeping:

- $\mathtt{PKI}[i] = k_i^{\mathsf{pki}}$: records the PKI key of each honest $P_i \in \mathcal{P} \setminus \mathcal{C}$.
- $\mathtt{H_{sh}}[]$, $\mathtt{H_{\mathbb{G}}}[]$, $\mathtt{H_{pd}}[]$,$\mathtt{H_{eq}}[]$: input/output for the corresponding instances of $\mathcal{F}_{\mathsf{ro}}$.
- $\mathtt{AZK}[i][\mathcal{R}][\mathsf{ssid}] := \phi_i$: records proof-related information for $\mathcal{R}_{\mathsf{dh}}$, $\mathcal{R}_{\mathsf{pd}}$ and $\mathcal{R}_{\mathsf{eq}}$.
- $\mathtt{func\text{-}key}[\mathsf{ssid}]$: records the public key $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ output by $\mathcal{F}_{\mathsf{dkg}}$ in the subsession ssid.
- $\mathtt{prot\text{-}key}[\mathsf{ssid}]$: records the secret key of $\mathcal{S}_{\mathsf{dkg}}^{\Pi}$ in subsession ssid.
- $\mathtt{rigged\text{-}gen}[q]$: (1) at $q = (\mathsf{ssid}, \dots)$ records (α, s, F) such that $\mathtt{H_{\mathbb{G}}}[q] = F$, and $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} = sG + \alpha F$, (2) at $q = (\mathsf{id}, \dots)$ records $(\tau_{\mathsf{id}}, H_{\mathsf{id}})$ such that $\mathtt{H_{\mathbb{G}}}[q] = H_{\mathsf{id}}$ and $H_{\mathsf{id}} = \tau_{\mathsf{id}} G$ for $\mathsf{id} \in \{u, v, w\}$.
- $\mathtt{func\text{-}secret\text{-}shares}[\mathsf{ssid}][i]$: record the secret key share of $P_i \in \mathcal{Q}$ in subsession ssid
- $\mathtt{vss\text{-}inputs}[\mathsf{ssid}][i] = (\mathbf{s}; \mathbf{r})$: record the vss secrets of $P_i \in \mathcal{Q}$ in subsession ssid
- $\mathtt{curr\text{-}target}[\mathsf{ssid}] = T$: Record the index of the target party $P_T \in \mathcal{Q}$ in subsession ssid.

Initialization. This corresponds to the initialization phase of the protocol.

On $(\mathtt{DKG\text{-}init}, i)$ **from** $\mathcal{F}_{\mathsf{dkg}}$**with** $P_i \in \mathcal{P} \setminus \mathcal{C}$**:**

1. Simulate $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$ Step 1- 3 for P_i.
2. Record simulated $\mathsf{PKI}[i] := k_i^{\mathsf{pki}}$.

On $\{(j, r_j)\}_{P_j \in \mathcal{C}}$ **from** $\mathcal{A}$ **to** $P_i \in \mathcal{P} \setminus \mathcal{C}$ **via** $\mathcal{F}_{\mathsf{bc}}$**:**

3. Define $R := ((1, r_1), \dots, (n, r_n))$.
4. For $\mathsf{id} \in \{u, v, w\}$: Define $H_{\mathsf{id}} \leftarrow \mathtt{H_{\mathbb{G}}}[\mathsf{id}, R]$.
5. Define $\mathbf{H} := (G, H_u, H_v, H_w)$
6. Record $(\mathtt{DKG\text{-}gens}, \mathtt{rigged\text{-}gen}[(u, R)], \mathtt{rigged\text{-}gen}[(v, R)], \mathtt{rigged\text{-}gen}[(w, R)])$.
7. Record and send (initialized) to $\mathcal{F}_{\mathsf{dkg}}$.

Protocol Simulation: for each ssid. Initialize $\mathcal{S}_{\mathsf{dkg}}^{\Pi}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$ and $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$. Forward protocol messages between $\mathcal{S}_{\mathsf{dkg}}^{\Pi}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$ and $\mathcal{A}$. Forward queries and responses for ssid between $\mathcal{S}_{\mathsf{dkg}}^{\Pi}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$ and $\mathcal{F}_{\mathsf{dkg}}$.

Adaptive Corruption: on $(\mathtt{DKG\text{-}corrupt}, i)$ **from** $\mathcal{A}$**.** Forward $(\mathtt{DKG\text{-}corrupt}, i)$ to $\mathcal{F}_{\mathsf{dkg}}$ and receive $\mathtt{shares}[i]$ in response. Add i to $\mathcal{C}$ and remove i from $\mathcal{H}$. For each ssid such that $i \in \mathcal{Q}$ forward $(\mathtt{DKG\text{-}corrupt}, \mathsf{ssid}, i, \mathtt{shares}[i][\mathsf{ssid}])$ to $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$. Await responses from all $\mathcal{S}_{\mathsf{dkg}}^{\mathtt{Corr}}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$. Output the (combined) responses to $\mathcal{A}$.

Hybrids Simulator Summary: $\mathcal{S}_{\mathsf{dkg}}^{\mathcal{F}}$. The hybrid simulator is detailed in Figure 11. The hybrids simulator (1) executes $\mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{bc}}$ honestly and (2) programs all instances of $\mathcal{F}_{\mathsf{ro}}$. Item (2) programs (2a) $\mathtt{H_{sh}}$ to output the

proper shared randomness to decrypt broadcasted ciphertexts to any shares, (2b) $\mathtt{H}_{\mathbb{G}}$ on any "appropriate" query $q = (\mathsf{ssid}, \dots)$ output a rigged F_q such that $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} = sG + \alpha_q F_q$ and (2c) $\mathtt{H}_{\mathbb{G}}$ on any "appropriate" query $q = (\mathsf{id}, (1, r_1), \dots)$ output a trapdoored group elements, H_{id}.

Figure 11: $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}(\mathcal{G}, t)$.
Hybrids Simulator for $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$.

The simulator $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$ in this Figure emulates the hybrid functionalities $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$, where $\mathtt{H}_{\mathbb{G}}$ and $\mathtt{H}_{\mathtt{sh}}$ are instances of $\mathcal{F}_{\mathsf{ro}}$ with outputs in $\mathbb{G}$ and $\{0,1\}^{4\log(q-1)}$, respectively.

Upon Invocation. Enmulates functionalities as below:

$\mathcal{F}_{\mathsf{bc}}$. Emulate as detailed in Section 2.1.

$\mathcal{F}_{\mathsf{pki}}$. Emulate as detailed in Section 2.1.
Record all secret keys in $\mathtt{PKI}[]$, as in Step 2 below for the honest parties.

$\mathcal{F}_{\mathsf{ro}}$. Emulate each instance as below:

8. $\mathtt{H}_{\mathsf{eq}}$: emulate as detailed in Section 2.1.
9. $\mathtt{H}_{\mathbb{G}}$ on input q:
   (a) If $\mathtt{H}_{\mathbb{G}}[q]$ is undefined, $(\mathsf{initialized})$ is not yet recorded and $q = (\mathsf{id}, (1, r_1), \dots, (n, r_n))$ is consistent with the transcript of Initialization thus far:
      i. If $\mathsf{id} \in \{u, v, w\}$:
         – Sample $\tau_{\mathsf{id}} \leftarrow_{\$} \mathbb{Z}_q$
         – Define $H_{\mathsf{id}} := \tau_{\mathsf{id}} \cdot G$
         – Record $\mathtt{rigged\text{-}gen}[q] := (\tau_{\mathsf{id}}, H_{\mathsf{id}})$
      ii. Define $\mathtt{H}_{\mathbb{G}}[q] := H_q$.
   (b) If $\mathtt{H}_{\mathbb{G}}[q]$ is undefined, $(\mathsf{initialized})$ is recorded, $\mathtt{func\text{-}key}[\mathsf{ssid}][\mathcal{S}] = \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ is defined, and $q = (\mathsf{ssid}, \cdots)$ is consistent with the transcript of ssid thus far:
      i. Sample $\alpha, s \leftarrow_{\$} \mathbb{Z}_q$.
      ii. Compute $F := \alpha^{-1}(\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} - sG)$.
      iii. Define $\mathtt{H}_{\mathbb{G}}[q] := F$.
      iv. Record $\mathtt{rigged\text{-}gen}[q] := (\alpha, s, F)$.
   (c) Else if $\mathtt{H}_{\mathbb{G}}[q]$ is undefined:
      – Define $\mathtt{H}_{\mathbb{G}}[q] := F$ for $F \leftarrow_{\$} \mathbb{G}$.
   (d) Output $\mathtt{H}_{\mathbb{G}}[q]$ to P_i.
10. $\mathtt{H}_{\mathtt{sh}}$ on input q:
   (a) If $\mathtt{H}_{\mathtt{sh}}[q]$ is undefined:
      i. If $q = (\mathsf{ssid}, i, j, K_{i,j})$ for $P_i, P_j \in \mathcal{P} \setminus \mathcal{C}$ and $K_{i,j} = \mathtt{PKI}[i] \cdot \mathtt{PKI}[j] \cdot G$:
         Abort simulation.
      ii. Else : Sample $\mathtt{H}_{\mathtt{sh}}[q] \leftarrow_{\$} \{0,1\}^{4\log(q-1)}$.
   (b) Output $\mathtt{H}_{\mathtt{sh}}[q]$ to P_i.

Protocol Simulator Summary: $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$. The protocol simulator is detailed in Figure 12. Essentially, the simulator simulates the honest parties that participate in subsession ssid honestly, with the exceptions that: (1) the

Round 1 commitments $\{A_{T,k}\}_{k=0}^{t}$ of a random honest party, $P_T \leftarrow_{\$} \mathcal{H}$ called the *target party*, are equivocated such that P_T shares $U_T(0) = \alpha$, (2) the ciphertexts exchanged between honest parties are sampled randomly, and (3) the NIZKs are simulated. Item (1) ensures that the protocol will output the public key $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} = sG + \alpha F$ received from the functionality, as required. Item (2) is because that the random oracle $\mathtt{H}_{\mathtt{sh}}$ will be programmed to the "correct" share if either party is corrupted. Item (3) ensures that the randomness used to generate the zero knowledge proofs can be produced for any witness, e.g., any opening of $A_{T,0}$, and ensures that any deviant behavior of P_T is not detected.

Figure 12: $\mathcal{S}^{\Pi}_{\mathsf{dkg}}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$.
Protocol Simulator for $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$.

Simulator $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$ for ssid and simulates $P_i \in \mathcal{Q}\backslash\mathcal{C}$.

On $(\mathtt{DKG\text{-}pk}, \mathsf{ssid}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}})$ **from** $\mathcal{F}_{\mathsf{dkg}}$**.** Record $\mathtt{func\text{-}key}[\mathsf{ssid}] := \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$.

KEY GENERATION

On $(\mathtt{DKG\text{-}gen}, \mathsf{ssid}, \mathcal{Q}, i)$:

11. Simulate party P_i as in $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ Step 7-8.
12. Record VSS input: $\mathtt{vss\text{-}inputs}[i][\mathsf{ssid}] := ((s_{i,0}, 0, 0, w_{i,0}), \mathbf{r}_i)$.
13. Instead of Step 10 in $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$
    - Simulate, $(\mathtt{H}_{\mathsf{pd}}, \pi_i^{\mathsf{pd}}) \leftarrow \mathcal{S}_{\mathsf{P}}^{\mathsf{pd}}(\mathtt{H}_{\mathsf{pd}}, (G, H_w), A_{i,0})$.
    - Record $\mathtt{AZK}[i][\mathcal{R}_{\mathsf{pd}}][\mathsf{ssid}] := ((G, H_w), A_{i,0}, \pi_i^{\mathsf{pd}})$.
14. In Step 11:
    - For all $P_j \in \mathcal{Q} \cap \mathcal{C}$: encrypt shares as in $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$.
    - For all $P_j \in \mathcal{Q}\backslash\mathcal{C}$: sample $\mathsf{ct}_{i,j} \leftarrow_{\$} \{0,1\}^{4\mathsf{log}(q-1)}$.
15. if $P_i' \in \mathcal{Q}\backslash\mathcal{C}$ have initialized $\mathcal{F}_{\mathsf{dkg}}$ for ssid:
    - Query $(\mathtt{DKG\text{-}init}, \mathsf{ssid}, j)$ to $\mathcal{F}_{\mathsf{dkg}}$ for all $P_j \in \mathcal{Q} \cap \mathcal{C}$.
    - Receive $(\mathtt{DKG\text{-}pk}, \mathsf{ssid}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}})$ from $\mathcal{F}_{\mathsf{dkg}}$.
    - Send $(\mathsf{ssid}, \mathtt{ok})$ to $\mathcal{F}_{\mathsf{dkg}}$.
16. Broadcast $(\mathsf{ssid}, i, \mathbf{A}_i, \pi_i^{\mathsf{pd}}, \{(j, \mathsf{ct}_{i,j})\}_{P_j \in \mathcal{Q}})$ to $\mathcal{Q}$ via emulated $\mathcal{F}_{\mathsf{bc}}$.

COMPLAINTS

On $(\mathsf{ssid}, j, \mathbf{A}_j, \pi_j^{\mathsf{pd}}, \{(j, \mathsf{ct}_{j',j})\}_{P_j' \in \mathcal{Q}})$ **through** $\mathcal{F}_{\mathsf{bc}}$ **via** $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$ **:**

17. For all $P_i \in \mathcal{Q}\backslash\mathcal{C}$:
    - Simulate Steps 13-16 in $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ with the change that only corrupt parties are considered
    - For any $(\mathsf{ssid}, j, K_{i,j}, \pi_{i,j}^{\mathsf{dh}}) \in \mathtt{BAD}_i$: Record $\mathtt{AZK}[i][\mathcal{R}_{\mathsf{dh}}][\mathsf{ssid}] \leftarrow r_{i,j}^{\mathsf{dh}}$

REVEAL THE PUBLIC KEY

On $\{\mathsf{ssid}, j, \mathtt{BAD}_j\}_{P_j \in \mathcal{Q}}$ **through** $\mathcal{F}_{\mathsf{bc}}$ **via** $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$ **:**

18. Define $F \leftarrow \mathtt{H}_{\mathbb{G}}(\mathsf{ssid}, \mathsf{trans})$ with trans the transcript of the protocol thus far and record $\mathbf{F} := (G, F, H_v)$.
19. For all $P_j \in \mathcal{Q}\backslash\mathcal{C}$: Simulate Step 19, of $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$.
20. For all $P_j \in \mathcal{Q} \cap \mathcal{C}$

– Interpolate the VSS shares sent to honest parties:

$$S_j(X), U_j(X), V_j(X), W_j(X) \leftarrow \texttt{Interpolate}(\{(j, \mathsf{sh}_{j,i})\}_{P_i \in \mathcal{Q} \setminus \mathcal{H}}),$$
$$((s_{j,0}, u_{j,0}, v_{j,0}, w_{j,0}), \mathbf{r}_j) \leftarrow \texttt{Coeff}(S_j(X), U_j(X), V_j(X), W_j(X)).$$

– Record $\texttt{vss-inputs}[\mathsf{ssid}][j] := ((s_{j,0}, u_{j,0}, v_{j,0}, w_{j,0}), \mathbf{r}_j)$.
– Compute $(s_j, u_j, v_j, w_j) = \mathsf{sh}_{\mathcal{V},j} := \sum_{P_i \in \mathcal{V}} \mathsf{sh}_{i,j}$.
– Record $\texttt{func-secret-shares}[\mathsf{ssid}][j] := s_j$.

21. Send $(\texttt{DKG-shares}, \mathsf{ssid}, \{(j, s_j)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$ to $\mathcal{F}_{\mathsf{dkg}}$.
22. Define $\hat{s} := \sum_{k \in \mathcal{V}} s_{k,0}$. Record $\texttt{prot-key}[\mathsf{ssid}] := \hat{s}$.
23. Retrieve $(\alpha, s, F) \leftarrow \texttt{rigged-gen}[\mathsf{trans}]$.
24. Sample $P_T \leftarrow_{\$} \mathcal{Q} \setminus \mathcal{C}$
25. Record $\texttt{curr-target}[\mathsf{ssid}] := T$.
26. Reshare $\mathcal{E}_{\mathsf{vss}}^{(s,w)}(\mathsf{ssid}, T, \tau_W^{-1}, \{(0, s - \hat{s})\} \cup \{(j, 0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C} \cup \mathcal{I}})$.
27. Reshare $\mathcal{E}_{\mathsf{vss}}^{(u,w)}(\mathsf{ssid}, T, (\tau_W^{-1} \tau_U), \{(0, \alpha)\} \cup \{(j, 0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C} \cup \mathcal{I}})$.
28. For all $P_j \in \mathcal{Q} \setminus \mathcal{C}$:
    Run Step 21-26 of Round 3 in $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$ with the modifications:
    – In Step 21: Use $\mathsf{sh}_{T,j}$, as the share received from P_T, where $((\ldots, \mathsf{sh}_{T,j}, \ldots), \cdot) := \texttt{VSShare}(\mathbf{H}, \mathbf{s}; \mathbf{r})$ with $(\mathbf{s}, \mathbf{r}) \leftarrow \texttt{vss-inputs}[\mathsf{ssid}][T]$.
    – In Step 25:
    • Simulate, $(\mathtt{H}_{\mathsf{eq}}, \pi_i^{\mathsf{eq}}) \leftarrow \mathcal{S}_{\mathsf{P}}^{\mathsf{eq}}(\mathtt{H}_{\mathsf{eq}}, (\mathbf{H}, \mathbf{F}, C_i, B_i))$.
    • Record $\texttt{AZK}[i][\mathcal{R}_{\mathsf{eq}}][\mathsf{ssid}] := ((\mathbf{H}, \mathbf{F}, C_i, B_i), \pi_i^{\mathsf{eq}})$.

Output Reconstruction

On $(\mathsf{ssid}, j, B_j, \pi_i^{\mathsf{eq}})$ **from** t **parties** $P_j \in \mathcal{Q}$ **such that** π_i^{eq} **validates (Step 32 in** $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$**):**

29. Send $(\texttt{DKG-deliverkey}, \mathsf{ssid})$ to $\mathcal{F}_{\mathsf{dkg}}$.

Corruption Simulator Summary: $\mathcal{S}_{\mathsf{dkg}}^{\mathsf{Corr}}$. The corruption simulator is detailed in Figure 13. At any time the states of the honest parties in $\mathcal{Q} \setminus \mathcal{C} \setminus \{P_T\}$, where T is the target party for ssid are *consistent* with the protocol subsession ssid. Unless t parties are corrupted the state of the target party can be adjusted, such that $U_T(0) = \alpha$, to rig the public key output by the protocol.

Upon an adaptive corruption of some $P_i \in \mathcal{Q} \setminus \mathcal{C}$ the simulator receives the secret key of P_i, $\mathsf{sk}_i := S_{\mathcal{F}_{\mathsf{dkg}}}(i)$. As discussed further below, the corruption simulators' behavior depends on whether:
(A) It is the t-th party that gets corrupted, (that is if $|\mathcal{Q} \cap \mathcal{C}| = t$),
(B) More than t parties are corrupted, (that is if $|\mathcal{Q} \cap \mathcal{C}| > t$)
(C) Less than t parties are corrupted and the target party gets corrupted, (that is if $i = \texttt{curr-target}[\mathsf{ssid}]$ and $|\mathcal{Q} \cap \mathcal{C}| < t$).
(D) Less than t parties are corrupted and an honest non-target party gets corrupted. (that is if $i \neq \texttt{curr-target}[\mathsf{ssid}]$ and $|\mathcal{Q} \cap \mathcal{C}| < t$).

Type A: $|\mathcal{Q} \cap \mathcal{C}| = t$**.** Upon returning the state of the t^{th} corrupted party is returned to $\mathcal{A}$ there should not be any remaining target party nor inconsistencies in the states of the honest parties. To achieve this, at the t^{th} corruption the simulator performs a final equivocation of the target party's polynomials and ensure consistency across all honest parties. This equivocation is possible because, at the t^{th} corruption,

the simulator has obtained t shares of the secret polynomial, $S_{\mathcal{F}_{\mathsf{dkg}}}(X)$. Using Lagrange interpolation the simulator can reconstruct $S_{\mathcal{F}_{\mathsf{dkg}}}(X)$, thereby recover the secret key, $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ and discrete logarithm of F, $\tau_F = \mathsf{log}_G F$. Using τ_F the simulator can equivocate the commitments $A_{T,k}$ and B_T of the target party such that $U_T(0) = 0$ and $S_T(X) = \alpha$, thereby ensuring that the state of the target party is consistent with the protocol, consequently the state of all honest parties are consistent with the protocol. The procedure for this equivocation is detailed in Step 31 in Figure 13.
Observe that after the equivocation the two polynomials $S(X) = \sum_{P_i \in \mathcal{V}} S_i(X)$ and $S_{\mathcal{F}_{\mathsf{dkg}}}(X)$ concide at t distinct points. Since both polynomials are of degree $t-1$ and two polynomials of degree $t-1$ that agree on t points must be identical, it follows that $S(X) = S_{\mathcal{F}_{\mathsf{dkg}}}(X)$.

Type B: $|\mathcal{Q} \cap \mathcal{C}| > t$**.** After the t^{th} corruption the state of all honest parties are consistent with the protocol therefore the state can directly be computed and returned to $\mathcal{A}$, without any equivocation of commitments.

Type C: $i = T_{\mathsf{ssid}}$ **and** $|\mathcal{Q} \cap \mathcal{C}| < t$. When the target party, P_T, is corrupted, the simulator must enforce the condition that $U_T(0) = 0$, by equivocating the commitments $\{A_{T,k}\}_{k=0}^{t-1}$, prior to returning $P_T's$ state to. However, this modification leaves the commitments $\{B_j\}_{P_j \in \mathcal{Q} \setminus \mathcal{C}}$ inconsistent with the shares computed from the honest parties recorded polynomials. To restore consistency across all parties except for a single party with an inconsistent state, the simulator samples a new target party $\tilde{T}$ and sets $U_{\tilde{T}}(0) = \alpha$ by equivocating the commitments $\{A_{\tilde{T},k}\}_{k=0}^{t-1}$. The equivocation procedure for both for P_T and $P_{\tilde{T}}$ are detailed in Step 32 in Figure 13. Following this re-targeting, the corrupted party is no longer the target party and the corruption is of Type D and the simulator proceeds accordingly.

Type D: $i \neq T_{\mathsf{ssid}}$ **and** $|\mathcal{Q} \cap \mathcal{C}| < t$**.** Corruption of a non-target party requires the simulator to "hit" the secret key sk_i output by $\mathcal{F}_{\mathsf{dkg}}$, this requires the global polynomial $S(X) = \sum_{i \in \mathcal{V}} S_i(X)$ to evaluate to sk_i at point i. This is achieved by equivocating the commitments $\{A_{i,k}\}_{k=0}^{t-1}$ and B_i such that $S_i(i) = \mathsf{sk}_i - s_i$, where $s_i = \sum_{i \in \mathcal{V}} S_j(i)$, denotes the aggregated contribution from all parties. The procedure for this equivocation is detailed in Step 33 in Figure 13.

Upon processing the different corruption types, the state of the corrupt party is computed as specified in Figure 13 before returned.

Figure 13: $\mathcal{S}_{\mathsf{dkg}}^{\mathsf{Corr}}(\mathsf{ssid}, \mathcal{Q}, \mathcal{G}, t)$**.**
Corruption Simulator for $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$.

The simulator $\mathcal{S}_{\mathsf{dkg}}^{\mathsf{Corr}}$ in this Figure generates a secret state on behalf of a newly corrupted party P_i which is consistent with the transcript of subsession ssid. $\mathcal{V}$ is defined by $\mathcal{S}_{\mathsf{dkg}}^{\Pi}(\mathsf{ssid}, \cdots)$.

Upon receiving $(\texttt{DKG-corrupt}, i, \mathsf{sk}_i)$ **from** $\mathcal{S}_{\texttt{DKG}}$:

30. Update $\texttt{func-secret-shares}[\mathsf{ssid}] := \mathsf{sk}_i$
31. **If the final corruption**, $|\mathcal{Q} \cap \mathcal{C}| = t$: *// match* sk *output by* $\mathcal{F}_{\mathsf{dkg}}$
    (a) Retrieve $\{\mathsf{sk}_j\}_{P_j \in (\mathcal{Q} \cap \mathcal{C})} \leftarrow \texttt{func-secret-shares}[\mathsf{ssid}]$
    (b) Retrieve $(\alpha, s, F) \leftarrow \texttt{rigged-gen}[\mathsf{trans}]$, with trans the transcript of the protocol, with subsession identifier ssid, after round 2.
    (c) Interpolate $S_{\mathcal{F}_{\texttt{DKG}}}(X) \leftarrow \texttt{Interpolate}(\{(j, \mathsf{sk}_j)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$.
    (d) Define $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}} := S_{\mathcal{F}_{\texttt{DKG}}}(0)$ for
    (e) Compute $\tau_F = \texttt{log}_G(F) := \alpha^{-1}(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}} - s)$.
    (f) Retrieve $T \leftarrow \texttt{curr-target}[\mathsf{ssid}]$
    (g) Reshare $\mathcal{E}_{\mathsf{vss}}^{(s,u,w)}(\mathsf{ssid}, T, (\tau_F, \tau_W^{-1}(\tau_F - \tau_U)), \{(0, \alpha)\} \cup \{(j, 0)\}_{P_j \in (\mathcal{Q} \cap \mathcal{C}) \setminus \{i\}})$.
32. **If the target party is corrupted** $(i = \texttt{curr-target}[\mathsf{ssid}])$:
    (a) Sample $P_{\tilde{T}} \leftarrow_{\$} \mathcal{Q} \setminus (\mathcal{C} \cup \{P_T\})$.

(b) Reshare $\mathcal{E}_{\mathtt{vss}}^{(u,w)}(\mathsf{ssid}, T, \tau_W^{-1}\tau_U, \{(0,-\alpha)\} \cup \{(j,0)\}_{P_j \in (\mathcal{Q} \cap \mathcal{C})})$. // detarget P_T

(c) Reshare $\mathcal{E}_{\mathtt{vss}}^{(u,w)}(\mathsf{ssid}, \tilde{T}, \tau_W^{-1}\tau_U, \{0,\alpha\} \cup \{(j,0)\}_{P_j \in (\mathcal{Q} \cap \mathcal{C})})$. // target $P_{\tilde{T}}$

(d) $\mathtt{curr\text{-}target}[\mathsf{ssid}] := \tilde{T}$.

33. **If $|\mathcal{Q} \cap \mathcal{C}| \leq t$, Update the secret key of P_i to be the sk_i received by $\mathcal{F}_{\mathtt{dkg}}$:**

(a) Compute $(s_i, \cdots) := \sum_{P_j \in \mathcal{Q}} \mathsf{sh}_{j,i}$
for $(\cdot, (\cdots, \mathsf{sh}_{j,i}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, \mathtt{vss\text{-}inputs}[j][\mathsf{ssid}])$.

(b) Reshare $\mathcal{E}_{\mathtt{vss}}^{(s,v)}(\mathsf{ssid}, i, \tau_V^{-1}, \{(i, \mathsf{sk}_i - s_i)\} \cup \{(0,0)\} \cup \{(j,0)\}_{P_j \in (\mathcal{Q} \cap \mathcal{C})})$.

34. **Simulate NIZK randomness:** // via $(\mathtt{H}_{\mathsf{pd}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{pd}})$ and $(\mathtt{H}_{\mathsf{eq}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}})$

(a) For π_i^{pd}

i. Retrieve $((s_{i,0}, u_{i,0}, v_{i,0}, w_{i,0}), \mathbf{r_i}) \leftarrow \mathtt{vss\text{-}inputs}[i][\mathsf{ssid}]$

ii. $(\mathtt{H}_{\mathsf{pd}}, r_i^{\mathsf{pd}}) \leftarrow \mathcal{S}_{\mathsf{R}}^{\mathsf{pd}}(\mathtt{H}_{\mathsf{pd}}, \mathtt{AZK}[\mathsf{ssid}][i][\mathcal{R}_{\mathsf{pd}}], (s_{i,0}, w_{i,0}))$.

iii. Overwrite $\mathtt{AZK}[i][\mathcal{R}_{\mathsf{pd}}][\mathsf{ssid}] := r_i^{\mathsf{eq}}$.

(b) For π_i^{eq}

i. Define $(s_i, u_i, v_i, w_i) := \sum_{P_j \in \mathcal{V}} \mathsf{sh}_{j,i}$
for $(\cdot, (\cdots, \mathsf{sh}_{j,i}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, \mathtt{vss\text{-}inputs}[j][\mathsf{ssid}])$.

ii. $(\mathtt{H}_{\mathsf{eq}}, r_i^{\mathsf{eq}}) \leftarrow \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}}(\mathtt{H}_{\mathsf{eq}}, \mathtt{AZK}[\mathsf{ssid}][i][\mathcal{R}_{\mathsf{eq}}], (s_i, u_i, v_i, w_i))$.

iii. Overwrite $\mathtt{AZK}[i][\mathcal{R}_{\mathsf{eq}}][\mathsf{ssid}] := r_i^{\mathsf{eq}}$.

35. **Program $\mathtt{H}_{\mathsf{sh}}$:** // match all shares exchanged with honest parties

(a) Recompute $(\cdot, (\mathsf{sh}_{i,1}, \ldots, \mathsf{sh}_{i,n})) \leftarrow \mathtt{VSShare}(\mathbf{H}, \mathtt{vss\text{-}inputs}[i][\mathsf{ssid}])$.

(b) For all $P_j \in \mathcal{P} \setminus \mathcal{C}$:

– Retrieve ciphertext $\mathsf{ct}_{i,j}$ sent to party P_j in subsession ssid.

– Define $\mathtt{H}_{\mathbb{G}}[\mathsf{ssid}, i, j, K_{i,j}] := \mathsf{ct}_{i,j} \oplus \mathsf{sh}_{i,j}$.

– Recompute $(\cdot, (\cdots, \mathsf{sh}_{j,i}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, \mathtt{vss\text{-}inputs}[j][\mathsf{ssid}])$.

– Retrieve ciphertext $\mathsf{ct}_{j,i}$ received from party P_j in subsession ssid.

– Define $\mathtt{H}_{\mathbb{G}}[\mathsf{ssid}, j, i, K_{i,j}] := \mathsf{ct}_{j,i} \oplus \mathsf{sh}_{j,i}$.

36. **Output** $\mathsf{st}_i = (i, \mathtt{PKI}[i], \mathtt{vss\text{-}inputs}[\mathsf{ssid}][i], \mathtt{AZK}[i][\mathcal{R}_{\mathsf{dh}}], \mathtt{AZK}[i][\mathcal{R}_{\mathsf{pd}}], \mathtt{AZK}[i][\mathcal{R}_{\mathsf{eq}}][\mathsf{ssid}])$ to $\mathcal{S}_{\mathtt{DKG}}$.

Helper Subroutine: VSS Resharing. Towards an easier presentation of the simulator $\mathcal{S}_{\mathtt{DKG}}$, in Definition 11 we define a subroutine that will be used by $\mathcal{S}_{\mathtt{DKG}}$. Looking ahead, the $\mathcal{S}_{\mathtt{DKG}}$ maintains a table $\mathtt{vss\text{-}inputs}[][]$ that at $\mathtt{vss\text{-}inputs}[\mathsf{ssid}][i] = ((s_{i,0}, u_{i,0}, v_{i,0}, w_{i,0}), \mathbf{r})$ records secrets and randomness for simulated honest party P_i that can be used to (re-)generate the Round 1 messages sent by P_i in the view of $\mathcal{Z}$ in subsession ssid. The subroutines $\mathcal{E}_{\mathtt{vss}}^{(\cdot)}$ share memory with $\mathcal{S}_{\mathtt{DKG}}$ and, in summary, the subroutines read a set of VSS inputs from this table and write a new set of VSS inputs to this table.

Definition 11 (PVSS Reshare). *The process $\mathcal{E}_{\mathtt{vss}}^{(\cdot)}$ comes in four forms: $\mathcal{E}_{\mathtt{vss}}^{(s,w)}, \mathcal{E}_{\mathtt{vss}}^{(u,w)}, \mathcal{E}_{\mathtt{vss}}^{(s,v)}, \mathcal{E}_{\mathtt{vss}}^{(s,u,w)}$. Each form $\mathcal{E}_{\mathtt{vss}}^{(\cdot)}$ is defined with access to a two dimensional table $\mathtt{vss\text{-}inputs}[][]$ initially populated by $\mathcal{S}_{\mathtt{DKG}}$, and takes as input: a subsession identifier ssid, a party identifier $i \in [1,n]$, one or two trapdoors $\tau \in \mathbb{Z}_q$, and a set of points $\mathtt{pts} = \{(j, \mathbf{x}_j)\}$ of size at most t.*

1. *If* $|\texttt{pts}| \leq t$.
   - *Let* $\mathcal{I}$ *to be the set of* $(t-1-|\texttt{pts}|)$ *parties with the lowest party identifier in the set* $\mathcal{Q} \cap \{P_j\}_{(j,\cdot)\in\texttt{pts}}$.
   - *Update* $\texttt{pts} \leftarrow \texttt{pts} \cup \{j, 0\}_{P_j \in \mathcal{I}}$
2. *Interpolate a shift polynomial from the set of points:*

$$\begin{aligned} \Delta(X) &\leftarrow \texttt{Interpolate}(\texttt{pts}), \\ (r_\Delta, \mathbf{r}_\Delta) &\leftarrow \texttt{Coeff}(\Delta(X)). \end{aligned}$$

3. *Retrieve the old inputs from the table:*

$$(s_{i,0}, u_{i,0}, v_{i,0}, w_{i,0}), (\mathbf{r}_{s,i}, \mathbf{r}_{u,i}, \mathbf{r}_{v,i}, \mathbf{r}_{w,i}) \leftarrow \texttt{vss-inputs}[\mathsf{ssid}][i].$$

4. *Recompute the secrets of* P_i*, and update the table: The behavior of each specific* $\mathcal{E}^{(\cdot)}_{\texttt{vss}}$ *is as below.*
   (a) $\mathcal{E}^{(s,w)}_{\texttt{vss}}(\mathsf{ssid}, i, \tau, \cdot)$*:*

$$\begin{aligned} s^*_{i,0} &:= s_{i,0} + r_\Delta, \quad & \mathbf{r}^*_{s,i} &:= \mathbf{r}_{s,i} + \mathbf{r}_\Delta \\ w^*_{i,0} &:= w_{i,0} - \tau r_\Delta, \quad & \mathbf{r}^*_{w,i} &:= \mathbf{r}_{w,i} - \tau \mathbf{r}_\Delta \end{aligned}$$

$$\textit{Overwrite}: \texttt{vss-inputs}[\mathsf{ssid}][i] := \begin{pmatrix} (s_{i,0}, s^*_{i,0}, v_{i,0}, w^*_{i,0}), \\ (\mathbf{r}_{s,i}, \mathbf{r}^*_{s,i}, \mathbf{r}_{v,i}, \mathbf{r}^*_{w,i}) \end{pmatrix}.$$

   (b) $\mathcal{E}^{(u,w)}_{\texttt{vss}}(\mathsf{ssid}, i, \tau, \cdot)$*:*

$$\begin{aligned} u^*_{i,0} &:= u_{i,0} + r_\Delta, \quad & \mathbf{r}^*_{u,i} &:= \mathbf{r}_{u,i} + \mathbf{r}_\Delta \\ w^*_{i,0} &:= w_{i,0} - \tau r_\Delta, \quad & \mathbf{r}^*_{w,i} &:= \mathbf{r}_{w,i} - \tau \mathbf{r}_\Delta \end{aligned}$$

$$\textit{Overwrite}: \texttt{vss-inputs}[\mathsf{ssid}][i] := \begin{pmatrix} (s_{i,0}, u^*_{i,0}, v_{i,0}, w^*_{i,0}), \\ (\mathbf{r}_{s,i}, \mathbf{r}^*_{u,i}, \mathbf{r}_{v,i}, \mathbf{r}^*_{w,i}) \end{pmatrix}.$$

   (c) $\mathcal{E}^{(s,v)}_{\texttt{vss}}(\mathsf{ssid}, i, \tau, \cdot)$*:*

$$\begin{aligned} s^*_{i,0} &:= s_{i,0} + r_\Delta, \quad & \mathbf{r}^*_{s,i} &:= \mathbf{r}_{s,i} + \mathbf{r}_\Delta \\ v^*_{i,0} &:= v_{i,0} - \tau r_\Delta, \quad & \mathbf{r}^*_{v,i} &:= \mathbf{r}_{v,i} - \tau \mathbf{r}_\Delta \end{aligned}$$

$$\textit{Overwrite}: \texttt{vss-inputs}[\mathsf{ssid}][i] := \begin{pmatrix} (s^*_{i,0}, u_{i,0}, v^*_{i,0}, w_{i,0}), \\ (\mathbf{r}^*_{s,i}, \mathbf{r}_{u,i}, \mathbf{r}^*_{v,i}, \mathbf{r}_{w,i}) \end{pmatrix}.$$

   (d) $\mathcal{E}^{(s,u,w)}_{\texttt{vss}}(\mathsf{ssid}, i, (\tau_1, \tau_2), \cdot)$*:*

$$\begin{aligned} s^*_{i,0} &:= s_{i,0} + \tau_1 r_\Delta, \quad & \mathbf{r}^*_{s,i} &:= \mathbf{r}_{s,i} + \tau_1 \mathbf{r}_\Delta \\ u^*_{i,0} &:= u_{i,0} - r_\Delta, \quad & \mathbf{r}^*_{u,i} &:= \mathbf{r}_{u,i} - \mathbf{r}_\Delta \\ w^*_{i,0} &:= w_{i,0} - \tau_2 r_\Delta, \quad & \mathbf{r}^*_{w,i} &:= \mathbf{r}_{w,i} - \tau_2 \mathbf{r}_\Delta \end{aligned}$$

$$\textit{Overwrite}: \texttt{vss-inputs}[\mathsf{ssid}][i] := \begin{pmatrix} (s^*_{i,0}, u^*_{i,0}, v_{i,0}, w^*_{i,0}), \\ (\mathbf{r}^*_{s,i}, \mathbf{r}^*_{u,i}, \mathbf{r}_{v,i}, \mathbf{r}^*_{w,i}) \end{pmatrix}.$$

The claim below states the property that we require from $\mathcal{E}^{(\cdot)}_{\texttt{vss}}$ in Definition 5. Essentially, the claim states that the re-sharing process computes new VSS inputs that result in the same set of public commitments, and new verifying shares that are *shifted* in the desired manner, as indicated by $\texttt{mode} \in \{(u,w),(s,v),(s,u,w)\}$ and the fixed input points.

Claim 1. [PVSS Reshare Correctness] *Let* VSShare *be the sharing algorithm in Definition 5 parameterized by* $\ell = 4$*, threshold* t*, number of parties* n *and* $\mathcal{G} = (\mathbb{G}, G, q)$*. For all* $i \in [1, n]$*, all* $\mathsf{ssid} \in \{0,1\}^*$*, all*

$\tau_u, \tau_v, \tau_w, \tau_F \in \mathbb{Z}_q$, *all* $(\mathbf{s}, \mathbf{r}) \in \mathbb{Z}_q^{4\times(t)}$, *and all subsets* $\{(d, x_d)\}_{d\in\mathcal{D}} \subset [1, n] \times \mathbb{Z}_q$ *such that* $|\mathcal{D}| = t$, *it is the case that:*

$$\begin{aligned}
\textit{For all } i \in [n]: \ \mathsf{sh}_i^* &= (s_i, u_i{+}x_i, v_i, w_i{-}\tau_w^{-1}\tau_u x_i) \textit{ for } \texttt{mode} = (u, w) \\
\mathsf{sh}_i^* &= (s_i{+}x_i, u_i, v_i{-}\tau_v^{-1}x_i, w_i) \textit{ for } \texttt{mode} = (s, v) \\
\mathsf{sh}_i^* &= (s_i{+}\tau_F x_i, u_i{-}x_i, v_i, w_i{-}\tau_w^{-1}(\tau_F - \tau_u)x_i) \textit{ for } \texttt{mode} = (s, u, w),
\end{aligned}$$

where $x_i = \Delta(i)$ *and* $\Delta(X) \leftarrow \texttt{Interpolate}(\{(d, x_d)\}_{d\in\mathcal{D}})$.
It holds that if $(((s_1, u_1, v_1, w_1), \ldots, (s_n, u_n, v_n, w_n)), \mathbf{C}) \leftarrow \texttt{VSShare}(\mathbf{H}, \mathbf{s}; \mathbf{r})$ *is a verifiable secret sharing with* $\mathbf{H} := (G, \tau_u G, \tau_v G, \tau_w G)$ *of the elements* $\mathbf{s}, \mathbf{r} = \texttt{vss-inputs}[\mathsf{ssid}][i]$ *before running* $\mathcal{E}_{\texttt{vss}}^{(\texttt{mode})}(\mathsf{ssid}, i, \tau, \{(d, x_d)\}_{d\in\mathcal{D}})$ *then* $(\{\mathsf{sh}_i^*\}_{i\in[n]}, \mathbf{C})$ *is a verifiable secret sharing of the elements* $\mathbf{s}^*, \mathbf{r}^* = \texttt{vss-inputs}[\mathsf{ssid}][i]$ *after running* $\mathcal{E}_{\texttt{vss}}^{(\cdot)}$. *Formally,* $\texttt{VSVerify}(\mathbf{H}, \mathbf{C}, \mathsf{sh}_i^*, i) = 1$ *for all* $i \in [n]$.

Remark 1. Using the notation, $((\mathsf{sh}_1, \ldots, \mathsf{sh}_n), \mathbf{C}) \leftarrow \texttt{VSShare}(\mathbf{H}, \mathbf{s}; \mathbf{r})$, the claim implies $\mathcal{E}_{\texttt{vss}}^{(\cdot)}$ can be run by $\mathcal{S}_{\texttt{DKG}}$ without altering the view of an adversary given access to the commitments $\mathbf{C}$ and a subset of the threshold shares of size $m < t$. Provided that none of the threshold shares in the subset known to the adversary are modified, the adversary's view remains unchanged, even when given access to all the (modified) shares.

Justification of the Claim Clearly, by definition, $((\mathsf{sh}'_1, \ldots, \mathsf{sh}'_n), \mathbf{C}') \leftarrow \texttt{VSShare}(\mathbf{H}, \mathbf{s}^*; \mathbf{r}^*)$ is a verifiable secret sharing of the elements $\mathbf{s}^*, \mathbf{r}^*$ and by construction $\mathsf{sh}_i^* = \mathsf{sh}'_i$ for all $i \in [n]$. Furthermore, if $\mathbf{C} = \mathbf{C}'$ then clearly it also holds, for all $i \in [n]$, that $\texttt{VSVerify}(\mathbf{H}, \mathbf{C}, \mathsf{sh}_i^*, i) = 1$. Thus, to justify the claim, we need to show that the commitment $\mathbf{C}$ is equal to the *fictive* commitment $\mathbf{C}'$. Next, we argue this explicitly for $\texttt{mode} = (u, w)$, and remark that the other modes follows by similar argument. By definition, for any party P_i, we have that $\mathbf{C} = (C_0, C_1, \ldots, C_t)$, where,

$$\begin{aligned}
C_0 &= s_{j,0}G + u_{j,0}H_u + v_{j,0}H_v + w_{j,0}H_w \\
&= (s_{j,0} + \tau_u u_{j,0} + \tau_v v_{j,0} + \tau_w w_{j,0})G, \\
C_j &= r_{s,j}G + r_{u,j}H_u + r_{v,j}H_v + r_{w,j}H_w \\
&= (r_{s,j} + \tau_u r_{u,j} + \tau_v r_{v,j} + \tau_w r_{w,j})G, \quad \forall j \in [t].
\end{aligned}$$

and $\mathbf{C}' = (C'_0, C'_1, \ldots, C'_t)$, where,

$$\begin{aligned}
C'_0 &= s_{j,0}G + r^*_{u,j}H_u + v_{j,0}H_v + w^*_{j,0}H_w \\
&= (s_{j,0} + \tau_u u^*_{j,0} + \tau_v v_{j,0} + \tau_w w^*_{j,0})G, \\
&= (s_{j,0} + \tau_u(u_{j,0}{+}r_\Delta) + \tau_v v_{j,0} + \tau_w(w_{j,0}{-}\tau_w^{-1}\tau_u r_\Delta))G \\
&= (s_{j,0} + \tau_u u_{j,0}{+}\tau_u r_\Delta + \tau_v v_{j,0} + \tau_w w_{j,0}{-}\tau_w\tau_w^{-1}\tau_u r_\Delta)G \\
&= (s_{j,0} + \tau_u u_{j,0} + \tau_v v_{j,0} + \tau_w w_{j,0})G \\
&= C_0 \\
C'_j &= r_{s,j}G + r^*_{u,j}H_u + r_{v,j}H_v + r^*_{w,j}H_w \\
&= (r_{s,j} + \tau_u r^*_{u,j} + \tau_v r_{v,j} + \tau_w r^*_{w,j})G \\
&= (r_{s,j} + \tau_u(r_{u,j}{+}r_\Delta) + \tau_v r_{v,j} + \tau_w(r_{w,j}{-}\tau_w^{-1}\tau_u r_\Delta))G \\
&= (r_{s,j} + \tau_u r_{u,j}{+}\tau_u r_\Delta + \tau_v r_{v,j} + \tau_w r_{w,j}{-}\tau_w\tau_w^{-1}\tau_u r_\Delta)G \\
&= (r_{s,j} + \tau_u r_{u,j} + \tau_v r_{v,j} + \tau_w r_{w,j})G \\
&= C_j, \forall j \in [t].
\end{aligned}$$

Thus, $\mathbf{C} = \mathbf{C}'$, which concludes the justification of the claim.

Hybrid Argument of Indistinguishability .

Hybrid H_0: *Real World.* The real execution of the protocol $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ where the environment $\mathcal{Z}$ runs the protocol in Figure 9 with n parties in $\mathcal{P}$ with identifiers in $\mathcal{I} = [1, n]$, and the dummy adversary $\mathcal{A}$, all with access to ideal functionalities ($\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}}$). The random variable $\mathsf{EXEC}_{\Pi^{\mathsf{go}}_{\mathsf{dkg}},\mathcal{A},\mathcal{Z}}(z)$ denotes the output of an execution of $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ with the adversary $\mathcal{A}$ and environment $\mathcal{Z}$ on input z. This hybrid corresponds to the ensemble:

$$\{\mathsf{EXEC}_{\Pi^{\mathsf{go}}_{\mathsf{dkg}},\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}.$$

Hybrid H_1: *Ideal Layout.* The existing machines (parties, hybrid functionalities, $\mathcal{A}$) are grouped into a single machine $\mathcal{S}_{\mathtt{DKG}}$, called the simulator. Dummy party nodes and an ideal functionality node $\mathcal{F}$ are added between $\mathcal{Z}$ and $\mathcal{S}_{\mathtt{DKG}}$. For now, $\mathcal{F}$ simply forwards inputs/outputs between $\mathcal{S}$ and the dummy party nodes. The three parts of the simulator are also initialized, but simulate their respective machines honestly (exactly as in the real world). This change is syntactic; the view of $\mathcal{Z}$ is unchanged.

Hybrid H_2: *Record Keeping,* `DKG-init` *and* `DKG-shares` *Queries.* The simulator generates queries to $\mathcal{F}$ on behalf of the corrupted parties from the messages received from $\mathcal{A}$. The functionality and the simulator now make all relevant records.[10]

– *Modifications to* $\mathcal{F}$*:*

- Relevant record keeping is done as in Figure 8.
- As in the `DKG-init` of $\mathcal{F}_{\mathtt{dkg}}$: the functionality $\mathcal{F}$ records queries made by the honest dummy parties and $\mathcal{A}^*$. In H_2 this interface is not yet implemented; the functionality does not sample any values or output a public key. The functionality merely forward input/outputs between dummy parties and $\mathcal{A}^*$.
- As in `DKG-shares` of $\mathcal{F}_{\mathtt{dkg}}$: for all corrupt parties, $P_j \in \mathcal{Q} \cap \mathcal{C}$ the functionality records the received shares in $\mathtt{shares}[j][\mathsf{ssid}] = \mathsf{sk}_j$, but otherwise ignores the query if there is no recorded pair $(\mathsf{pk}, \mathsf{sk})$, $\mathtt{keys}[\mathsf{ssid}] = \emptyset$. In H_2 this interface is not yet implemented; $\mathcal{F}$ does not interpolate any secret polynomial.
- As in `DKG-deliverkey` of $\mathcal{F}_{\mathtt{dkg}}$: the functionality $\mathcal{F}$ send out shares to all parties $P \in \mathcal{Q}$, however the shares are given as input to $\mathcal{F}$, in the function call, $(\mathtt{DKG\text{-}deliverkey}, \mathsf{ssid}, \{\mathsf{sk}_i\}_{P_i\in\mathcal{Q}\setminus\mathcal{C}})$, and stored in $\mathtt{shares}[\mathsf{ssid}]$. In H_2 this interface is not yet implemented; $\mathcal{F}$ takes the shares as input from $\mathcal{S}$.
- As in the `DKG-iabort` of $\mathcal{F}_{\mathtt{dkg}}$: the functionality $\mathcal{F}$ forwards an abort query from $\mathcal{A}^*$ to the set of parties $\mathcal{Q}$. In H_2 this interface is fully implemented.
- As in the `DKG-corrupt` of $\mathcal{F}_{\mathtt{dkg}}$: the functionality $\mathcal{F}$ adds the newly corrupt party to the set to the set of corrupt parties $\mathcal{C}$ and outputs the shares of this party, for all subsessions, to $\mathcal{A}^*$. In H_2 this interface is fully implemented.

– *Modifications to* $\mathcal{S}_{\mathtt{DKG}}$*:*

- Relevant record keeping is done by $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$, $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$ and $\mathcal{S}^{\mathtt{Corr}}_{\mathsf{dkg}}$.
- The simulator sends $(\mathsf{initialized})$ to $\mathcal{F}$ once all parties have executed Initialization, as in Step 7.
- *Hybrids Simulator* $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$*:* The simulator emulates functionalities ($\mathcal{F}_{\mathsf{bc}}$, $\mathcal{F}_{\mathsf{pki}}$,$\mathcal{F}_{\mathsf{ro}}$) honestly. The hybrids simulator does all relevant record keeping in Figure 11, including maintaining input/output tables for each instance of $\mathcal{F}_{\mathsf{ro}}$.
- $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$*:* The protocol simulator simulates the honest parties honestly, and does the relevant record keeping done in Figure 12.
  - The simulator submits a `DKG-shares` query to $\mathcal{F}$ as in Step 21 in Figure 12, after receiving (potential empty) Round 1 and 2 messages from every party in subsession ssid.

[10] By "relevant" record keeping we mean that the simulator records the values *that it actually receives/computes*, and creates empty records for the remaining values.

∗ The simulator calls DKG-deliverkey as in Step 29, but includes $\{s_i\}_{i\in\mathcal{Q}\setminus\mathcal{C}}$, in the function call.

- $\mathcal{S}^{\mathsf{Corr}}_{\mathsf{dkg}}$: The corruption simulator returns the real contents of the honest party secret state, unmodified. The simulator does all relevant record keeping as in Figure 13.

  On $(\texttt{DKG-corrupt}, i, \mathsf{sk}_i)$ from $\mathcal{S}_{\texttt{DKG}}$ the simulator composes and returns the state $\texttt{st}_i$ of the corrupt party in subsession ssid according to Figure 13 with the modifications that The hash table, $\texttt{H}_{\texttt{sh}}$, is not updated, that is Step 35 is skipped.

In this game, $\mathcal{F}$ still forwards the outputs sent by $\mathcal{S}_{\texttt{DKG}}$ so the view of $\mathcal{Z}$ is not affected.

Hybrid H_3: *Simulated NIZKs.* In this game, the simulator makes black-box use of the NIZK and NIZKPoK simulators from Definition 7 as described below.

– *Modifications to* $\mathcal{S}_{\texttt{DKG}}$ *:*

- $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$: For all honest $P_i \in \mathcal{H}$, the proofs π_i^{pd} and π_i^{eq} are simulated with $\mathcal{S}_{\mathsf{P}}^{\mathsf{pd}}$ and $\mathcal{S}_{\mathsf{P}}^{\mathsf{eq}}$, as described in Steps 13 and 28 of Figure 12.
- $\mathcal{S}^{\mathsf{Corr}}_{\mathsf{dkg}}$: On the adaptive corruption of party P_i the simulator uses $\mathcal{S}_{\mathsf{R}}^{\mathsf{pd}}$ and $\mathcal{S}_{\mathsf{R}}^{\mathsf{eq}}$ to generate randomness r_i^{pd} respective r_i^{eq} that can be used to recompute the simulated proof, as in Step 34.

Under the assumption that Π_{pd} and Π_{eq} satisfies zero knowledge against adaptive corruption (Definition 9) this change is indistinguishable to $\mathcal{Z}$; H_3 is indistinguishable from H_2 and $\mathsf{H}_3 \approx \mathsf{H}_0$.

Hybrid H_4: *Trapdoored Group Elements.* The simulator trapdoors the group generators output by $\texttt{H}_{\mathbb{G}}$.

– *Modifications to* $\mathcal{S}_{\texttt{DKG}}$ *:*

- $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$: The elements $H_u := \tau_u G, H_v := \tau_v G, H_w := \tau_w G$ are generated according to Step 9a in Figure 11.

This change is perfectly indistinguishable; the distribution of H_u, H_v, H_w in H_4 and H_3 is uniform in $\mathbb{G}$.

Hybrid H_5: *Random Ciphertexts Exchanged Between Honest Parties.* In this game, the simulator samples the ciphertexts sent between honest parties uniformly and programs $\texttt{H}_{\texttt{sh}}$ on an adaptive corruption instruction, such that the ciphertexts decrypt to the correct shares.

– *Modifications to* $\mathcal{S}_{\texttt{DKG}}$ *:*

- $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$: Program the random oracle $\texttt{H}_{\texttt{sh}}$ as described in Step 10 of Figure 11. As of H_5, the emulation of $\texttt{H}_{\texttt{sh}}$ is fully implemented.
- $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$: For honest $P_i \in \mathcal{H}$ the ciphertexts sent to honest $P_j \in \mathcal{H}$ are sampled at random $\texttt{ct}_{i,j} \leftarrow_{\$} \{0,1\}^{4\mathsf{log}(q-1)}$ as in Step 14 of Figure 12.
- $\mathcal{S}^{\mathsf{Corr}}_{\mathsf{dkg}}$: Compose and return the state $\texttt{st}_i$ of the corrupted party as in H_3 with the modifications that the $\texttt{H}_{\texttt{sh}}$ is updated according to Step 35 of Figure 13.

The distribution of the ciphertexts is uniform in the ciphertext space in both H_4 and H_5, as the output of $\texttt{H}_{\texttt{sh}}$ is sampled randomly.

An environment that can distinguish H_5 from H_4 implies an adversary against the Decisional Diffie-Hellman, $\mathcal{B}(G, X, Y, Z)$ as in Definition 2:

1. Execute H_5 with $\mathcal{A}$ with the changes below.
   (a) Sample $P_i, P_j \leftarrow_{\$} \mathcal{P} \setminus \mathcal{C}$.
   (b) Define $K_i := r_i \cdot X$, $K_j := r_j \cdot Y$, $K_{i,j} := r_i \cdot r_j \cdot Z$.
   (c) Retrieve from $\mathcal{F}_{\mathsf{pki}}$ the secret key of each party $P_c \in \mathcal{C}$ and use this secret to encrypt shares from P_i, P_j to P_c. If P_i or P_j must complain about P_c, invoke the simulated prover for $\mathcal{R}_{\mathsf{dh}}$ to produce $\pi^{\mathsf{dh}}_{i,c}$ for statement $\phi^{\mathsf{dh}}_{i,c} = (G, K_i, K_c, K_{i,c})$.

(d) On a query of $(\mathsf{ssid}, i, j, K_{i,j})$ or $(\mathsf{ssid}, j, i, K_{i,j})$ to $\mathtt{H}_{\mathtt{sh}}$: stop the simulation and go to Step 2.

(e) If $\mathcal{A}$ corrupts one of P_i, P_j: rewind $\mathcal{A}$ and go to Step 1a.

2. Await $b' \leftarrow \mathcal{A}$.
3. Output $b^* := 1 - b'$.

Assume that $\mathcal{A}$ has non-negligible advantage $\epsilon(\cdot)$ in distinguishing H_5 from H_4. The advantage of the adversary can be expressed as below.

$$\mathsf{Adv}(\mathcal{A}) := \Pr[b' = 1 \wedge \mathsf{H}_5] - \Pr[b' = 1 \wedge \mathsf{H}_4] = \epsilon(\lambda) \tag{3}$$

Denote by E_{dh} the event that (G, X, Y, Z) is a Diffie-Hellman tuple and $\mathsf{E}_\perp$ be the event that the reduction aborts. If E_{dh} and $\neg\mathsf{E}_\perp$ then the interaction of $\mathcal{A}$ above is equivalent to H_4; $\mathcal{A}$ would output $b' = 0$ for a correct guess. Otherwise, if either $\neg\mathsf{E}_{\mathsf{dh}}$ or $\mathsf{E}_\perp$ then the interaction of $\mathcal{A}$ is equivalent to H_5; $\mathcal{A}$ would output $b' = 1$ for a correct guess. The advantage of the adversary can be expressed as below.

$$\begin{aligned}
\epsilon(\lambda) &= \Pr[b' = 1 \wedge \underbrace{(\neg\mathsf{E}_{\mathsf{dh}} \vee \mathsf{E}_\perp)}_{\mathsf{H}_5}] - \Pr[b' = 1 \wedge \underbrace{\mathsf{E}_{\mathsf{dh}} \wedge \neg\mathsf{E}_\perp}_{\mathsf{H}_4}] \\
&= \underbrace{\Pr[b' = 1 \wedge \neg\mathsf{E}_{\mathsf{dh}}]}_{=\frac{1}{2}\cdot\Pr[b'=1|\neg\mathsf{E}_{\mathsf{dh}}]} + \underbrace{\Pr[b' = 1 \wedge \mathsf{E}_\perp] - \Pr[b' = 1 \wedge \mathsf{E}_\perp \wedge \neg\mathsf{E}_{\mathsf{dh}}]}_{=\Pr[b'=1\wedge\mathsf{E}_\perp\wedge\mathsf{E}_{\mathsf{dh}}]\leq\frac{1}{2}\cdot\Pr[b'=1|\mathsf{E}_{\mathsf{dh}}]} - \underbrace{\Pr[b' = 1 \wedge \neg\mathsf{E}_\perp \wedge \mathsf{E}_{\mathsf{dh}}]}_{=1-\Pr[b'=0\vee\mathsf{E}_\perp\vee\neg\mathsf{E}_{\mathsf{dh}}]} \\
&= \frac{1}{2}\cdot\Pr[b' = 1|\neg\mathsf{E}_{\mathsf{dh}}] + \frac{1}{2}\cdot\Pr[b' = 1|\mathsf{E}_{\mathsf{dh}}] - 1 + \underbrace{\Pr[b' = 0 \vee \mathsf{E}_\perp \vee \neg\mathsf{E}_{\mathsf{dh}}]}_{\leq\frac{1}{2}} \\
&\leq \frac{1}{2}\cdot\Pr[b' = 1|\neg\mathsf{E}_{\mathsf{dh}}] + \frac{1}{2}\cdot\Pr[b' = 1|\mathsf{E}_{\mathsf{dh}}] := \mathsf{Adv}_{\mathsf{DDH}}(\mathcal{B}).
\end{aligned} \tag{4}$$

We see that the advantage of $\mathcal{B}$ against the DDH problem is therefore non-negligible if $\epsilon(\lambda)$ is non-negligible as was assumed. Therefore, no such $\mathcal{A}$ that can distinguish between H_5 and H_4 with non-negligible probability exists under the DDH assumption in $\mathcal{G}$ given that Π_{dh} is a zero knowledge NIZK (Def 9): $\mathsf{H}_5 \approx \mathsf{H}_4$.

Therefore, no such $\mathcal{A}$ that can distinguish between H_5 and H_4 with non-negligible probability exists under the DDH assumption in $\mathcal{G}$, and $\mathsf{H}_5 \approx \mathsf{H}_4$.

Hybrid H_6: *Functionality Samples Keys.* The functionality $\mathcal{F}$ samples and sends key pairs $(\mathsf{sk}, \mathsf{pk})$ to the simulator. The simulator ensures that the protocol outputs pk.

– *Modifications to $\mathcal{F}$:*
- As in the `DKG-init` interface of $\mathcal{F}_{\mathtt{dkg}}$: upon receipt of a query from all $P_j \in \mathcal{Q}$ in subsession ssid, the functionality $\mathcal{F}$ functionality samples a key pair $(\mathsf{sk}, \mathsf{pk})$. The interface is not yet fully implemented; as in H_6, $\mathcal{F}$ outputs the pair $(\mathtt{DKG\text{-}pk}, \mathsf{ssid}, (\mathsf{sk}, \mathsf{pk}))$ to $\mathcal{A}^*$.

– *Modifications to $\mathcal{S}_{\mathtt{DKG}}$:*
- $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$: The simulator receives $(\mathtt{DKG\text{-}pk}, \mathsf{ssid}, (\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}))$ from $\mathcal{F}$ upon the initialized of the last dummy party, then immediately upon receiving $\{\mathsf{ssid}, j, \mathtt{BAD}_j\}_{P_j\in\mathcal{Q}}$, runs: $\mathcal{E}^{(s,w)}_{\mathsf{vss}}(\mathsf{ssid}, i, \tau_w^{-1}, \{(0, (\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}} - \hat{s}))\} \cup \{(j, 0)\}_{P_j\in(\mathcal{Q}\cap\mathcal{C})\cup\mathcal{I}})$, with $\hat{s} := \sum_{P_j\in\mathcal{Q}} s_{j,0}$

The random ciphertexts reveal no information about the shares exchanged between the honest parties; therefore, this modification is perfectly indistinguishable unless inconsistencies arise with respect to the VSS commitments $\{A_{i,l}\}_{l=0}^t$ of the shares. We proceed to argue such inconsistencies does not occur.

Remark, that the computed shift polynomial, $\Delta(X)$, will be equal in both calls, as it is obtained though Lagrange interpolation over the same set of points. Thus, calling the two processes $\mathcal{E}^{(s,w)}_{\mathsf{vss}}$ effectively causes an equivocating between (s, w) on the form,

$$s^*_{i,0} := s_{i,0} + r_\Delta, \quad \mathbf{r}^*_{s,i} := \mathbf{r}_{s,i} + \mathbf{r}_\Delta$$

$$w^*_{i,0} := w_{i,0} - \tau_w^{-1} r_\Delta, \quad \mathbf{r}^*_{w,i} := \mathbf{r}_{w,i} - \tau_w^{-1}\mathbf{r}_\Delta$$

First we argue that upon these calls, the secret key output by the protocol is indeed $\mathsf{sk}_{\mathcal{F}_{\text{dkg}}}$. The secret key share of party P_i is modified to be $s^*_{i,0} = s_{i,0} + r_\Delta = s_{i,0} + (\mathsf{sk}_{\mathcal{F}_{\text{dkg}}} - \hat{s}) = s_{i,0} + \mathsf{sk}_{\mathcal{F}_{\text{dkg}}} - \sum_{P_j \in \mathcal{Q}} s_{j,0} = \mathsf{sk}_{\mathcal{F}_{\text{dkg}}} - \sum_{P_j \in \mathcal{Q} \setminus \{P_i\}} s_{j,0}$. Thereby the new secret key output by the protocol is $s^* := s^*_{i,0} + \sum_{P_j \in \mathcal{Q} \setminus \{P_i\}} s_{j,0} = \mathsf{sk}_{\mathcal{F}_{\text{dkg}}} - \sum_{j \in \mathcal{Q} \setminus \{P_i\}} s_{j,0} + \sum_{P_j \in \mathcal{Q} \setminus \{P_i\}} s_{j,0} = \mathsf{sk}_{\mathcal{F}_{\text{dkg}}}$.

Next, we note that the updated coefficient vectors $\mathbf{s}^*, \mathbf{r}^*$, computed as described above, satisfy the PVSS Reshare Correctness property by a straightforward extension the argument to these vectors. Additionally, the shift polynomial evaluates to zero at all points j such that $P_j \in \mathcal{Q} \cap \mathcal{C}$, ensuring that the threshold shares sent to any corrupt party are fixed. Thus, the view of $\mathcal{Z}$ is unchanged.

Hybrid H_7: *Rigged Group Elements Output by* $\mathtt{H}_{\mathbb{G}}$. In this game, the simulator programs the random oracle, $\mathtt{H}_{\mathbb{G}}$, such that the group element F is computed from $\mathsf{pk}_{\mathcal{F}_{\text{dkg}}}$.

– *Modifications to* $\mathcal{S}_{\mathtt{DKG}}$ *:*

- $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$*:* The simulation programs the random oracle $\mathtt{H}_{\mathbb{G}}$ as in in Step 9 of Figure 11, the random oracle is programmed to output *rigged* group elements F after the receipt of $(\mathsf{sk}_{\mathcal{F}_{\text{dkg}}}, \mathsf{pk}_{\mathcal{F}_{\text{dkg}}})$ from $\mathcal{F}$. The instance $\mathtt{H}_{\mathbb{G}}$, and thus both the emulation of $\mathcal{F}_{\mathsf{ro}}$ and the entire simulator $\mathcal{S}^{\mathcal{F}}_{\mathsf{dkg}}$, are fully implemented.

This change is perfectly indistinguishable; the distribution of the output of $\mathtt{H}_{\mathbb{G}}$ in H_7 is uniform in $\mathbb{G}$ since the scalar α is sampled uniformly in Step 9(b)i, and is therefore exactly the distribution of the output of $\mathtt{H}_{\mathbb{G}}$ in H_6. Thus, the view of $\mathcal{Z}$ is unchanged.

Hybrid H_8: *Hit Parties Key Shares.* In this game, $\mathcal{F}$ upon corruption of a party P_i ensures that the secret key share output to $\mathcal{A}$ on an adaptive corruption is this key share delivered to the dummy parties.

– *Modifications to* $\mathcal{S}_{\mathtt{DKG}}$*:* As in Step 33 of Figure 13, the simulator $\mathcal{S}^{\mathtt{Corr}}_{\mathsf{dkg}}$ calls $\mathcal{E}^{(s,v)}_{\mathsf{vss}}$ to ensure that the secret shares returned to $\mathcal{A}$ are $(\mathsf{sk}_i, u_i, v_i)$ such that sk_i was received from $\mathcal{F}$.

The random ciphertexts reveal no information about the shares exchanged between the honest parties; therefore, this modification is perfectly indistinguishable unless inconsistencies arise with respect to other information exchanged about the shares. We proceed to argue such inconsistencies does not occur. The modified threshold shares obtained by $\mathcal{S}^{\mathtt{Corr}}_{\mathsf{dkg}}$'s call to $\mathcal{E}^{(s,v)}_{\mathsf{vss}}$ satisfy the both the VSS commitments $\{A_{i,k}\}_k$ and the aggregated shares commitments $\{B_j\}_{P_j \in \mathcal{V}}$, as a direct consequence of the correctness claim of $\mathcal{E}^{(s,v)}_{\mathsf{vss}}$. Thus, this equivocating is consistent with all information exchanged about the shares.

Consequently, it remains to argue that the secret share of a corrupted party P_i is indeed sk_i received from the functionality, that is $s^*_i = \mathsf{sk}_i$. We have that,

$$
\begin{aligned}
s^*_i &= \mathtt{SShare}(s^*_{i,0}; \mathbf{r}^*_{s,i})[i] + \sum_{P_j \in \mathcal{V}' \setminus \{P_i\}} \mathtt{SShare}(s_{j,0}; \mathbf{r}_{s,j})[i] \\
&= \mathtt{SShare}(s_{i,0} + r_\Delta; \mathbf{r}_{s,i} + \mathbf{r}_\Delta)[i] + \sum_{P_j \in \mathcal{V} \setminus \{P_i\}} \mathtt{SShare}(s_{j,0}; \mathbf{r}_{s,j})[i] \\
&= \mathtt{SShare}(s_{i,0}; \mathbf{r}_{s,i})[i] + \mathtt{SShare}(r_\Delta; \mathbf{r}_\Delta)[i] + \sum_{P_j \in \mathcal{V}' \setminus \{P_i\}} \mathtt{SShare}(s_{j,0}; \mathbf{r}_{s,j})[i] \\
&= \mathtt{SShare}(r_\Delta; \mathbf{r}_\Delta)[i] + \sum_{P_j \in \mathcal{V}'} \mathtt{SShare}(s_{j,0}; \mathbf{r}_{s,j})[i] \\
&:= (\mathsf{sk}_i - s_i) + s_i = \mathsf{sk}_i
\end{aligned}
$$

Hybrid H_9: *Targeting: Share* $(s, \alpha, 0)$ *Instead of* $(\mathsf{sk}_{\mathcal{F}_{\text{dkg}}}, 0, 0)$. In this game, the simulator modifies the state of an honest party P_T, referred to as the *target party*, such that the protocol outputs the public key $\mathsf{pk}_{\mathcal{F}_{\text{dkg}}}$.

– *Modifications to* $\mathcal{S}_{\mathtt{DKG}}$ *:*

- $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$: The protocol simulator no longer follows the process detailed in H_6 to ensure that the output of the subsession is $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$. Instead, as in Step 23-28 of Figure 12, upon receiving $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ from $\mathcal{F}$ the simulator instead shifts the secrets shared by P_T to $(s^*_{T,0}, \alpha, 0, w^*_{T,0})$. This change, by the programming of the random oracle $\mathtt{H}_{\mathbb{G}}$, ensures the output of the protocol to be $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$. The simulator $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$ is fully implemented.
- $\mathcal{S}^{\mathtt{Corr}}_{\mathsf{dkg}}$: Upon the t-th corruption or if the target party is corrupted, the corruption simulator behaves as in Step 31 of Figure 13, to equivocate the sharing of P_T.
  As of this hybrid, the simulator does not follow the *retargeting* process of Step 32 in Figure 13.

First we show that the public key output by the protocol is indeed $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ received from the functionality. The simulation of the protocol, $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$, modifies the state of the target party, such that it shared threshold shares of the secrets $(s_{i,0}+\delta_s, \alpha, 0, w_{i,0}-\tau_w^{-1}\delta_s-\tau_w^{-1}\tau_u\alpha)$, where $\delta_s = (s-\hat{s})$ and $\hat{s} = \sum_{P_j\in\mathcal{V}} s_{j,0}$, causing the aggregated share commitments $\{B_j\}$ to be commitments to these shares. Thereby the public key, computed by Lagrange interpolation over these commitments, will be on the form $\mathsf{pk} = (\hat{s} + (s-\hat{s}))G + \alpha F = (s+\alpha\tau_F)G = (s+\alpha\alpha^{-1}(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}} - s)) = \mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}G$ as desired. Since all parties have, provably, shared threshold shares of the secrets $(s_{j,0}, 0, 0, w_{j,0})$ except the target party whose state was modified by $\mathcal{S}^{\Pi}_{\mathsf{dkg}}$.

Next we argue that upon the t-th corruption or the corruption of the target party, all parties states are consistent[11] with the protocol. It is clear by inspection that upon the call to $\mathcal{E}^{(s,u,w)}_{\mathtt{vss}}$ the target party holds threshold shares of the secrets $((s^*_{i,0} + \mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}} - s), 0, 0, (w^*_{i,0} + \tau_w^{-1}(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}} - s)))$, thus the state of the target party is consistent with the protocol.

Finally, we argue that the modifications in H_9 are indistinguishable to $\mathcal{Z}$. The random ciphertexts reveal no information about the shares exchanged between the honest parties; therefore, this modification is perfectly indistinguishable unless inconsistencies arise with respect to other information exchanged about the shares. We proceed to argue such inconsistencies does not occur. By the claim of correctness of Definition 11 execution of the processes $\mathcal{E}^{(s,w)}_{\mathtt{vss}}, \mathcal{E}^{(u,w)}_{\mathtt{vss}}, \mathcal{E}^{(s,u,w)}_{\mathtt{vss}}$ are indistinguishable from the view of an adversary having access the VSS commitments, $\{A_{T,l}\}_{l=0}^{T}$ and up to t threshold shares, $\{\mathsf{sh}_{T,j}\}_{P_j\in\mathcal{Q}\cap\mathcal{C}}$ as long as $\Delta(j) = 0$ at all $P_j \in \mathcal{Q}\cap\mathcal{C}$. The only other elements in the view of $\mathcal{Z}$ that contains information about the threshold shares exchanged between honest parties is the commitments to the aggregated shares. Thus, it remains to argue that this change is indistinguishable also given the elements $\{B_j\}_{P_j\in\mathcal{P}}$. Until the t-th corruption less than t threshold shares of the honest parties' secrets are leaked to $\mathcal{Z}$. Thus, neither the secrets $(s_{j,0}, u_{j,0}, v_{j,0}, w_{j,0})$ nor the shares exchange between honest parties are fixed. Since exchanged shares are not fixed, nor are the aggregated shares derived from them. Consequently, the values of which B_i commits to are not determined by the view of $\mathcal{Z}$. Thus, provided that the updated values still satisfy the commitment, they can be modified without altering the view of $\mathcal{Z}$. It remains to argue that, after $\mathcal{S}^{\mathtt{Corr}}_{\mathsf{dkg}}$ runs $\mathcal{E}^{(s,u,w)}_{\mathtt{vss}}$, the updated threshold shares still satisfy the commitments $\{B_j\}_{P_j\in\mathcal{V}}$. We have that $(s^*_j, u^*_j, v^*_j, \cdot) \leftarrow \mathtt{AggSh}(\{\mathsf{sh}^*_{j,i}\}_{P_j\in\mathcal{V}})$ should be an opening to $B_j := s_jG + u_jF + v_jH_v$, for all $P_j \in \mathcal{V}$. This holds since, $s^*_j + u^*_j\tau_F + v^*_j\tau_v = s_j + u_j\tau_F + v_j\tau_v$. We leave this as a claim, as the computation is length and straightforward.

Hybrid H_{10}: *Retargeting When the Target Party is Corrupted.* In this game, the simulator allows for up to $t-1$ corruptions before correcting the states of all of the parties.

– *Modifications to* $\mathcal{S}_{\mathtt{DKG}}$ *:*

- $\mathcal{S}^{\mathtt{Corr}}_{\mathsf{dkg}}$: The simulator behaves as in H_9, except when the target party, P_T, gets corrupted while fewer than t parties are corrupted, in this case the simulator follows the *retargeting* process of Step 32. As of this hybrid, the simulator is fully implemented.

The random ciphertexts reveal no information about the shares exchanged between the honest parties; therefore, this modification is perfectly indistinguishable unless inconsistencies arise with respect to other information exchanged about the shares. We proceed to argue such inconsistencies does not occur.

[11] With "consistent" we mean that the secret state of an honest party upon shared to the environment without any equivocation does not cause the environment to distinguish between the hybrid games

All reasoning in H_9 about indistinguishability under calls to $\mathcal{E}^{(\cdot)}_{\mathtt{vss}}$, remains valid until t threshold shares of an honest party's secrets has been revealed to the environment, and thus continues to hold in this hybrid. Consequently, it remains to argue that (1) After the retargeting process the modified threshold shares satisfies the commitments, $\{B_j\}_{P_j\in\mathcal{V}}$ (2) the retargeting process preserves the invariant that there is at most one party with an inconsistent state —namely, the new target party ($P_{\tilde{T}}$). To first bullet states that, $(s^*_j, u^*_j, v^*_j, \cdot) \leftarrow \mathtt{AggSh}(\{\mathsf{sh}^*_{i,j}\}_{P_j\in\mathcal{V}})$ should be a valid opening to $B_j := s_jG + u_jF + v_jH_v$. This holds since, $s^*_j + u^*_j\tau_F + v^*_j\tau_v = s_i + u_i\tau_F + v_i\tau_v$. We leave this as a claim, as the computation is length and straightforward; it follows from the mirrored update on the threshold shares of the old and new target party. By inspection, after the retargeting process $u_{T,0} = 0$ and $u_{\tilde{T},0} = \alpha$, thus the state of the previous target is consistent with the protocol and the $P_{\tilde{T}}$ is the new inconsistent party. This ensures that the equivocating upon the t-th corruption ensures that all states become consistent, as argued in H_9

Hybrid H_{11}: *Functionality Samples Secret Key Shares.* In this game, $\mathcal{F}$ chooses the secret keys of all the parties and $\mathcal{S}_{\mathtt{DKG}}$ ensures that the secret key share output to $\mathcal{A}$ on an adaptive corruption is this key share.

– *Modifications to $\mathcal{F}$:*

- As in the `DKG-shares` interface of $\mathcal{F}_{\mathtt{dkg}}$, $\mathcal{F}$ interpolates the polynomial $S(X)$ from sk and the corrupted party shares received from $\mathcal{A}^*$, thus defines the secret key shares of the honest parties. In H_{11} this interface is fully implemented
- The functionality $\mathcal{F}$ no longer forwards key shares $\{\mathsf{sk}_i\}_{P_i\in\mathcal{Q}\setminus\mathcal{C}}$ received from $\mathcal{A}^*$ in a query to $(\mathtt{DKG\text{-}deliverkey}, \mathsf{ssid}, i)$. Instead, the shares are determined by `DKG-shares` and retrieved from $\mathtt{shares}[i][\mathsf{ssid}]$. In H_{11} this interface is fully implemented

In H_{10}, the honest party's key shares are computed as the sum of the VSS shares it receives. Despite adversarial chosen VSS shares from the corrupt parties, the resulting key share is uniformly random due to the honest parties sampling their secret polynomials at random. In H_{11}, the key shares are computed via Lagrange interpolation from a fixed set of points; the adversarial chosen shares of the corrupt parties and the uniformly random secret key sampled by the functionality. Since the adversary chooses at most $t-1$ fixed points, the honest parties shares are still uniformly random.

Thus, the distribution of the honest parties' secret key shares is identical in H_{11} and H_{10}, and the view of $\mathcal{Z}$ is unchanged.

Hybrid H_{12}: *Eliminate Functionality Leakage.* In this game, $\mathcal{F}$ no longer sends $\mathsf{sk}_{\mathcal{F}_{\mathtt{dkg}}}$ to $\mathcal{S}_{\mathtt{DKG}}$.

– *Modifications to $\mathcal{F}$:*

- As in the `DKG-init` interface of $\mathcal{F}_{\mathtt{dkg}}$, upon receipt of a query from all $P_j \in \mathcal{Q}$ in subsession ssid, the functionality $\mathcal{F}$ outputs $(\mathtt{DKG\text{-}pk}, \mathsf{ssid}, \mathsf{pk}_{\mathcal{F}_{\mathtt{dkg}}})$, no longer leaking $\mathsf{sk}_{\mathcal{F}_{\mathtt{dkg}}}$, to $\mathcal{A}^*$.
- As of this change, $\mathcal{F} = \mathcal{F}_{\mathtt{dkg}}$, so the functionality is fully implemented.

This change is perfectly indistinguishable as the simulator $\mathcal{S}_{\mathtt{DKG}}$ did not use $\mathsf{sk}_{\mathcal{F}_{\mathtt{dkg}}}$ for the simulation in H_{11}

Hybrid H_{13}: *Ideal World.* This hybrid corresponds to the ideal execution, where the environment $\mathcal{Z}$ invokes $\mathcal{F}_{\mathtt{dkg}}$ of Figure 8 with $\mathcal{S}_{\mathtt{DKG}}$. The change is syntactic. As of H_{12}, both $\mathcal{F}_{\mathtt{dkg}}$ and $\mathcal{S}_{\mathtt{DKG}}$ are fully implemented.

Consider the random variable $\mathsf{EXEC}_{\mathcal{F}_{\mathtt{dkg}},\mathcal{S}_{\mathtt{DKG}},\mathcal{Z}}(z)$, which denotes the output of an interaction with the functionality $\mathcal{F}_{\mathtt{dkg}}$, simulator $\mathcal{S}_{\mathtt{DKG}}$ and environment $\mathcal{Z}$ on input z. The set of all interaction withing this hybrid corresponds to the ensemble:

$$\{\mathsf{EXEC}_{\mathcal{F}_{\mathtt{dkg}},\mathcal{S}_{\mathtt{DKG}},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}.$$

Guaranteed Output Delivery. The protocol requires all parties in subsession ssid to broadcast a (potentially empty) message in both Round 1 and Round 2, and parties wait to receive t valid commitments before computing and outputting the public key. Guaranteed output delivery holds if $\mathcal{F}_{\mathtt{bc}}$ receives enough activations, assuming an honest majority ($t < \frac{n}{2}$), and that $\mathcal{Q} = \mathcal{P}$ in ssid.

If all $n-|\mathcal{C}'|$ honest (dummy) parties invoke the functionality $\mathcal{F}_{\mathsf{dkg}}$, then $\mathcal{S}_{\mathtt{DKG}}$ receives $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ and simulates all of the honest Round 1 messages. If all $|\mathcal{C}'|$ corrupted parties activate $\mathcal{F}_{\mathsf{bc}}$ in Round 1, then $\mathcal{S}_{\mathtt{DKG}}$ proceeds to Round 2 and simulates the $n-|\mathcal{C}'|$ Round 2 honest party messages. If all $|\mathcal{C}'|$ corrupted parties activate $\mathcal{F}_{\mathsf{bc}}$ in Round 2, then $\mathcal{S}_{\mathtt{DKG}}$ can process the complaints (as in Step 19) and will activate $\mathcal{F}_{\mathsf{dkg}}$ with a `DKG-shares` query (Step 21). After this step the $n-|\mathcal{C}'|$ honest party messages are simulated. Since all honest parties complete the protocol—under the assumption that all broadcasted messages are delivered and an honest majority—it follows that at least $n-|\mathcal{C}'| \geq n-t \geq n-\frac{n}{2} = \frac{n}{2} > t$ parties will broadcast valid commitments to their aggregated shares in **Round 3**. Once these messages are delivered by $\mathcal{F}_{\mathsf{bc}}$ the simulator queries `DKG-deliverkey` (Step 29).

More formally; given an honest majority, that all $P_j \in \mathcal{Q} = \mathcal{P}$ broadcasts a—potentially empty—message in the first and second rounds, and that all broadcasted messages are delivered $\mathcal{S}_{\mathtt{DKG}}$ never invokes `DKG-iabort`. The simulator always queries `DKG-deliverkey`, causing output to be delivered to the honest parties. □

5 Committed Distributed Key Generation with Identifiable Abort

In Section 5.1 we present the committed distributed key generation functionality $\mathcal{F}_{\mathsf{cdkg}}$. Most importantly, this functionality differs from $\mathcal{F}_{\mathsf{dkg}}$ in that it outputs perfectly hiding commitments to the secret key shares of each party as, essentially, a set of public keys to be used in a higher-level protocol. In Section 5.2 we present the protocol $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ that enables parties to generate multiple independent threshold-shared keys after a one time setup. Theorem 3 proves security in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model under the DDH assumption. In Section 5.3 we present the guaranteed output variant of this protocol $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$, and Theorem 4 similarly proves security in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model under the DDH assumption.

5.1 The Committed DKG Functionality

The committed DKG functionality $\mathcal{F}_{\mathsf{cdkg}}$ in Figure 14 is a variant of $\mathcal{F}_{\mathsf{dkg}}$ that additionally outputs commitments to the key shares, as is highly useful for protocols written in the $\mathcal{F}_{\mathsf{cdkg}}$-hybrid model. Its use is demonstrated in Section 6 via our incredibly simple protocol for threshold Schnorr signatures in the $\mathcal{F}_{\mathsf{cdkg}}$-hybrid model that crucially relies on these commitments for partial signature verification. The differences from $\mathcal{F}_{\mathsf{dkg}}$ are highlighted and marked with $\star$, where relevant, for ease of comparison.

Figure 14: $\mathcal{F}_{\mathsf{cdkg}}(\mathcal{G}, t)$.
Ideal Committed DKG Functionality.

The committed distributed key generation functionality is parameterized by the group $\mathcal{G} := (\mathbb{G}, G, q)$, and the threshold $t < n$. The functionality interacts with the n parties in $\mathcal{P}$ and the ideal adversary $\mathcal{A}^*$ via the queries below, all of which implicitly contain a session identifier sid. Upon invocation, the functionality receives the set of corrupted parties $\mathcal{C}$ from $\mathcal{A}^*$ and maintains the following initially empty tables `keys`[], `status`[], `shares`[][], `coms`[][][], and `rands`[][] .

Initialization.

On (`CDKG-init`, i) **from** $P_i \in \mathcal{P}$**.** If not yet recorded, record and send (`CDKG-init`, $i, \mathcal{P}$) to $\mathcal{A}^*$. Once queried by all $P_i \in \mathcal{P} \setminus \mathcal{C}$, open the following interfaces to $\mathcal{A}^*$ until there is a record (`CDKG-CK`, $[\cdots]$):

- $\star$ **On** (`CDKG-newCK`, c) **from** $\mathcal{A}^*$**.** If there is no record $(c, [\cdots])$ then sample $\tau_c \leftarrow_{\$} \mathbb{Z}_q$, define $H_c := \tau_c \cdot G$, record (c, H_c, τ_c) and send (`CDKG-newCK`, H_c) to $\mathcal{A}^*$.
- $\star$ **On** (`CDKG-setCK`, c) **from** $\mathcal{A}^*$**.** If there is a record (c, H_c, τ_c) then record (`CDKG-CK`, τ_c, H_c). Output (`CDKG-initialized`) to $\mathcal{P}$.

$\star$ **On** (`CDKG-CK`) **from** $P_i \in \mathcal{P}$**.** If there is a record (`CDKG-CK`, τ, H) then output (`CDKG-CK`, H) to P_i.

Key Generation.

On $(\texttt{CDKG-gen}, \mathsf{ssid}, i, \mathcal{Q})$ **from** $P_i \in \mathcal{P}$**.** Ignore the query if there exists a record $(\mathsf{ssid}, \mathcal{Q}')$ with if $\mathcal{Q}' \neq \mathcal{Q}$ or if $i \notin \mathcal{Q} \subseteq \mathcal{P}$. If there does not exist a record $(\mathsf{ssid}, \cdot)$ then record and send $(\mathsf{ssid}, \mathcal{Q})$ to $\mathcal{A}^*$. Record and send (ssid, i) to $\mathcal{A}^*$. If (ssid, j) is recorded for exactly $t - |\mathcal{C}|$ parties $P_j \in \mathcal{Q}$ sample $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$, define $\mathsf{pk} := \mathsf{sk} \cdot G$, record $\texttt{keys}[\mathsf{ssid}] := (\mathsf{pk}, \mathsf{sk})$.

⋆ For each $P_i \in \mathcal{Q} \setminus \mathcal{C}$: Compute $D_i := d_i \cdot G$ for $d_i \leftarrow_\$ \mathbb{Z}_q$.
Record $\texttt{coms}[\mathsf{ssid}][i] := D_i$ and $\texttt{shares}[i][\mathsf{ssid}] := d_i$.

Send $(\texttt{CDKG-pk}, \mathsf{ssid}, \mathsf{pk}, \texttt{coms}[\mathsf{ssid}])$ to $\mathcal{A}^*$.

On $(\texttt{CDKG-shares}, \mathsf{ssid}, \{(j, \mathsf{sk}_j, r_j)\}_{P_j \in \mathcal{C}})$ **from** $\mathcal{A}^*$**.** Ignore if there is no record $\texttt{keys}[\mathsf{ssid}] = (\mathsf{pk}, \mathsf{sk})$. Retrieve $(\mathsf{ssid}, \mathcal{Q})$. If $\mathsf{sk}_j \notin \mathbb{Z}_q$ for any $P_j \in \mathcal{Q} \cap \mathcal{C}$ then define $\mathsf{sk}_j := 0$. Sample a random degree $t-1$ polynomial that interpolates $(0, sk)$ and (j, sk_j) for all $j \in \mathcal{Q} \cap \mathcal{C}$. For each $P_i \in \mathcal{P}$ record $\texttt{shares}[i][\mathsf{ssid}] := S(i)$. Record $\texttt{status}[\mathsf{ssid}] := \mathsf{ready}$.

⋆ For all $P_j \in \mathcal{Q} \cap \mathcal{C}$: if $r_j \in \mathbb{Z}_q$ set $D_j := \mathsf{sk}_j \cdot G + r_j \cdot H$, else $D_j := \perp$.
Record $\texttt{coms}[\mathsf{ssid}][j] := D_j$ and $\texttt{rands}[j][\mathsf{ssid}] := r_j$.

⋆ For all $P_j \in \mathcal{Q} \setminus \mathcal{C}$: overwrite $\texttt{rands}[i][\mathsf{ssid}] := \tau^{-1}(\texttt{rands}[i][\mathsf{ssid}] - \mathsf{sk}_i)$.

On $(\texttt{CDKG-deliver}, \mathsf{ssid})$ **from** $\mathcal{A}^*$**.** Ignore the query unless $\texttt{status}[\mathsf{ssid}] = \texttt{ready}$. Overwrite $\texttt{status}[\mathsf{ssid}] := \top$. Retrieve $(\mathsf{ssid}, \mathcal{Q})$ and $(\mathsf{pk}, \cdot) := \texttt{keys}[\mathsf{ssid}]$. Send $(\texttt{CDKG-keys}, \mathsf{ssid}, \mathsf{pk}, \texttt{shares}[i][\mathsf{ssid}], \texttt{rands}[i][\mathsf{ssid}], \texttt{coms}[\mathsf{ssid}])$ to all $P_i \in \mathcal{Q} \setminus \mathcal{C}$.

On $(\texttt{CDKG-iabort}, \mathsf{ssid}, j)$ **from** $\mathcal{A}^*$**.** Retrieve $(\mathsf{ssid}, \mathcal{Q})$ and ignore the query if $j \notin \mathcal{Q} \cap \mathcal{C}$ or if $\texttt{status}[\mathsf{ssid}] = \top$. Record $\texttt{status}[\mathsf{ssid}] := \perp$. Send $(\texttt{CDKG-abort}, \mathsf{ssid}, j)$ to all $P_i \in \mathcal{Q} \setminus \mathcal{C}$.

On $(\texttt{CDKG-corrupt}, i)$ **from** $\mathcal{A}^*$**.** Add $P_i \to \mathcal{C}$ and send $(\texttt{CDKG-corrupt}, i, \texttt{shares}[i][:], \texttt{rands}[i][:])$ to $\mathcal{A}^*$, including entries for all ssid such that $\texttt{status}[\mathsf{ssid}] \in \{\texttt{ready}, \top, \perp\}$.

Note: Differences between $\mathcal{F}_{\mathsf{cdkg}}$ *and* $\mathcal{F}_{\mathsf{dkg}}$ *are highlighted.*

Commitments. The primary difference of our committed DKG functionality is that $\mathcal{F}_{\mathsf{cdkg}}$ outputs Pedersen commitments to the key shares of the parties. The functionality allows $\mathcal{A}^*$ to rejection sample the commitment key H during the initialization phase of the functionality, corresponding to interfaces ($\texttt{CDKG-newCK}$, $\texttt{CDKG-setCK}$); the commitment key is biased but $\mathcal{A}^*$ does not know the trapdoor for the commitments. These commitments must be computationally binding for $\mathcal{A}^*$.

The functionality leaks the honest party commitments $\{D_i\}_{P_i \in \mathcal{Q} \setminus \mathcal{C}}$ to the honest party key shares along with the public key pk to $\mathcal{A}^*$. This is, notably, before the functionality has received the corrupted party key shares from $\mathcal{A}^*$, and therefore before the functionality has actually defined the honest party key shares $\{\mathsf{sk}_i\}_{P_i \in \mathcal{Q} \setminus \mathcal{C}}$. This modeling allows us to save on communication, as will be discussed in Section 5.2.

As the functionality $\mathcal{F}_{\mathsf{cdkg}}$ knows the trapdoor τ for the commitment key $H = \tau \cdot G$, the functionality can output the commitments $\{D_i\}_{P_i \in \mathcal{Q} \setminus \mathcal{C}}$ before $\mathcal{A}^*$ queries $\texttt{CDKG-shares}$, then simply equivocate the commitments once $\mathcal{A}^*$ makes this query. Otherwise, the functionality is exactly the same as $\mathcal{F}_{\mathsf{dkg}}$ in Section 4.

5.2 Two Round Committed DKG Protocol with Identifiable Abort

For a graphical depiction of the communication flow of $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ see Figure 2. Intuitively, the difference between $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ and $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ are the following:

1. **Identifiable Abort**: This protocol targets identifiable abort; if parties cannot reconstruct the output they simply output the identifier of a corrupted party. As parties wait until receipt of messages from all of the parties executing the subsession, the parties will agree on the set of qualified parties and will output the smallest identifier j of a party $P_j \in \mathcal{Q} \setminus \mathcal{V}$.
2. **Round Complexity**: As parties do not have the goal of output reconstruction if a verifying complaint is issued in Round 2, parties *either* broadcast their set of complaints $\mathtt{BAD}_i$ *or* proceed to compute and broadcast their Round 2 message $(B_i, \pi_i^{\mathsf{eq}}, D_i)$, which is essentially the Round 3 message of $\Pi_{\mathsf{dkg}}^{\mathsf{go}}$.
3. **Committed:** Parties broadcast a pedersen commitment $D_i := (s_i, r_{i,D}) \cdot \mathbf{G}$ to their aggregate key share in Round 2, and prove that this commitment is consistent with the other commitments B_i, C_i via $\mathcal{F}_{\mathsf{azk}}$.

Straightline Extractable Proofs via $\mathcal{F}_{\mathsf{azk}}$. As we prove security for a full corruption threshold, i.e., $t < n$, our simulator can no longer reconstruct the corrupted party secrets from the $|\mathcal{Q} \setminus \mathcal{C}| \geq t$ shares received by the simulated honest parties $P_i \in \mathcal{Q} \setminus \mathcal{C}$. This then prevents the simulator from querying $\mathcal{F}_{\mathsf{cdkg}}$ with `CDKG-shares` that include the secret key shares of the corrupted parties. Instead, we replace extracting from the honest majority VSS with a single straightline extractable proof of knowledge. Our protocol requires that parties prove their commitments are consistent with the aggregate shares received from the qualified parties in $\mathcal{Q}$, i.e., parties must prove that $D_i = (s_i, r_{i,D}) \cdot \mathbf{G}$, $B_i = (s_i, u_i, v_i) \cdot \mathbf{F}$, and $C_i = (s_i, u_i, v_i, w_i) \cdot \mathbf{H}$ for $(s_i, u_i, v_i, w_i) := \mathtt{AggSh}(\{\mathsf{sh}_{j,i}\}_{P_j \in \mathcal{Q}})$. This is split into two pieces, parties prove consistency of B_i, C_i with π_i^{eq}, and prove consistency of B_i, D_i with $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$. As discussed, $\mathcal{F}_{\mathsf{azk}}$ models straightline extractable NIZKPoKs. This way, the simulator can extract $\{(\mathsf{sk}_j, r_{j,D})\}_{P_j \in \mathcal{Q} \cap \mathcal{C}}$ from $\mathcal{F}_{\mathsf{azk}}$ and query `CDKG-shares` with those extracted values. We note that, absent the committed property, parties could equivalently query $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eq}})$ on the statement ϕ_i^{eq} to instead prove consistency of B_i, C_i.

Figure 15: $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}(\mathcal{G}, t)$.
Committed DKG Protocol with Identifiable Abort.

The DKG protocol parameterized by a group $\mathcal{G} := (\mathbb{G}, G, q)$ and threshold $t < n$ is executed by the n parties in $\mathcal{P}$ with access to:

- Functionalities $\mathcal{F}_{\mathsf{bc}}$, $\mathcal{F}_{\mathsf{pki}}$, $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{pd}})$,
- $\mathcal{F}_{\mathsf{ro}}$ instances $\mathtt{H}_{\mathbb{G}}$ and $\mathtt{H}_{\mathtt{sh}}$
- NIZK $\Pi_{\mathsf{dh}} = (\mathsf{Prv}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}, \mathsf{Ver}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}})$ ZK and S for $\mathcal{R}_{\mathsf{dh}}$ (Defs 7, 9 10)
- NIZK $\Pi_{\mathsf{eq}} = (\mathsf{Prv}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}, \mathsf{Ver}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}})$ aZK and SS for $\mathcal{R}_{\mathsf{eq}}$ (Defs 7, 9 10)

$\star$ Additional functionality instance: $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$

Below is the code of party $P_i \in \mathcal{P}$:
The contents of the state of P_i are highlighted and flushed right.

Initialization.

On $(\mathtt{CDKG\text{-}init}, i)$**:** k_i^{pki}

1. $K_i := k_i^{\mathsf{pki}} \cdot G$; $k_i^{\mathsf{pki}} \leftarrow_{\$} \mathbb{Z}_q$; $r_i \leftarrow_{\$} \{0,1\}^{\lambda}$
2. Query $(\mathtt{PKI\text{-}register}, i, K_i, k_i^{\mathsf{pki}})$ to $\mathcal{F}_{\mathsf{pki}}$.
3. Broadcast $(\mathtt{CDKG\text{-}init}, i, r_i)$ to $\mathcal{P}$.

On (j, r_j) **from all** $P_j \in \mathcal{P}$**:**

4. For $\mathsf{id} \in \{\mathsf{com}, u, v, w\}$: Define $H_{\mathsf{id}} := \mathtt{H}_{\mathbb{G}}(\mathsf{id}, (j_1, r_{j_1}, \ldots, j_n, r_{j_n}))$.

5* $\mathbf{G} := (G, H_{\mathsf{com}})$

6. $\mathbf{H} := (G, H_u, H_v, H_w)$

7. Output (CDKG-initialized).

Commitment Key.

On (CDKG-CK)**:** Output $(\texttt{CDKG-CK}, H_{\mathsf{com}})$, if the value is defined.

Key Generation.

On $(\texttt{CDKG-init}, \mathsf{ssid}, i, \mathcal{Q})$**:** $s_{i,0}, w_{i,0}, \mathbf{r}_i^{\mathsf{sh}}$

8. $\texttt{BAD}_i := \emptyset$, $\mathcal{V} := \mathcal{Q}$
9. $((\mathsf{sh}_{i,1}, \ldots, \mathsf{sh}_{i,n}), \mathbf{A}_i) \leftarrow \texttt{VSShare}(\mathbf{H}, (s_{i,0}, 0, 0, w_{i,0}); \mathbf{r}_i^{\mathsf{sh}})$
   $s_{i,0}, w_{i,0} \leftarrow_{\$} \mathbb{Z}_q$, $\mathbf{r}_i^{\mathsf{sh}} \leftarrow_{\$} \mathbb{Z}_q^{4\times(t-1)}$
10. Query $(\texttt{ZK-prove}, \mathsf{ssid}, ((G, H_w), A_{i,0}), (s_{i,0}, w_{i,0}))$ to $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{pd}})$.
11. For all $P_j \in \mathcal{Q}$:
    (a) Retrieve (j, K_j) from $\mathcal{F}_{\mathsf{pki}}$.
    (b) If K_j defined: $K_{i,j} := k_i^{\mathsf{pki}} \cdot K_j$
    (c) Else: $K_{i,j} := \mathbf{0}$.
    (d) $\mathsf{ct}_{i,j} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid}, i, j), K_{i,j}) \oplus \mathsf{sh}_{i,j}$
12. Broadcast $(\mathsf{ssid}, i, \mathbf{A}_i, \{(j, \mathsf{ct}_{i,j})\}_{P_j \in \mathcal{Q}})$

On the following for $P_j \in \mathcal{Q}$**:** $\{\mathsf{sh}_{j,i}\}_{P_j \in \mathcal{Q}}, \{r_{i,j}^{\mathsf{dh}}\}_{(j,\cdots)\in \texttt{BAD}_i}$

– $(\mathsf{ssid}, j, \mathbf{A}_j = (A_{j,0}, \ldots), \{(j', \mathsf{ct}_{j,j'})\}_{P_{j'} \in \mathcal{Q}})$ from P_j, and
– $(\texttt{ZK-proven}, \mathsf{ssid}, P_j, ((G, H_w), A_{j,0}))$ from $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{pd}})$

13. Remove P_j from $\mathcal{V}$ if $(j', \mathsf{ct}_{j,j'})$ was not sent for some $P_{j'} \in \mathcal{Q}$.
14. $\mathsf{sh}_{j,i} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid}, j, i), K_{i,j}) \oplus \mathsf{ct}_{j,i}$
15. If $1 \neq \texttt{VSVerify}(\mathbf{H}, \mathbf{A}_j, \mathsf{sh}_{j,i}, i)$:
    (a) $\phi_{i,j}^{\mathsf{dh}} := (G, K_i, K_j, K_{i,j})$
    (b) $\pi_{i,j}^{\mathsf{dh}} \leftarrow \mathsf{Prv}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}\left(\phi_{i,j}^{\mathsf{dh}}, k_i^{\mathsf{pki}}; r_{i,j}^{\mathsf{dh}}\right)$; $r_{i,j}^{\mathsf{dh}} \leftarrow_{\$} \{0,1\}^{\lambda}$
    (c) Add $\left(j, K_{i,j}, \pi_{i,j}^{\mathsf{dh}}\right) \rightarrow \texttt{BAD}_i$.
16. If $\texttt{BAD}_i \neq \emptyset$ or $|\mathcal{V}| < t$:
    – Broadcast $(\mathsf{ssid}, i, \texttt{BAD}_i)$, and go to **Output Computation**.

On the above from all $P_j \in \mathcal{Q}$**:** r_i^{eq}, $r_{i,D}$

17. Query $F \leftarrow \mathtt{H}_{\mathbb{G}}(\mathsf{ssid}, \mathsf{trans})$ with trans the transcript of ssid thus far.
18. Define $\mathbf{F} := (G, F, H_v)$
19. $(s_i, u_i, v_i, w_i) \leftarrow \texttt{AggSh}(\{\mathsf{sh}_{j,i}\}_{P_j \in \mathcal{V}})$.
20. $C_i := (s_i, u_i, v_i, w_i) \cdot \mathbf{H}$
21. $B_i := (s_i, u_i, v_i) \cdot \mathbf{F}$
22. $\phi_i^{\mathsf{eq}} := (\mathbf{H}, \mathbf{F}, C_i, B_i)$
23. $\pi_i^{\mathsf{eq}} \leftarrow \mathsf{Prv}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}(\phi_i^{\mathsf{eq}}, (s_i, u_i, v_i, w_i); r_i^{\mathsf{eq}})$; $r_i^{\mathsf{eq}} \leftarrow_{\$} \{0,1\}^{\lambda}$

24⋆ $D_i := (s_i, r_{i,D}) \cdot \mathbf{G}$; $r_{i,D} \leftarrow_{\$} \mathbb{Z}_q$

25⋆ $\phi_i^{\mathsf{eqm}} := (\mathbf{F}, \mathbf{G}, B_i, D_i)$, $w_i := ((s_i, u_i, v_i), (s_i, r_{i,D}))$

26⋆ Query $(\texttt{ZK-prove}, \mathsf{ssid}, \phi_i^{\mathsf{eqm}}, w_i)$ to $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$.

27. Broadcast $\left(\mathsf{ssid}, i, B_i, \pi_i^{\mathsf{eq}}, \boxed{D_i}\right)$.

Output Computation

On the following for $P_j \in \mathcal{V}$:

– $(\mathsf{ssid}, j, B_j, \pi_j^{\mathsf{eq}}, \boxed{D_j})$ or $(\mathsf{ssid}, j, \mathtt{BAD}_j)$ from P_j,

⋆ $(\texttt{ZK-proven}, \mathsf{ssid}, P_j, (\mathbf{F}, \mathbf{G}, B_j, D_j))$ from $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$

28. $C_j = \sum_{j \in \mathcal{I}_{\mathcal{Q}}} \sum_{k=0}^{t} (j^k) A_{j,k}$
29. Remove j from $\mathcal{V}$ if $1 \neq \mathsf{Ver}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}(((\mathbf{H}, \mathbf{F}), C_j, B_j), \pi_j^{\mathsf{eq}})$.
30. For all $\left(j', K_{j,j'}, r_{j,j'}^{\mathsf{dh}}\right) \in \mathtt{BAD}_j$ remove $P_{j'}$ from $\mathcal{V}$ if:
    – $1 = \mathsf{Ver}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}\left(\phi_{j,j'}^{\mathsf{dh}}, \pi_{j,j'}^{\mathsf{dh}}\right)$ for $\phi_{j,j'}^{\mathsf{dh}} = (G, K_j, K_{j'}, K_{j,j'})$, and
    – $1 \neq \texttt{VSVerify}(\mathbf{H}, \mathbf{A}_{j'}, \mathtt{H}_{\mathtt{sh}}((\mathsf{sid}, \mathsf{ssid}, j', j), K_{j,j'}) \oplus \mathsf{ct}_{j',j}, j)$
31. On messages from all $P_j \in \mathcal{V}$: continue.
32. If any complaint verified in Step 30:
    Output $(\texttt{CDKG-iabort}, \mathsf{ssid}, j)$ for $P_j \in \mathcal{Q} \setminus \mathcal{V}$ with smallest pid j.
33. If t messages $(\mathsf{ssid}, j, B_j, \pi_j^{\mathsf{eq}}, \boxed{D_j})$ verified in Step 29:
    (a) $B(X) := \texttt{Interpolate}(\{(j, B_j)\}_{j \in \mathcal{V}})$
    (b) $\mathsf{pk} := B(0)$, $\mathsf{sk}_i := s_i$
    (c) **Output** $\left(\mathsf{pk}, \mathsf{sk}_i, \boxed{r_{i,D}, \{D_j\}_{P_j \in \mathcal{V}}}\right)$.
34. Else:
    Output $(\texttt{CDKG-iabort}, \mathsf{ssid}, j)$ for $P_j \in \mathcal{Q} \setminus \mathcal{V}$ with smallest j.

Theorem 3 (Security of $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}$). *For all $\mathcal{G}$ and all $t < n$ protocol $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}(\mathcal{G}, t)$ UC-realizes functionality $\mathcal{F}_{\mathsf{cdkg}}(\mathcal{G}, t)$ in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model in the presence of an adaptive malicious adversary under the DDH assumption in $\mathcal{G}$, given Π_{eq} and Π_{dh} are ZK+S and AZK+SS NIZKs (Definitions 7, 8, 9, 10), respectively. Moreover, the protocol achieves identifiable abort against a dishonest majority under the condition that $\mathcal{F}_{\mathsf{bc}}$ guarantees delivery of all messages.*

Proof. The simulator appears in Figure 16, which is followed by a hybrid argument of indistinguishability starting at the real world and ending at the ideal world.

Figure 16: $\mathcal{S}_{\mathsf{cdkg}}$.
Committed DKG Simulator for $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}(\mathcal{G}, t)$, Theorem 3.

The simulator forwards all communication between $\mathcal{A}$ and $\mathcal{Z}$ and, upon invocation, receives the set of corrupted parties $\mathcal{C} \subset \mathcal{P}$ and maintains the set $\mathcal{H} := \mathcal{P} \setminus \mathcal{C}$.

The simulator makes black-box use of the following for the proof systems:

– Π_{eq} for $\mathcal{R}_{\mathsf{eq}}$: $(\mathcal{S}_{\mathsf{P}}^{\mathsf{eq}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}})$ as per the AZK property in Definition 9.

The simulator also makes use of the resharing subroutine $\mathcal{E}^{(\cdot)}_{\mathsf{vss}}$ from Definition 11 during adaptive corruption. Essentially, resharing operates over all of the honest parties VSS inputs simultaneously and equivocates the VSS commitments while keeping the corrupt party shares and the commitments fixed.

On Invocation. Initialize the following tables for record keeping:

- $\mathtt{PKI}[i] = k_i^{\mathsf{pki}}$: records the PKI key of each honest $P_i \in \mathcal{P} \setminus \mathcal{C}$.
- $\mathtt{H}_{\mathsf{sh}}[]$, $\mathtt{H}_{\mathbb{G}}[]$, $\mathtt{H}_{\mathsf{eq}}[]$: input/output for the corresponding instances of $\mathcal{F}_{\mathsf{ro}}$.
- $\mathtt{AZK}[i][\mathcal{R}][\mathsf{ssid}] := \phi_i$: records proven statements of every $P_i \in \mathcal{Q}$ for $\mathcal{F}_{\mathsf{azk}}(\mathcal{R})$, and records proof-related information for $\mathcal{R}_{\mathsf{dh}}$, $\mathcal{R}_{\mathsf{eq}}$.
- $\mathtt{func\text{-}key}[\mathsf{ssid}][i]$: at ssid when $i = \mathcal{S}$ records the public key $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ output by $\mathcal{F}_{\mathsf{cdkg}}$, and when $i \neq \mathcal{S}$ records the commitment D_i for P_i in ssid.
- $\mathtt{prot\text{-}key}[i][\mathsf{ssid}] = (s_i, u_i, v_i, w_i)$: records the shares of P_i that open B_i, C_i in subsession ssid, when $i = \mathcal{S}$ records the aggregate shared values in the simulated protocol.
- $\mathtt{rigged\text{-}gen}[]$: at q records (α, s, F) such that $\mathtt{H}_{\mathbb{G}}[q] = F$, and $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} = sG + \alpha F$.

Hybrid Functionalities. Emulate functionalities as below:

$\mathcal{F}_{\mathsf{azk}}$. Emulate each instance as below:

1. $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{pd}})$:
   - On $(\mathtt{ZK\text{-}prove}, \mathsf{ssid}, ((G, H_w), A_{j,0}), (s_{j,0}, w_{j,0}))$ from $P_j \in \mathcal{Q} \cap \mathcal{C}$:
     - If $(((G, H_w), A_{j,0}), (s_{j,0}, w_{j,0})) \in \mathcal{R}_{\mathsf{pd}}$:
       - Record $\mathtt{AZK}[j][\mathcal{R}_{\mathsf{pd}}][\mathsf{ssid}] := (((G, H_w), A_{j,0}), (s_{j,0}, w_{j,0}))$.
       - Go to Step 21.
     - Else: ignore the query.
2. $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$:
   - On $(\mathtt{ZK\text{-}prove}, \mathsf{ssid}, P_j, (\mathbf{F}, \mathbf{G}, B_j, D_j), (\mathsf{sk}_j, (u_j, v_j), r_j))$ from $P_j \in \mathcal{Q} \cap \mathcal{C}$:
     - If $((\mathbf{F}, \mathbf{G}, B_j, D_j), (\mathsf{sk}_j, (u_j, v_j), r_j)) \in \mathcal{R}_{\mathsf{eqm}}$:
       - Record $\mathtt{AZK}[j][\mathcal{R}_{\mathsf{eqm}}][\mathsf{ssid}] := ((\mathbf{F}, \mathbf{G}, B_j, D_j), (\mathsf{sk}_j, (u_j, v_j), r_j))$.
       - Go to Step 31.
     - Else: ignore the query.

$\mathcal{F}_{\mathsf{bc}}$. Emulate as detailed in Section 2.1.

$\mathcal{F}_{\mathsf{pki}}$. Emulate as detailed in Section 2.1.
Record all secret keys in $\mathtt{PKI}[]$, as in Step 7 below for the honest parties.

$\mathcal{F}_{\mathsf{ro}}$. Emulate each instance as below:

3. $\mathtt{H}_{\mathsf{eq}}$: emulate as detailed in Section 2.1.
4. $\mathtt{H}_{\mathbb{G}}$ on input q:
   (a) If $\mathtt{H}_{\mathbb{G}}[q]$ is undefined, $(\mathtt{CDKG\text{-}CK}, H)$ not recorded, and $q = (\mathsf{id}, (1, r_1), \ldots, (n, r_n))$, for (i, r_i) the message $P_i \in \mathcal{P} \setminus \mathcal{C}$ from Step 6:
      i. If $\mathsf{id} = \mathsf{com}$:
         - Query $(\mathtt{CDKG\text{-}newCK}, q)$ to $\mathcal{F}_{\mathsf{cdkg}}$.
         - Receive $(\mathtt{CDKG\text{-}newCK}, H_q)$.
      ii. Else if $\mathsf{id} \in \{u, v, w\}$:

- Sample $\tau_{\mathsf{id}} \leftarrow_{\$} \mathbb{Z}_q$
- Define $H_q := \tau_q \cdot G$
- Record $\texttt{rigged-gen}[q] := (\tau_q, H_q)$

iii. Record $\texttt{H}_{\mathbb{G}}[q] := H_q$.

(b) If $\texttt{H}_{\mathbb{G}}[q]$ is undefined, $q = (\mathsf{ssid}, \cdots)$ for defined $\texttt{func-key}[\mathsf{ssid}][\mathcal{S}] = \mathsf{pk}_{\mathcal{F}_{\text{dkg}}}$, and $q = (\mathsf{ssid}, \cdots)$ is consistent with the transcript of ssid thus far.

i. Sample $\alpha, s \leftarrow_{\$} \mathbb{Z}_q$.

ii. Compute $F := \alpha^{-1}(\mathsf{pk}_{\mathcal{F}_{\text{dkg}}} - sG)$.

iii. Define $\texttt{H}_{\mathbb{G}}[q] := F$.

iv. Record $\texttt{rigged-gen}[q] := (\alpha, s, F)$.

(c) Else if $\texttt{H}_{\mathbb{G}}[q]$ is undefined:

- Define $\texttt{H}_{\mathbb{G}}[q] := F$ for $F \leftarrow_{\$} \mathbb{G}$.

(d) Output $\texttt{H}_{\mathbb{G}}[q]$ to P_i.

5. $\texttt{H}_{\texttt{sh}}$ on input q:

(a) If $\texttt{H}_{\texttt{sh}}[q]$ is undefined:

i. If $q = (\mathsf{ssid}, i, j, K_{i,j})$ for $P_i, P_j \in \mathcal{P} \setminus \mathcal{C}$ and $K_{i,j} = \texttt{PKI}[i] \cdot \texttt{PKI}[j] \cdot G$: Abort simulation.

ii. Else : Sample $\texttt{H}_{\texttt{sh}}[q] \leftarrow_{\$} \{0,1\}^{4\log(q-1)}$.

(b) Output $\texttt{H}_{\texttt{sh}}[q]$ to P_i.

Initialization.

On $(\texttt{CDKG-init}, i, \mathcal{P})$ **from** $\mathcal{F}_{\mathsf{cdkg}}$ **with** $P_i \in \mathcal{P} \setminus \mathcal{C}$**:**

6. Simulate $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Step 1- 3 for P_i.
7. Record simulated $\mathsf{PKI}[i] := k_i^{\mathsf{pki}}$.

On $\{(j, r_j)\}_{P_j \in \mathcal{C}}$ **from** $\mathcal{A}$ **to** $P_i \in \mathcal{P}$ **via** $\mathcal{F}_{\mathsf{bc}}$**:**

8. Define $R := ((1, r_1), \ldots, (n, r_n))$.
9. For $\mathsf{id} \in \{\mathsf{com}, u, v, w\}$: Retrieve (or define, as in Step 4) $H_{\mathsf{id}} := \texttt{H}_{\mathbb{G}}[(\mathsf{id}, R)]$.
10. Record $(\texttt{CDKG-CK}, \texttt{H}_{\mathbb{G}}[(\mathsf{com}, R)])$.
11. Define $\mathbf{G} := (G, \texttt{H}_{\mathbb{G}}[(\mathsf{com}, R)])$
12. Record $(\texttt{CDKG-gens}, \texttt{rigged-gen}[(u, R)], \texttt{rigged-gen}[(v, R)], \texttt{rigged-gen}[(w, R)])$.
13. Define $\mathbf{H} := (G, H_u, H_v, H_w)$.
14. Query $(\texttt{CDKG-setCK}, R)$ to $\mathcal{F}_{\mathsf{cdkg}}$.

Key Generation.

On $(\texttt{CDKG-gen}, \mathsf{ssid}, i, \mathcal{Q})$ **from** $\mathcal{F}_{\mathsf{cdkg}}$ **with** $P_i \in \mathcal{Q} \setminus \mathcal{C}$**:**

15. Simulate P_i as in $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Step 9.
16. Record $\texttt{vss-inputs}[i][\mathsf{ssid}] := ((s_{i,0}, 0, 0, w_{i,0}), \mathbf{r}_i^{\mathsf{sh}})$.
17. Instead of $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Step 10, emulate $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{pd}})$:
    - Send $(\texttt{Prover}, \mathsf{ssid}, P_i, ((G, H_w), A_{i,0}))$ to $\mathcal{A}^*$.
    - Receive $(\texttt{ZK-deliver}, \mathsf{ssid}, P_i)$ from $\mathcal{A}^*$.

– Record $\texttt{AZK}[i][\mathcal{R}_{\mathsf{pd}}][\mathsf{ssid}] := ((G, H_w), A_{i,0})$.

18. Instead of $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Step 11:
    (a) For all $P_j \in \mathcal{Q} \cap \mathcal{C}$: encrypt shares as in $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$.
    (b) For all $P_j \in \mathcal{Q} \setminus \mathcal{C}$: sample $\mathsf{ct}_{i,j} \leftarrow_{\$} \{0,1\}^{4\log(q-1)}$.
19. Broadcast $(\mathsf{ssid}, i, \mathbf{A}_i, \{(j, \mathsf{ct}_{i,j})\}_{P_j \in \mathcal{Q}})$ to $\mathcal{Q}$.
20. Once received for all $P_j \in \mathcal{Q} \setminus \mathcal{C}$:
    – Receive $(\texttt{CDKG-pk}, \mathsf{ssid}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}, (D_i)_{P_i \in \mathcal{Q} \setminus \mathcal{C}})$ from $\mathcal{F}_{\mathsf{cdkg}}$
    – Record $\texttt{func-key}[\mathsf{ssid}][\mathcal{S}] := \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$.
    – For all $P_i \in \mathcal{Q} \setminus \mathcal{C}$: record $\texttt{func-key}[\mathsf{ssid}][i] := D_i$.

On $(\mathsf{ssid}, j, \mathbf{A}_j, \{(j', \mathsf{ct}_{j,j'})\}_{P_{j'} \in \mathcal{Q}})$ **from** $\mathcal{A}$ **for** $P_j \in \mathcal{Q} \cap \mathcal{C}$**:**

21. If $\texttt{AZK}[j][\mathcal{R}_{\mathsf{pd}}][\mathsf{ssid}] = (((G, H_w), A_{j,0}), (s_{j,0}, w_{j,0}))$ and the message was received: continue as below.
22. For all $P_i \in \mathcal{Q} \setminus \mathcal{C}$:
    – Simulate verification as in $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Steps 13-15.
    – For any $(\mathsf{ssid}, j, \phi^{\mathsf{dh}}_{i,j}, \pi^{\mathsf{dh}}_{i,j}) \in \texttt{BAD}_i$:
      • Add $\texttt{AZK}[i][\mathcal{R}_{\mathsf{dh}}][\mathsf{ssid}] \leftarrow r^{\mathsf{dh}}_{i,j}$.
23. If $\mathbf{A}_j = (A_{j,0}, [\cdots])$: record $\texttt{vss-inputs}[j][\mathsf{ssid}] := (s_{j,0}, 0, 0, w_{j,0})$.
24. Else: Record $\texttt{vss-inputs}[j][\mathsf{ssid}] := (0, 0, 0, 0)$.

Once $\texttt{vss-inputs}[j][\mathsf{ssid}]$ **defined for all** $P_j \in \mathcal{Q} \cap \mathcal{C}$**, and** $\mathcal{A}$ **releases messages of all** $P_i \in \mathcal{Q} \setminus \mathcal{C}$**:**

25. Define trans to be the transcript of ssid thus far.
26. Retrieve (or define, as in Step 9(b)i) $F \leftarrow \mathtt{H}_{\mathbb{G}}[\mathsf{ssid}, \mathsf{trans}]$, and $(\alpha, s, F) \leftarrow \texttt{rigged-gen}[\mathsf{ssid}, \mathsf{trans}]$.
27. Record $(\texttt{CDKG-RG}, \mathsf{ssid}, \alpha, s, F)$.
28. Define (for ssid) as $\mathbf{F} := (G, F, H_v)$.
29. Recompute a random honest party's secrets:
    (a) Sample $P_T \leftarrow_{\$} \mathcal{Q} \setminus \mathcal{C}$.
    (b) Compute $(s_\Pi, [\cdots]) := \sum_{P_j \in \mathcal{Q}} \texttt{vss-inputs}[j][\mathsf{ssid}]$.
    (c) Retrieve $(\texttt{CDKG-gens}, \tau_U, H_u, \tau_V, H_v, \tau_W, H_w)$.
    (d) Reshare $\mathcal{E}^{(s,w)}_{\mathtt{vss}}(\mathsf{ssid}, T, \tau_W^{-1}, \{(0, s - s_\Pi)\} \cup \{(j, 0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$
    (e) Reshare $\mathcal{E}^{(u,w)}_{\mathtt{vss}}(\mathsf{ssid}, T, (\tau_W^{-1}\tau_U), \{(0, \alpha)\} \cup \{(j, 0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$.
    (f) Record $\texttt{curr-target}[\mathsf{ssid}] := T$.
30. For all $P_i \in \mathcal{Q} \setminus \mathcal{C}$ such that $\texttt{BAD}_i = \emptyset$:
    (a) For $P_j \in \mathcal{Q} \setminus \mathcal{C}$: $((\cdots, \mathsf{sh}_{j,i}, \cdots), \cdot) \leftarrow \texttt{VSShare}(\mathbf{H}, \texttt{vss-inputs}[j][\mathsf{ssid}])$.
    (b) Define $(s_i, u_i, v_i, w_i) := \sum_{P_j \in \mathcal{Q}} \mathsf{sh}_{j,i}$.
    (c) Simulate Step 21, as in $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$.
    (d) Instead of Step 23 of $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$:
      – Simulate $(\mathtt{H}_{\mathsf{eq}}, \pi^{\mathsf{eq}}_i) \leftarrow \mathcal{S}^{\mathsf{eq}}_{\mathsf{P}}(\mathtt{H}_{\mathsf{eq}}, (\mathbf{H}, \mathbf{F}, C_i, B_i))$.
      – Record $\texttt{AZK}[i][\mathcal{R}_{\mathsf{eq}}][\mathsf{ssid}] := ((\mathbf{H}, \mathbf{F}, C_i, B_i), \pi^{\mathsf{eq}}_i)$.
    (e) Instead of Step 26 of $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$, emulate $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$:

– Send $(\texttt{Prover}, \mathsf{ssid}, P_i, (\mathbf{F}, \mathbf{G}, B_i, D_i))$ to $\mathcal{A}^*$.
– Receive $(\texttt{ZK-deliver}, \mathsf{ssid}, P_i)$ from $\mathcal{A}^*$.
– Record $\texttt{AZK}[i][\mathcal{R}_{\mathsf{eqm}}][\mathsf{ssid}] := (\mathbf{F}, \mathbf{G}, B_i, D_i)$.

(f) Step 27: Broadcast $(\mathsf{ssid}, i, B_i, \pi_i^{\mathsf{eq}}, D_i)$ from P_i via $\mathcal{A}$.

On $(\mathsf{ssid}, j, B'_j, \pi_j^{\mathsf{eq}}, D_j)$ **from** $\mathcal{A}$ **for** $P_j \in \mathcal{Q} \cap \mathcal{C}$**:**

31. If $\texttt{AZK}[j][\mathcal{R}_{\mathsf{eqm}}][\mathsf{ssid}] = ((\mathbf{F}, \mathbf{G}, B_j, D_j), (\mathsf{sk}_j, (u_j, v_j), r_j)) \in \mathcal{R}_{\mathsf{eqm}}$ and the message was received: continue as below.
32. If π_j^{eq} verifies and $B_j = B'_j$:
    – Record $\texttt{prot-key}[j][\mathsf{ssid}] := (\mathsf{sk}_j, u_j, v_j, \perp)$.
    – Record $\texttt{rands}[j][\mathsf{ssid}] := r_j$.
33. Else: set $(\mathsf{sk}_j, r_j) := \perp$.
34. For all $P_i \in \mathcal{Q} \setminus \mathcal{C}$:
    – Simulate receipt and verification by as in $\Pi^{\mathsf{ia}}_{\mathsf{cdkg}}$ Steps 29-30.

On a Round 2 Message from all $P_j \in \mathcal{Q} \cap \mathcal{C}$**:**

35. If there was a verifying complaint($\texttt{BAD}_k \neq \emptyset$ for some $P_k \in \mathcal{Q}$):
    – Query $(\texttt{CDKG-iabort}, \mathsf{ssid}, j)$ to $\mathcal{F}_{\mathsf{cdkg}}$, for the smallest pid of $P_j \in \mathcal{Q} \setminus \mathcal{V}$.
36. Else:
    – Query $(\texttt{CDKG-shares}, \mathsf{ssid}, \{(j, \mathsf{sk}_j, r_j)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$ to $\mathcal{F}_{\mathsf{cdkg}}$.
    – Query $(\texttt{CDKG-deliver}, \mathsf{ssid})$ to $\mathcal{F}_{\mathsf{cdkg}}$.

Adaptive Corruptions.

– $\texttt{PKI}[i]$: the set up randomness was recorded in Step 7, k_i^{pki}
– $\texttt{vss-inputs}[i][\mathsf{ssid}] = (s_{i,0}, u_{i,0}, v_{i,0}, w_{i,0}, \mathbf{r}_i^{\mathsf{sh}})$: recorded in Step 16, unless $i = \texttt{curr-target}[\mathsf{ssid}]$ in which case computed as in Step 40(a)ii below, $s_{i,0}, w_{i,0}; \mathbf{r}_i^{\mathsf{sh}}$
– $\texttt{AZK}[\mathsf{ssid}][i][\mathcal{R}_{\mathsf{dh}}]$: was recorded in Step 22, $\{r_{i,j}^{\mathsf{dh}}\}$ for all $P_j \in \texttt{BAD}_i$
– $\texttt{AZK}[\mathsf{ssid}][i][\mathcal{R}_{\mathsf{eq}}] = r_i^{\mathsf{eq}}$: simulated in Step 40d below, r_i^{eq}
– $\texttt{rands}[\mathsf{ssid}]$: commitment randomness is returned by $\mathcal{F}_{\mathsf{cdkg}}$ in Step 39, $r_{i,D}$
– $\texttt{prot-key}[i][\mathsf{ssid}]$: opening of B_i, C_i in ssid computed in Step 40(d)i,
– $\{(\mathsf{ssid}, \mathsf{sh}_{j,i})\}_{P_j \in \mathcal{Q} \setminus \mathcal{C}}$: computable from k_i^{pki} and $\mathtt{H}_{\mathsf{sh}}$ programmed in Step 40(e)ii below.

On $(\texttt{CDKG-corrupt}, i)$ **from** $\mathcal{A}$:

37. Add $P_i \to \mathcal{C}$.
38. Query $(\texttt{CDKG-corrupt}, i)$ to $\mathcal{F}_{\mathsf{cdkg}}$.
39. Receive $(\texttt{CDKG-corrupt}, i, \texttt{shares}, \texttt{rands})$ from $\mathcal{F}_{\mathsf{cdkg}}$.
40. For each ssid such that $\texttt{shares}[\mathsf{ssid}]$ is non-empty:
    (a) If $i = \texttt{curr-target}[\mathsf{ssid}]$ and $|\mathcal{Q} \cap \mathcal{C}| < t$:
        i. Sample $P_{\tilde{T}} \leftarrow_{\$} \mathcal{Q} \setminus \mathcal{C}$.
        ii. Reshare $\mathcal{E}_{\mathsf{vss}}^{(u,w)}(\mathsf{ssid}, T, \tau_W^{-1}\tau_U, \{(0, -\alpha)\} \cup \{(j, 0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$. *// detarget P_i*
        iii. Reshare $\mathcal{E}_{\mathsf{vss}}^{(u,w)}(\mathsf{ssid}, \tilde{T}, \tau_W^{-1}\tau_U, \{(0, \alpha)\} \cup \{(j, 0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$. *// target $P_{\tilde{T}}$*

iv. Overwrite $\texttt{curr-target}[\mathsf{ssid}] := \tilde{T}$.

(b) Update for $\mathsf{sk}_i := \texttt{shares}[\mathsf{ssid}]$: *// match shares output by* $\mathcal{F}_{\mathsf{cdkg}}$

i. Compute $(s_i, \cdots) := \sum_{P_j \in \mathcal{Q}} \mathsf{sh}_{j,i}$
for $(\cdot, (\cdots, \mathsf{sh}_{j,i}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, \texttt{vss-inputs}[j][\mathsf{ssid}])$.

ii. Reshare $\mathcal{E}_{\mathtt{vss}}^{(s,v)}(\mathsf{ssid}, i, \tau_V^{-1}, \{(i, \mathsf{sk}_i - s_i)\} \cup \{(0,0)\} \cup \{(j,0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$.

(c) If $|\mathcal{Q} \cap \mathcal{C}| = t$: *// match* sk *output by* $\mathcal{F}_{\mathsf{cdkg}}$

i. Retrieve $(\mathsf{ssid}, \alpha, s, F)$.

ii. Retrieve $(\mathsf{sk}_j, \cdots) \leftarrow \texttt{prot-key}[j][\mathsf{ssid}]$ for all $P_j \in \mathcal{Q} \cap \mathcal{C}$.

iii. Interpolate $S_{\mathcal{F}_{\mathtt{DKG}}}(X) \leftarrow \mathtt{Interpolate}(\{(j, \mathsf{sk}_j)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$.

iv. Define $\mathsf{sk}_{\mathcal{F}_{\mathtt{dkg}}} := S_{\mathcal{F}_{\mathtt{DKG}}}(0)$.

v. Compute $\tau_F := \alpha^{-1}(\mathsf{sk}_{\mathcal{F}_{\mathtt{dkg}}} - s)$.

vi. Retrieve $T \leftarrow \texttt{curr-target}[\mathsf{ssid}]$.

vii. Reshare $\mathcal{E}_{\mathtt{vss}}^{(s,u,w)}(\mathsf{ssid}, T, \left(\tau_F, \tau_W^{-1}(\tau_F - \tau_U)\right), \{0, \alpha\} \cup \{(j,0)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$.

(d) Simulate NIZK randomness: *// via* $(\mathtt{H}_{\mathsf{eq}}, \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}})$

i. Define $\texttt{prot-key}[i][\mathsf{ssid}] := (s_i, u_i, v_i, w_i) := \sum_{P_j \in \mathcal{V}} \mathsf{sh}_{j,i}$
for $(\cdot, (\cdots, \mathsf{sh}_{j,i}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, \texttt{vss-inputs}[j][\mathsf{ssid}])$.

ii. $(\mathtt{H}_{\mathsf{eq}}, r_i^{\mathsf{eq}}) \leftarrow \mathcal{S}_{\mathsf{R}}^{\mathsf{eq}}(\mathtt{H}_{\mathsf{eq}}, \texttt{AZK}[\mathsf{ssid}][i][\mathcal{R}_{\mathsf{eq}}], \texttt{prot-key}[i][\mathsf{ssid}])$.

iii. Overwrite $\texttt{AZK}[i][\mathcal{R}_{\mathsf{eq}}][\mathsf{ssid}] := r_i^{\mathsf{eq}}$.

(e) Program $\mathtt{H}_{\mathtt{sh}}$: *// match all shares exchanged with honest parties*

i. Recompute $(\cdot, (\mathsf{sh}_{i,1}, \ldots, \mathsf{sh}_{i,n})) \leftarrow \mathtt{VSShare}(\mathbf{H}, \texttt{vss-inputs}[i][\mathsf{ssid}])$.

ii. For all $P_j \in \mathcal{Q} \setminus \mathcal{C}$:

– Retrieve ciphertext $\mathsf{ct}_{i,j}$ sent to party P_j in subsession ssid.

– Define $\mathtt{H}_{\mathbb{G}}[\mathsf{ssid}, i, j, K_{i,j}] := \mathsf{ct}_{i,j} \oplus \mathsf{sh}_{i,j}$.

– Recompute $(\cdot, (\cdots, \mathsf{sh}_{j,i}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, \texttt{vss-inputs}[j][\mathsf{ssid}])$.

– Retrieve ciphertext $\mathsf{ct}_{j,i}$ received from party P_j in subsession ssid.

– Define $\mathtt{H}_{\mathbb{G}}[\mathsf{ssid}, j, i, K_{i,j}] := \mathsf{ct}_{j,i} \oplus \mathsf{sh}_{j,i}$.

41. **Output:** $(i, \texttt{PKI}[i], \texttt{vss-inputs}[i], \texttt{AZK}[i][\mathcal{R}_{\mathsf{dh}}], \texttt{AZK}[i][\mathcal{R}_{\mathsf{eq}}], \texttt{rands})$ **to** $\mathcal{A}$.

Hybrid H_0: *Real World.* This corresponds to the real execution of $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}$, where the environment $\mathcal{Z}$ runs the protocol $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}$ with the n parties in $\mathcal{P}$ and the adversary $\mathcal{A}$, all with access to ideal functionalities $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$. For the random variable $\mathsf{EXEC}_{\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}, \mathcal{A}, \mathcal{Z}}(z)$ the output on an execution of $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}$ with the $\mathcal{Z}$ and $\mathcal{A}$ on input z in this hybrid correspond to the ensemble:

$$\mathsf{H}_0 := \{\mathsf{EXEC}_{\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}, \mathcal{A}, \mathcal{Z}}(z)\}_{z \in \{0,1\}^*}.$$

Hybrid H_1: *Ideal Layout.* The existing ITIs are grouped into a single machine called the simulator $\mathcal{S}$ that forwards communication between $\mathcal{A}$ and $\mathcal{Z}$ unchanged. Between $\mathcal{Z}$ and $\mathcal{S}$ new dummy party nodes are added, and between these nodes and $\mathcal{S}$ a new functionality node $\mathcal{F}$ is added. For now, $\mathcal{F}$ forwards input and output between $\mathcal{S}$ and the dummy parties, unchanged. This change is syntactic, so the view of $\mathcal{Z}$ is unchanged; H_1 is indistinguishable from H_0.

Hybrid H_2: *Functionality Record Keeping & Queries on Behalf of Corrupted Parties.* In this hybrid, $\mathcal{S}$ generates queries to $\mathcal{F}$ from the messages received from $\mathcal{A}$. Both the functionality and simulator create all relevant records. The changes to $\mathcal{F}$ and $\mathcal{S}$ are listed below.

– *Modifications to $\mathcal{F}$:* Record keeping is done as in Figure 14. As of H_2, the functionality still forwards inputs and outputs between $\mathcal{S}$ and the dummy parties. As of H_2 the `CDKG-shares` and `CDKG-iabort` interfaces of $\mathcal{F}$ are fully implemented.

– *Modifications to $\mathcal{S}$:*

- *Record Keeping.* All record keeping tables are initialized and maintained.
- *Corrupt Party Shares Queries.* As in Step 36, queries $(\texttt{CDKG-shares}, \mathsf{ssid}, \{(j, \mathsf{sk}_j, r_j)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$ to $\mathcal{F}_{\mathsf{cdkg}}$ with the shares extracted from $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$ as in Step 31.
- *Identifiable Abort Queries.* As in Step 35, after receiving (potentially empty) Round 2 messages in subsession ssid from every party in $\mathcal{Q}$ the simulator submits a query $(\texttt{CDKG-iabort}, \mathsf{ssid}, \cdot)$ if at least one verifying complaint was sent in Round 2.
- *Deliver Key Queries.* As in Step 36, after receiving (potentially empty) Round 2 messages from every party in subsession ssid the simulator submits a deliver key query of $(\texttt{CDKG-deliver}, \mathsf{ssid}, \cdots)$ to $\mathcal{F}$ that in H_2 includes the values generated in the simulated protocol.

This change is syntactic, so the view of $\mathcal{Z}$ is unchanged; H_2 is indistinguishable from H_1.

Hybrid H_3: *Simulated NIZKs.* In this hybrid, the simulator makes black-box use of the NIZK and NIZKPoK simulators from Definition 7 as described below. As in Step 30d the simulator simulates the proofs π_i^{eq} with $\mathcal{S}_{\mathsf{P}}^{\mathsf{eq}}$. As in Step 40d on the adaptive corruption of party P_i the simulator uses $\mathcal{S}_{\mathsf{R}}^{\mathsf{eq}}$ to generate randomness r_i^{eq} that can be used to recompute the simulated proof. Under the assumption that Π_{eq} satisfies zero knowledge against adaptive corruption (Definition 9) this change is indistinguishable to $\mathcal{Z}$; H_3 is indistinguishable from H_2 and $\mathsf{H}_3 \approx \mathsf{H}_2$.

Hybrid H_4: *Trapdoored Group Generators, Functionality Samples Commitment Key.* In this hybrid, $\mathcal{F}$ samples the commitment key H via the $(\texttt{CDKG-newCK}, \texttt{CDKG-setCK})$ interfaces of $\mathcal{F}_{\mathsf{cdkg}}$. As in Step 4, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{ro}}$ to output H sampled via the `CDKG-newCK` query of $\mathcal{F}$. The simulator trapdoors the group generators (H_u, H_v, H_w) output by $\mathcal{F}_{\mathsf{ro}}$ as in Step 4; the elements $H_u := \tau_u G, H_v := \tau_v G, H_w := \tau_w G$ are generated with known (and recorded) τ_u, τ_v, τ_w. The distribution of H, H_u, H_v, H_w in both H_3 and H_4 is uniform in $\mathbb{G}$. As of H_4 the interfaces `CDKG-newCK` and `CDKG-setCK` of $\mathcal{F}$ are fully implemented.

Hybrid H_5: *Random Ciphertexts between Honest Parties.* In this hybrid, the simulator samples the ciphertexts sent between honest parties uniformly and programs $\mathtt{H}_{\mathtt{sh}}$ on an adaptive corruption instruction such that these ciphertexts decrypt to the correct shares.

– *Modifications to $\mathcal{S}$:*

- As in Step 18, the simulator samples the ciphertexts sent from honest P_i to honest $P_j \in \mathcal{H}$ at random $\mathsf{ct}_{i,j} \leftarrow_{\$} \{0,1\}^{4\log(q-1)}$.
- As in Step 40e, the simulator programs the random oracle $\mathtt{H}_{\mathtt{sh}}$ on an adaptive corruption of party P_i to output $\mathsf{ct}_{i,j} \oplus \mathsf{sh}_{i,j}$ for every honest P_j.
- As in Step 8(a)i, the simulator aborts the simulation entirely if the environment submits a query to $\mathtt{H}_{\mathtt{sh}}$ of the form $(\mathsf{ssid}, i, j, K_{i,j})$, where $K_{i,j}$ is the shared Diffie-Hellman key for honest parties P_i, P_j. As of H_5, the emulation of $\mathtt{H}_{\mathtt{sh}}$ is fully implemented.

The distribution of the ciphertexts is uniform in both H_5 and H_4 as the output of $\mathtt{H}_{\mathtt{sh}}$ is sampled randomly. Therefore, this change is perfectly indistinguishable unless $\mathcal{Z}$ can trigger the simulation to abort by querying $(\mathsf{ssid}, i, j, K_{i,j})$ for an honest pair $P_i, P_j \in \mathcal{H}$. The shared key $K_{i,j}$ is a Diffie-Hellman key, where the transcript of the key exchange are the keys K_i, K_j stored and output by $\mathcal{F}_{\mathsf{pki}}$. An environment that can distinguish

H_5 from H_4 implies an adversary against the confidentiality of the Diffie-Hellman Key Exchange protocol, exactly as in H_5 for Theorem 2 in Section 4.

Hybrid H_6: *Functionality Samples Keys.* In this hybrid, the functionality $\mathcal{F}$ samples and sends key pairs $(\mathsf{sk}, \mathsf{pk})$ to the simulator, and the simulator ensures that the protocol outputs pk received from $\mathcal{F}$.

– *Modifications to $\mathcal{F}$:* As in the `CDKG-gen` interface of $\mathcal{F}_{\mathsf{cdkg}}$, upon receipt of a query from all $P_j \in \mathcal{Q}$ in subsession ssid the functionality $\mathcal{F}$ functionality samples the key pair $(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}})$. The interface is not yet fully implemented; in H_6, $\mathcal{F}$ outputs $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ to $\mathcal{S}$.

– *Modifications to $\mathcal{S}$:* The simulator receives $(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}})$ from $\mathcal{F}$ and executes Step 29 with $\alpha = 0$ and $s = \mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ to ensure that the output of the simulated protocol matches $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$.

The distribution of the key pair in H_5 and H_6 is uniform. The executed steps do not change the shares of the corrupted parties, but the aggregate shared secret matches $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ received from $\mathcal{F}$. The simulator effectively equivocates the Pedersen commitments to the coefficients of the sharing polynomials. Thus, the view of $\mathcal{Z}$ is unchanged. We refer to Theorem 2 for more details.

Hybrid H_7: *Functionality Samples Honest Party Key Shares.* In this hybrid, $\mathcal{S}$ ensures that the secret key share output to $\mathcal{A}$ on an adaptive corruption is the key share output to the dummy parties by $\mathcal{F}$; as in Step 40b of Figure 16, the simulator $\mathcal{S}$ calls $\mathcal{E}_{\mathsf{vss}}^{(s,v)}$ to ensure that the secret shares returned to $\mathcal{A}$ are $(\mathsf{sk}_i, u_i, v_i)$ such that sk_i was received from $\mathcal{F}$.

– *Modifications to $\mathcal{F}$:*

- As in the `CDKG-shares` interface of $\mathcal{F}_{\mathsf{cdkg}}$, the functionality $\mathcal{F}$ interpolates a polynomial $S(X)$ from $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ and the corrupted party shares received from $\mathcal{S}$. This polynomial defines the shares of the honest parties. The functionality samples and records the randomness r_i for the commitments $\mathsf{sk}_i G + r_i H$ output to $\mathcal{S}$. In H_7 this interface is fully implemented.
- As in the `CDKG-corrupt` interface of $\mathcal{F}_{\mathsf{cdkg}}$, the key shares are output to $\mathcal{S}$ in response to an adaptive corruption instruction. In H_7 this interface is fully implemented.
- As in the `CDKG-deliver` interface of $\mathcal{F}_{\mathsf{cdkg}}$, the functionality $\mathcal{F}$ does not forward the key shares received from $\mathcal{S}$ and instead outputs the recorded shares and commitments. In H_7 this interface is fully implemented

– *Modifications to $\mathcal{S}$:*

- As in Step 40b, the simulator equivocates the VSS commitments such that the secret shares returned to $\mathcal{A}$ are $(\mathsf{sk}_i, u_i, v_i)$ for sk_i received from $\mathcal{F}$.
- As in Step 20, the simulator copies the commitment D_i received from $\mathcal{F}$, and qs in Step 39, the randomness received from $\mathcal{F}$ are copied directly into the secret state of the newly corrupted party.

The view of $\mathcal{Z}$ is unchanged. The transcript otherwise reveals no information about the shares exchanged between the honest parties; the ciphertexts are random as of H_5, the commitments $\mathbf{A}_i$, and B_i, C_i are perfectly hiding. This modification is perfectly indistinguishable as the call to $\mathcal{E}_{\mathsf{vss}}^{(s,v)}$ has been proven to preserve that the shares are valid openings of these commitments. We refer to Theorem 2 for more details.

Hybrid H_8: *Rigged Group Elements Output by* $\mathtt{H}_{\mathbb{G}}$. In this hybrid, the simulator programs the random oracle $\mathtt{H}_{\mathbb{G}}$ such that the group element F is computed from $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$.

– *Modifications to $\mathcal{S}$:* As in Step 4(b)ii, the simulator programs the random oracle $\mathtt{H}_{\mathbb{G}}$ to output *rigged* group elements F, and records (α, s, F) such that $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} = sG + \alpha F$, after the receipt of $(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}})$ from $\mathcal{F}$. As of H_8 the emulation of $\mathcal{F}_{\mathsf{ro}}$ is fully implemented.

This change is perfectly indistinguishable; the distribution of the output of $\mathtt{H}_{\mathbb{G}}$ in H_8 is uniform in $\mathbb{G}$ since the scalar α is sampled uniformly, and the distribution of the output of $\mathtt{H}_{\mathbb{G}}$ in H_7 is also uniform.

Hybrid H_9: *Protocol VSS Shares* $(s, \alpha, 0)$ *and Not* $(\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}, 0, 0)$. In this hybrid, the simulator modifies the state of an honest party such that the protocol outputs the public key $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$, given that the group element F is rigged as in H_8.

– *Modifications to* $\mathcal{S}$*:*

- As in Step 29, the simulator no longer shares $s = \mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ and $\alpha = 0$ as in H_7, and instead sets α, s as in the record $(\mathsf{ssid}, \alpha, s, F)$ such that $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}} = sG + \alpha F$.
- In Step 40, on the corruption of the target party P_T *or* on the final corruption $|\mathcal{Q} \cap \mathcal{C}| = t-1$ the simulator follows the instructions in Step 40c, treating both cases as if this corruption is the final corruption. In H_{10} the simulator skips Step 40a.

The public key output by the protocol is indeed $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ received from the functionality. The simulation of the protocol modifies the state of the target party, such that it shared threshold shares of the secrets $(s_{i,0}, \alpha, 0, w_{i,0} + \tau_w^{-1}\tau_u\alpha)$. All parties have used $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{pd}})$ to prove that their VSS shared $(s_{j,0}, 0, 0, w_{j,0})$; the commitments $\{B_j\}$ interpolate to $sG + \alpha F + 0H_v = \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$. This hybrid is perfectly indistinguishable from H_8 as the call to $\mathcal{E}_{\mathtt{vss}}^{(\cdots)}$ has been proven to preserve that the shares are valid openings of these commitments. We refer to Theorem 2 for more details.

Hybrid H_{10}: *Retargeting Until Final Corruption.* In this hybrid the simulator allows for up to $t-1$ corruptions before correcting the states of all of the parties by *retargeting*, i.e., by changing the target party to another random honest party on the adaptive corruption of P_T.

– *Modifications to* $\mathcal{S}$*:* As in Step 40a when the target party P_T is corrupted the simulator samples a new target party $P_{\tilde{T}}$ and equivocates the state of P_T and $P_{\tilde{T}}$ such that the state of $P_{\tilde{T}}$ is the inconsistent state.

This hybrid is perfectly indistinguishable from H_9 as the call to $\mathcal{E}_{\mathtt{vss}}^{(\cdots)}$ has been proven to preserve that the shares are valid openings of these commitments. We refer to Theorem 2 for more details.

Hybrid H_{11}: *Functionality Keeps Secrets.* In this hybrid, $\mathcal{F}$ does not send $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ to $\mathcal{S}$ and the value remains secret until $|\mathcal{C}| = t$.

– *Modifications to* $\mathcal{F}$*:*

- As in the `CDKG-gen` interface of $\mathcal{F}_{\mathsf{cdkg}}$, on receipt of a query from every $j \in \mathcal{Q}$ in subsession ssid the functionality $\mathcal{F}$ outputs $(\mathtt{CDKG\text{-}pk}, \mathsf{ssid}, \mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}})$ and no longer includes $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ in the output to $\mathcal{A}^*$.

This change is perfectly indistinguishable; in H_{10}, the simulator $\mathcal{S}$ did not use $\mathsf{sk}_{\mathcal{F}_{\mathsf{dkg}}}$ to produce the view of $\mathcal{Z}$ until until the final corruption. In H_{11} $\mathcal{F} = \mathcal{F}_{\mathsf{cdkg}}$ and $\mathcal{S} = \mathcal{S}_{\mathsf{cdkg}}$ are fully implemented.

Hybrid H_{12}: *Ideal World.* This hybrid corresponds to the ideal execution, where the environment $\mathcal{Z}$ invokes $\mathcal{F}_{\mathsf{cdkg}}$ with $\mathcal{S}_{\mathsf{cdkg}}$. As of H_{11} the simulator $\mathcal{S} = \mathcal{S}_{\mathsf{cdkg}}$ and functionality $\mathcal{F} = \mathcal{F}_{\mathsf{cdkg}}$ are fully implemented. The random variable $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{cdkg}}, \mathcal{S}_{\mathsf{cdkg}}, \mathcal{Z}}(z)$ denotes the output on this interaction with $\mathcal{F}_{\mathsf{dkg}}$, $\mathcal{Z}$ and $\mathcal{S}_{\mathsf{cdkg}}$ on input z. This hybrid corresponds to the ensemble:

$$\{\mathsf{EXEC}_{\mathcal{F}_{\mathsf{cdkg}}, \mathcal{S}_{\mathsf{cdkg}}, \mathcal{Z}}(z)\}_{z \in \{0,1\}^*}.$$

Combining all of the steps we have that $\mathsf{H}_{12} \approx \mathsf{H}_0$ and $\Pi_{\mathsf{cdkg}}^{\mathsf{ia}}$ is UC-secure under the DDH assumption in $\mathcal{G}$ if Π_{eq} is AZK for $\mathcal{R}_{\mathsf{eq}}$ and Π_{dh} is ZK for $\mathcal{R}_{\mathsf{dh}}$.

Identifiable Abort. Our protocol satisfies identifiable abort if all parties in $\mathcal{Q} = \mathcal{P}$ for each subsession ssid broadcast a (potentially empty) message in both rounds, or if $\mathcal{F}_{\mathsf{bc}}$ has (known) bounded delay. We argue that our simulator $\mathcal{S}_{\mathsf{cdkg}}$ always queries either $(\mathtt{CDKG\text{-}iabort}, \mathsf{ssid}, \cdot)$ or $(\mathtt{CDKG\text{-}deliver}, \mathsf{ssid})$.

If all $n - |\mathcal{C}|$ honest parties invoke the functionality $\mathcal{F}_{\mathsf{cdkg}}$, then $\mathcal{S}_{\mathsf{cdkg}}$ receives $\mathsf{pk}_{\mathcal{F}_{\mathsf{dkg}}}$ and simulates all of the honest Round 1 messages. If all $|\mathcal{C}|$ corrupted parties activate $\mathcal{F}_{\mathsf{bc}}$ in Round 1, then $\mathcal{S}_{\mathsf{cdkg}}$ proceeds to Round 2 and simulates the $n - |\mathcal{C}|$ Round 2 honest party messages. If all $|\mathcal{C}|$ corrupted parties activate $\mathcal{F}_{\mathsf{bc}}$

in Round 2, then $\mathcal{S}_{\mathsf{cdkg}}$ processes the complaints. If any party broadcast a verifying complaint then $\mathcal{S}_{\mathsf{cdkg}}$ queries `CDKG-iabort`. Otherwise, $\mathcal{S}_{\mathsf{cdkg}}$ queries `CDKG-deliver`. Therefore, given that all $P_j \in \mathcal{P}$ broadcast a—potentially empty—message in the first and second rounds, and that all broadcasted messages are eventually delivered then $\mathcal{S}_{\mathsf{cdkg}}$ always queries either `CDKG-iabort` or `CDKG-deliver` in every subsession ssid.

□

5.3 Three Round Committed DKG Protocol with Guaranteed Output

For completeness, we also include a committed DKG protocol $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}$ in Figure 17 that can be seen as the combination of the previous two protocols; in three rounds parties essentially execute $\Pi^{\mathsf{go}}_{\mathsf{dkg}}$ and, along with the final round message, broadcast an independent pedersen commitment to their secret key share along with a straightline extractable proof that this commitment is consistent with the other committed values via $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$.

Figure 17: $\Pi^{\mathsf{god}}_{\mathsf{cdkg}}(\mathcal{G}, t)$.
Committed Distributed Key Generation with Guaranteed Output.

The DKG protocol parameterized by a group $\mathcal{G} := (\mathbb{G}, G, q)$ and threshold $t < n/2$ is executed between the n parties in $\mathcal{P}$ with access to:

- Functionalities: $\mathcal{F}_{\mathsf{bc}}$, $\mathcal{F}_{\mathsf{pki}}$, $\mathcal{F}_{\mathsf{ro}}$ instances $\mathtt{H}_{\mathbb{G}}$ and $\mathtt{H}_{\mathsf{sh}}$.
- ⋆ Additional functionality instance: $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$
- $\Pi_{\mathsf{dh}} = (\mathsf{Prv}^{\mathtt{H}_{\mathsf{dh}}}_{\mathsf{dh}}, \mathsf{Ver}^{\mathtt{H}_{\mathsf{dh}}}_{\mathsf{dh}})$ ZK and S for $\mathcal{R}_{\mathsf{dh}}$ (Defs 7, 9 10)
- $\Pi_{\mathsf{pd}} = (\mathsf{Prv}^{\mathtt{H}_{\mathsf{pd}}}_{\mathsf{pd}}, \mathsf{Ver}^{\mathtt{H}_{\mathsf{pd}}}_{\mathsf{pd}})$ aZK and SS for $\mathcal{R}_{\mathsf{pd}}$ (Defs 7, 9 10)
- $\Pi_{\mathsf{eq}} = (\mathsf{Prv}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}}, \mathsf{Ver}^{\mathtt{H}_{\mathsf{eq}}}_{\mathsf{eq}})$ aZK and SS for $\mathcal{R}_{\mathsf{eq}}$ (Defs 7, 9 10)

Below is the code of party $P_i \in \mathcal{P}$:
The contents of the state of P_i are highlighted and flushed right.

Initialization.

On $(\mathtt{CDKG\text{-}init}, i)$**:** k_i^{pki}

1. $K_i := k_i^{\mathsf{pki}} \cdot G$; $k_i^{\mathsf{pki}} \leftarrow_{\$} \mathbb{Z}_q$; $r_i \leftarrow_{\$} \{0,1\}^{\lambda}$
2. Query $(\mathtt{PKI\text{-}register}, i, k_i^{\mathsf{pki}}, K_i)$ to $\mathcal{F}_{\mathsf{pki}}$.
3. Broadcast $(\mathtt{CDKG\text{-}init}, i, r_i)$ to $\mathcal{P}$.

On $(\mathtt{CDKG\text{-}init}, j, r_j)$ **from all** $P_j \in \mathcal{P}$:

4. For $\mathsf{id} \in \{\mathsf{com}, u, v, w\}$: $H_{\mathsf{id}} := \mathtt{H}_{\mathbb{G}}(\mathsf{id}, ((1, r_1), \ldots, (n, r_n)))$.

5⋆ $\mathbf{G} := (G, H_{\mathsf{com}})$

6. Define $\mathbf{H} := (G, H_u, H_v, H_w)$.
7. Output initialized.

Commitment Key.

On $(\mathtt{CDKG\text{-}CK})$**:** Output $(\mathtt{CDKG\text{-}CK}, H_{\mathsf{com}})$, if the value is defined.

Key Generation

On $(\mathtt{CDKG\text{-}gen}, \mathsf{ssid}, \mathcal{Q}, i)$: $s_{i,0}, w_{i,0}, \mathbf{r}_i^{\mathsf{sh}}, r_i^{\mathsf{pd}}$

8. Define $\mathtt{BAD}_i := \emptyset$, $\mathcal{V} := \mathcal{Q}$.
9. $((\mathsf{sh}_{i,1}, \ldots, \mathsf{sh}_{i,n}), \mathbf{A}_i = (A_{i,0}, \cdots)) \leftarrow \mathtt{VSShare}(\mathbf{H}, (s_{i,0}, 0, 0, w_{i,0}); \mathbf{r}_i^{\mathsf{sh}})$
   $s_{i,0}, w_{i,0} \leftarrow_{\$} \mathbb{Z}_q$, $\mathbf{r}_i^{\mathsf{sh}} \leftarrow_{\$} \mathbb{Z}_q^{4\times(t-1)}$.
10. $\phi_i^{\mathsf{pd}} := ((G, H_w), A_{i,0})$
11. $\pi_i^{\mathsf{pd}} := \mathsf{Prv}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}(\phi_i^{\mathsf{pd}}, (s_{i,0}, w_{i,0}); r_i^{\mathsf{pd}})$; $r_i^{\mathsf{pd}} \leftarrow_{\$} \{0,1\}^{\lambda}$
12. For all $P_j \in \mathcal{Q} \setminus \{P_i\}$:
    (a) Retrieve (j, K_j) from $\mathcal{F}_{\mathsf{pki}}$.
    (b) $K_{i,j} := k_i^{\mathsf{pki}} \cdot K_j$
    (c) $\mathsf{ct}_{i,j} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid}, i, j), K_{i,j}) \oplus \mathsf{sh}_{i,j}$
13. Broadcast $(\mathsf{ssid}, i, \mathbf{A}_i, \pi_i^{\mathsf{pd}}, \{(j, \mathsf{ct}_{i,j})\}_{P_j \in \mathcal{Q}})$.

Complaints

On $(\mathsf{ssid}, j, \mathbf{A}_j, \pi_j^{\mathsf{pd}}, \{(j', \mathsf{ct}_{j,j'})\}_{P_{j'} \in \mathcal{Q}})$ **from** $P_j \in \mathcal{Q}$: $\{\mathsf{sh}_{j,i}\}_{P_j \in \mathcal{Q}}, \{r_{i,j}^{\mathsf{dh}}\}_{(j,\ldots)\in \mathtt{BAD}_i}$

14. Remove P_j from $\mathcal{V}$ if any of the following hold:
    – $(j', \mathsf{ct}_{j,j'})$ was not broadcast for some $P_{j'} \in \mathcal{Q}$.
    – $1 \neq \mathsf{Ver}_{\mathsf{pd}}^{\mathtt{H}_{\mathsf{pd}}}(A_{j,0}, \pi_j^{\mathsf{pd}})$ for $\mathbf{A}_j := (A_{j,0}, \ldots)$.
15. $\mathsf{sh}_{j,i} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid}, j, i), K_{i,j}) \oplus \mathsf{ct}_{j,i}$
16. If $1 \neq \mathtt{VSVerify}(\mathbf{H}, \mathbf{A}_j, \mathsf{sh}_{j,i}, i)$:
    (a) $\phi_{i,j}^{\mathsf{dh}} := (G, K_i, K_j, K_{i,j})$
    (b) $\pi_{i,j}^{\mathsf{dh}} \leftarrow \mathsf{Prv}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}\left(\phi_{i,j}^{\mathsf{dh}}, k_i^{\mathsf{pki}}; r_{i,j}^{\mathsf{dh}}\right)$; $r_{i,j}^{\mathsf{dh}} \leftarrow_{\$} \{0,1\}^{\lambda}$
    (c) Add $\left(\mathsf{ssid}, j, K_{i,j}, \pi_{i,j}^{\mathsf{dh}}\right) \rightarrow \mathtt{BAD}_i$.
17. Once received from all $P_j \in \mathcal{Q}$: Broadcast $(\mathsf{ssid}, i, \mathtt{BAD}_i)$.

Reveal the Public Key

On $(\mathsf{ssid}, j, \mathtt{BAD}_j)$ **from** $P_j \in \mathcal{V}$**:** $r_i^{\mathsf{eq}}, r_{i,D}$

18. Query $F \leftarrow \mathtt{H}_{\mathbb{G}}(\mathsf{ssid}, \mathsf{trans})$ with trans the transcript of ssid thus far.
19. Define $\mathbf{F} := (G, F, H_v)$
20. For all $(\mathsf{ssid}, j', K_{j,j'}, \pi_{j,j'}^{\mathsf{dh}}) \in \mathtt{BAD}_j$ remove $P_{j'}$ from $\mathcal{V}$ if:
    (a) $1 = \mathsf{Ver}_{\mathsf{dh}}^{\mathtt{H}_{\mathsf{dh}}}\left(\phi_{j,j'}^{\mathsf{dh}}, \pi_{j,j'}^{\mathsf{dh}}\right)$ for $\phi_{j,j'}^{\mathsf{dh}} := (G, K_j, K_{j'}, K_{j,j'})$, and
    (b) $1 \neq \mathtt{VSVerify}(\mathbf{H}, \mathbf{A}_{j'}, \mathsf{sh}_{j',j}, j)$ for $\mathsf{sh}_{j',j} := \mathtt{H}_{\mathtt{sh}}((\mathsf{ssid}, j', j), K_{j,j'}) \oplus \mathsf{ct}_{j',j}$
21. Once received from all $P_j \in \mathcal{V}$: continue.
22. $(s_i, u_i, v_i, w_i) \leftarrow \mathtt{AggSh}(\{\mathsf{sh}_{j,i}\}_{P_j \in \mathcal{V}})$
23. $C_i := (s_i, u_i, v_i, w_i) \cdot \mathbf{H}$
24. $B_i := (s_i, u_i, v_i) \cdot \mathbf{F}$
25. $\phi_i^{\mathsf{eq}} := (\mathbf{H}, \mathbf{F}, C_i, B_i)$
26. $\pi_i^{\mathsf{eq}} := \mathsf{Prv}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}(\phi_i^{\mathsf{eq}}, (s_i, u_i, v_i, w_i); r_i^{\mathsf{eq}})$; $r_i^{\mathsf{eq}} \leftarrow_{\$} \{0,1\}^{\lambda_P}$

27⋆ $D_i := (s_i, r_{i,D}) \cdot \mathbf{G}$; $r_{i,D} \leftarrow \mathbb{Z}_q$

28⋆ $\phi_i^{\mathsf{eqm}} := (\mathbf{F}, \mathbf{G}, B_i, D_i)$, $w_i := ((s_i, u_i, v_i), (s_i, r_{i,D}))$

29⋆ Query $(\texttt{ZK-prove}, \mathsf{ssid}, \phi_i^{\mathsf{eqm}}, w_i)$ to $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$.

30. Broadcast $\left(\mathsf{ssid}, i, B_i, \pi_i^{\mathsf{eq}}, D_i\right)$.

Output Computation

On the following for $P_j \in \mathcal{V}$:

– $(\mathsf{ssid}, j, B_j, \pi_j^{\mathsf{eq}}, D_j)$,

⋆ $(\texttt{ZK-proven}, \mathsf{ssid}, P_j, (\mathbf{F}, \mathbf{G}, B_j, D_j))$ from $\mathcal{F}_{\mathsf{azk}}(\mathcal{R}_{\mathsf{eqm}})$

31. $C_j := \sum_{k=0}^{t-1} \sum_{P_{j'} \in \mathcal{V}} (j^k) A_{j',k}$

32. Remove P_j from $\mathcal{V}$ if $1 \neq \mathsf{Ver}_{\mathsf{eq}}^{\mathtt{H}_{\mathsf{eq}}}\left(((\mathbf{H}, \mathbf{F}), C_j, B_j), \pi_j^{\mathsf{eq}}\right)$.

Once verifying messages received from t parties in $\mathcal{V}$: continue.

33. $B(X) \leftarrow \texttt{Interpolate}(\{(j, B_j)\}_{j \in \mathcal{V}})$

34. $\mathsf{pk} := B(0)$, $\mathsf{sk}_i := s_i$

35. **Output** $\left(\texttt{CDKG-keys}, \mathsf{pk}, \mathsf{sk}_i, r_{i,D}, \{D_j\}_{P_j \in \mathcal{V}}\right)$.

Theorem 4 (Security of $\Pi_{\mathsf{cdkg}}^{\mathsf{god}}$). *For all $\mathcal{G}$ and all $t < n$ protocol $\Pi_{\mathsf{cdkg}}^{\mathsf{god}}(\mathcal{G}, t)$ UC-realizes functionality $\mathcal{F}_{\mathsf{cdkg}}(\mathcal{G}, t)$ in the $(\mathcal{F}_{\mathsf{azk}}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{pki}}, \mathcal{F}_{\mathsf{ro}})$-hybrid model in the presence of an adaptive malicious adversary under the DDH assumption in $\mathcal{G}$, given that Π_{dh}, Π_{pd}, and Π_{dh} are ZK+S, AZK+SS, and AZK+SS NIZKs (Definitions 7, 8, 9, 10), respectively. Moreover, the protocol achieves guaranteed output under the assumption that $t < n/2$ under the condition that $\mathcal{F}_{\mathsf{bc}}$ guarantees delivery of all messages.*

6 Adaptively Secure Threshold Schnorr

In this section we recall the threshold schnorr functionality $\mathcal{F}_{\mathsf{sch}}$, and present a three round protocol $\Pi_{\mathsf{sch}}^{\mathsf{ia}}$ that UC-realizes the functionality. The protocol achieves identifiable abort in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$-hybrid model against an adaptive adversary.

6.1 The Ideal Threshold Schnorr Functionality

The ideal threshold Schnorr functionality $\mathcal{F}_{\mathsf{sch}}$ appears in Figure 18. The functionality is standard. Notably, the Schnorr hash function $\mathtt{H}_{\mathsf{sch}} : \{0,1\}^* \to \mathbb{Z}_q$ is explicitly passed as a parameter to $\mathcal{F}_{\mathsf{sch}}$ and is not modeled as a programmable random oracle.

Figure 18: $\mathcal{F}_{\mathsf{sch}}(\mathcal{G}, t, \mathtt{H}_{\mathsf{sch}})$.
Ideal Threshold Schnorr Signing Functionality.

The functionality $\mathcal{F}_{\mathsf{sch}}$ is parameterized by the group $\mathcal{G} = (\mathbb{G}, G, q)$, the threshold $t < n$, and the function $\mathtt{H}_{\mathsf{sch}} : \{0,1\}^* \to \mathbb{Z}_q$. The functionality interacts with the n parties in $\mathcal{P}$ and the ideal adversary $\mathcal{A}^*$ via the queries below, all of which implicitly include a session identifier sid. Upon invocation the functionality receives the set of corrupted parties $\mathcal{C} \subset \mathcal{P}$ from $\mathcal{A}^*$ and initializes a set of empty tables $\texttt{status}[], \texttt{nonces}[], \texttt{sigs}[]$.

On $(\texttt{SCH-init}, i)$ **from** $P_i \in \mathcal{P}$**.** If not already recorded, record and send $(\texttt{SCH-init}, i)$ to $\mathcal{A}^*$. If $(\texttt{SCH-init}, j)$ is recorded for all $P_j \in \mathcal{P} \setminus \mathcal{C}$ sample $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$, define $\mathsf{pk} := \mathsf{sk} \cdot G$, record $(\texttt{SCH-keys}, \mathsf{pk}, \mathsf{sk})$, set $\texttt{status}[\mathsf{sid}] := \texttt{ready}$, and send $(\texttt{SCH-pk}, \mathsf{pk})$ to $\mathcal{A}^*$.

On $(\texttt{SCH-deliverPK})$ **from** $\mathcal{A}^*$**.** Ignore unless $\texttt{status}[\mathsf{sid}] = \texttt{ready}$. Overwrite $\texttt{status}[\mathsf{sid}] = \top$. Retrieve $(\texttt{SCH-keys}, \mathsf{pk}, \cdot)$. Output $(\texttt{SCH-pk}, \mathsf{pk})$ to all $P_i \in \mathcal{P} \setminus \mathcal{C}$.

On $(\texttt{SCH-abortkey}, j \in \mathcal{C})$ **from** $\mathcal{A}^*$**.** Ignore if $\texttt{status}[\mathsf{sid}] = \top$. Overwrite $\texttt{status}[\mathsf{sid}] := \bot$. Output $(\texttt{SCH-abortkey}, j)$ to all $P_i \in \mathcal{P} \setminus \mathcal{C}$.

On $(\texttt{SCH-sign}, \mathsf{ssid}, \mathcal{Q}, i, m)$ **from** $P_i \in \mathcal{Q}$**.** Ignore if $|\mathcal{Q}| < t$, $\mathcal{Q} \not\subseteq \mathcal{P}$ or if there is a record $(\texttt{SCH-sign}, \mathsf{ssid}, \mathcal{Q}', m')$ such that $(\mathcal{Q}', m') \neq (\mathcal{Q}, m)$. If there does not exist a record $(\texttt{SCH-sign}, \mathsf{ssid}, \cdot, \cdot)$ then record and send $(\texttt{SCH-sign}, \mathsf{ssid}, \mathcal{Q}, m)$ to $\mathcal{A}^*$. Record and send $(\texttt{SCH-sign}, \mathsf{ssid}, i)$ to $\mathcal{A}^*$. If there is a record $(\texttt{SCH-sign}, \mathsf{ssid}, j)$ for all $P_j \in \mathcal{Q} \setminus \mathcal{C}$ sample $k \leftarrow_\$ \mathbb{Z}_q$, define $R := k \cdot G$, $e \leftarrow \mathtt{H}_{\mathsf{sch}}(\mathsf{pk}, R, m)$, $s := k + \mathsf{sk} \cdot e$, record $\texttt{status}[\mathsf{ssid}] := \texttt{ready}$, $\texttt{nonces}[\mathsf{ssid}] := (R, k)$, $\texttt{sigs}[\mathsf{ssid}] := (s, e)$, and send $(\texttt{SCH-sig}, \mathsf{ssid}, (s, e))$ to $\mathcal{A}^*$.

On $(\texttt{SCH-deliverSIG}, \mathsf{ssid})$ **from** $\mathcal{A}^*$**.** If $\texttt{status}[\mathsf{ssid}] = \texttt{ready}$ overwrite $\texttt{status}[\mathsf{ssid}] := \top$ and send $(\texttt{SCH-sig}, \mathsf{ssid}, \texttt{sigs}[\mathsf{ssid}])$ to $\mathcal{P}$.

On $(\texttt{SCH-abortsig}, \mathsf{ssid}, j)$ **from** $\mathcal{A}^*$**.** Ignore if either $\texttt{status}[\mathsf{ssid}] = \top$ or $P_j \notin \mathcal{Q} \cap \mathcal{C}$ for the record $(\texttt{SCH-sign}, \mathsf{ssid}, \mathcal{Q}, \cdot)$. Overwrite $\texttt{status}[\mathsf{ssid}] := \bot$. Output $(\texttt{SCH-abortsig}, \mathsf{ssid}, j)$ to all $P_i \in \mathcal{Q} \setminus \mathcal{C}$.

On $(\texttt{SCH-corrupt}, i)$ **from** $\mathcal{A}^*$**.** Add P_i to $\mathcal{C}$. If there exists a record $(\texttt{SCH-keys}, \mathsf{pk}, \mathsf{sk})$ and $|\mathcal{C}| = t$ then send $(\texttt{SCH-corrupted}, \mathsf{sk}, \texttt{nonces})$ to $\mathcal{A}^*$.

6.2 Three Round Threshold Schnorr with Identifiable Abort

For a graphical depiction of the communication flow of $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ see Figure 3. The protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ in Figure 19 is written in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$-hybrid model. Both key generation and nonce generation are computed with $\mathcal{F}_{\mathsf{cdkg}}$. Key generation outputs a public key pk, key share sk_i, and commitments to the key shares $\mathbf{PK} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. In subsession ssid, nonce generation outputs a public nonce R, nonce share k_i, commitments to nonce shares $\mathbf{R} = (R_1, \ldots, R_n)$, and randomness r'_i such that $R_i = k_i \cdot G + r'_i \cdot H$. A partial signature is a standard schnorr partial signature $\sigma_i = k_i + e \cdot \mathsf{sk}_i$ along with a corresponding value $\rho_i = r'_i + e \cdot r_i$.

Figure 19: $\Pi^{\mathsf{ia}}_{\mathsf{sch}}(\mathcal{G}, t, \mathtt{H}_{\mathsf{sch}})$**.**
Threshold Schnorr Protocol.

The Schnorr protocol, parameterized by a group $\mathcal{G} = (\mathbb{G}, G, q)$, the threshold t, and the function $\mathtt{H}_{\mathsf{sch}} : \{0,1\}^* \to \mathbb{Z}_q$, is executed by the set of n parties in $\mathcal{P}$.

Below is the code of party $P_i \in \mathcal{P}$.
The contents of the state of P_i are highlighted and flushed right.

KEY GENERATION.

On $(\texttt{SCH-init}, i)$**:** sk_i, r_i

1. Query $(\texttt{CDKG-init}, i, \mathcal{P})$ to $\mathcal{F}_{\mathsf{cdkg}}(\mathcal{G}, t)$.
   – Await $(\texttt{CDKG-initialized})$ from $\mathcal{F}_{\mathsf{cdkg}}$.
2. Query $(\texttt{CDKG-CK})$ to $\mathcal{F}_{\mathsf{cdkg}}$.
   – Await $(\texttt{CDKG-CK}, H)$ from $\mathcal{F}_{\mathsf{cdkg}}$.

3. Query $(\texttt{CDKG-gen}, 0, i, \mathcal{P})$ to $\mathcal{F}_{\mathsf{cdkg}}$:
   - On $(\texttt{CDKG-iabort}, 0, j)$ in response:
     Output $(\texttt{SCH-abortkey}, j)$.
4. Await $(\texttt{CDKG-keys}, 0, \mathsf{pk}, \mathsf{sk}_i, \mathbf{PK}, r_i)$ from $\mathcal{F}_{\mathsf{cdkg}}$.
5. Output $(\texttt{SCH-pk}, \mathsf{pk})$.

Threshold Signing.

On $(\texttt{SCH-sign}, \mathsf{ssid}, \mathcal{Q}, i, m)$**:** k_i, r'_i

6. Query $(\texttt{CDKG-gen}, \mathsf{ssid}, i, \mathcal{Q})$ to $\mathcal{F}_{\mathsf{cdkg}}$.
   - On $(\texttt{CDKG-iabort}, \mathsf{ssid}, j)$:
     Output $(\texttt{SCH-abortsig}, \mathsf{ssid}, j)$.
7. Await $(\texttt{CDKG-pk}, \mathsf{ssid}, R, k_i, \mathbf{R}, r'_i)$ from $\mathcal{F}_{\mathsf{cdkg}}$.
8. Compute $e \leftarrow \mathtt{H}_{\mathsf{sch}}(\mathsf{pk}, R, m)$.
9. Define $\sigma_i := k_i + e \cdot \mathsf{sk}_i$.
10. Define $\rho_i := r'_i + e \cdot r_i$.
11. Broadcast $(\mathsf{ssid}, i, \sigma_i, \rho_i)$.

On $(\mathsf{ssid}, j, \sigma_j, \rho_j)$ **from** $P_j \in \mathcal{Q}$**:**

12. On the first such message, define $\mathcal{V} := \mathcal{Q}$.
13. Remove P_j from $\mathcal{V}$ if, for $(\cdots, \mathsf{pk}_j, \cdots) = \mathbf{PK}$ and $(\cdots, R_j, \cdots) = \mathbf{R}$:
$$\sigma_j \cdot G + \rho_j \cdot H \neq R_j + e \cdot \mathsf{pk}_j.$$
14. If $|\mathcal{V}| < t$: Output $(\texttt{SCH-abortsig}, \mathsf{ssid}, j)$ for $P_j \in \mathcal{Q} \setminus \mathcal{V}$ with smallest j.
15. Await t verifying messages.
16. Compute $s := S(0)$ for $S(X) \leftarrow \texttt{Interpolate}(\{(j, \sigma_j)\}_{P_j \in \mathcal{V}})$.
17. Output $(\texttt{SCH-sig}, \mathsf{ssid}, (s, e))$ to $\mathcal{Z}$.

Theorem 5 (Security of $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$). *For all $\mathcal{G}$, $t < n$, and $\mathtt{H}_{\mathsf{sch}}$ protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}(\mathcal{G}, t, \mathtt{H}_{\mathsf{sch}})$ UC-realizes functionality $\mathcal{F}_{\mathsf{sch}}(\mathcal{G}, t, \mathtt{H}_{\mathsf{sch}})$ in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$-hybrid model in the presence of an adaptive malicious adversary corrupting $t < n$ parties under the DL assumption in $\mathcal{G}$. Moreover, if $\mathcal{F}_{\mathsf{bc}}$ guarantees the delivery of all messages then protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ achieves security with identifiable abort when $\mathcal{F}_{\mathsf{cdkg}}$ satisfies identifiable abort, and for $t < n/2$ protocol $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ achieves guaranteed output delivery when $\mathcal{F}_{\mathsf{cdkg}}$ satisfies guarantees output delivery.*

Proof. The simulator $\mathcal{S}_{\mathsf{sch}}$ is presented in Figure 20.

Figure 20: $\mathcal{S}_{\mathsf{sch}}$.
Schnorr Simulator for $\Pi^{\mathsf{ia}}_{\mathsf{sch}}(\mathcal{G}, t, \mathtt{H}_{\mathsf{sch}})$, Theorem 5.

The simulator forwards all communication between $\mathcal{A}$ and $\mathcal{Z}$ without modification, and upon invocation receives the set $\mathcal{C} \subseteq \mathcal{P}$ of corrupted parties.

On Invocation.
The simulator makes records in initially empty tables:
- $\texttt{SCH-status}[\mathsf{ssid}]$: records if the signature of of ssid was delivered.

– $\texttt{sigs}[\mathsf{ssid}][i]$: records the signature of party P_i sent in signing ssid.
– $\texttt{CK}[c]$: records the commitment keys generated by $\mathcal{A}$ via $\mathcal{F}_{\mathsf{cdkg}}$.
– $\texttt{DKG-status}[\mathsf{ssid}][i]$ records `init` if the party has initialized ssid via $\mathcal{F}_{\mathsf{cdkg}}$, but when $i = \mathcal{S}$ records the session status as `defined`, `ready`, $\perp$, or $\top$.
– $\texttt{DKG-pub}[\mathsf{ssid}][i] = R_i$: records the commitment of party P_i in ssid; $i = \mathcal{S}$ records R received from $\mathcal{F}_{\mathsf{cdkg}}$; $\mathsf{ssid} = 0$ corresponds to pk.
– $\texttt{rands}[i][\mathsf{ssid}] = r'_i$: records randomness r'_i.
– $\texttt{DKG-sec}[i][\mathsf{ssid}] = k_i$: records the secret share of P_i in ssid; $\mathsf{ssid} = 0$ corresponds to sk_i.

The relations between the elements of the tables is clarified below, where the overhead for adaptive security is highlighted.

$$
\begin{aligned}
\mathsf{pk} &= \texttt{DKG-pub}[\mathcal{S}][0] = \texttt{DKG-sec}[0][\mathcal{S}] \cdot G + \texttt{rands}[0][\mathcal{S}] \cdot H \\
\mathsf{pk}_i &= \texttt{DKG-pub}[i][0] = \texttt{DKG-sec}[0][i] \cdot G + \texttt{rands}[0][i] \cdot H \\
R &= \texttt{DKG-pub}[\mathsf{ssid}][\mathcal{S}] = \texttt{DKG-sec}[\mathcal{S}][\mathsf{ssid}] \cdot G + \texttt{rands}[\mathcal{S}][\mathsf{ssid}] \cdot H \\
R_i &= \texttt{DKG-pub}[\mathsf{ssid}][i] = \texttt{DKG-sec}[i][\mathsf{ssid}] \cdot G + \texttt{rands}[i][\mathsf{ssid}] \cdot H \\
e_{\mathsf{ssid}} &= \mathsf{H}_{\mathsf{sch}}\left(\texttt{DKG-pub}[0][\mathcal{S}], \texttt{DKG-pub}[\mathsf{ssid}][\mathcal{S}], m\right) \\
\begin{pmatrix}\sigma_i \\ \rho_i\end{pmatrix} &= \texttt{sigs}[\mathsf{ssid}][i] = \begin{pmatrix}\texttt{DKG-sec}[i][\mathsf{ssid}] + e_{\mathsf{ssid}} \cdot \texttt{DKG-sec}[i][0] \\ \texttt{rands}[i][\mathsf{ssid}] + e_{\mathsf{ssid}} \cdot \texttt{rands}[i][0]\end{pmatrix}
\end{aligned}
$$

Hybrid Functionalities.

$\mathcal{F}_{\mathsf{bc}}$. Emulate honestly, as detailed in Section 2.1.

$\mathcal{F}_{\mathsf{cdkg}}(\mathcal{G}, t)$. Emulate as described below.

1. On $(\texttt{CDKG-init}, i)$ from simulated $P_i \in \mathcal{P} \setminus \mathcal{C}$:
   – Send $(\texttt{CDKG-init}, i, \mathcal{P})$ to $\mathcal{A}$.
2. Accept the queries below from $\mathcal{A}$ if $\texttt{DKG-pub}[\mathcal{S}][0] = \mathsf{pk}$ recorded but $(\texttt{CDKG-CK}, [\cdots])$ not recorded:
   – On $(\texttt{CDKG-newCK}, c)$ from $\mathcal{A}$:
   • Sample $\tau \leftarrow \mathbb{Z}_q$, define $H := \tau \cdot G$.
   • Record $\texttt{CK}[c] := (\tau, H)$.
   • Output $(\texttt{CDKG-newCK}, H)$ to $\mathcal{A}$.
   – On $(\texttt{CDKG-setCK}, c)$ from $\mathcal{A}$:
   • Retrieve $(\tau, H) := \texttt{CK}[c]$.
   • Record $(\texttt{CDKG-CK}, \tau, H)$.
3. On $(\texttt{CDKG-gen}, \mathsf{ssid}, i, \mathcal{Q})$ from a simulated $P_i \in \mathcal{P} \setminus \mathcal{C}$:
   (a) Once the query is simulated by all $P_i \in \mathcal{Q} \setminus \mathcal{C}$:
   Record $\texttt{DKG-status}[\mathsf{ssid}] := \mathsf{defined}$.
   (b) For each $P_i \in \mathcal{P} \setminus \mathcal{C}$:
   – Sample $d_i \leftarrow \mathbb{Z}_q$, define $D_i := d_i \cdot G$.
   – Record $\texttt{rands}[i][\mathsf{ssid}] := d_i$ and $\texttt{DKG-pub}[\mathsf{ssid}][i] := D_i$.

(c) If $\mathsf{ssid} = 0$, $\mathcal{Q} = \mathcal{P}$, and $\texttt{DKG-status}[0] := \mathsf{defined}$:
- Retrieve $\mathsf{pk} := \texttt{DKG-pub}[\mathcal{S}][0]$.
- Send $(\texttt{CDKG-pk}, \mathsf{ssid}, \mathsf{pk}, \texttt{coms}[\mathsf{ssid}])$ to $\mathcal{A}$.

(d) Else if $\texttt{DKG-status}[\mathsf{ssid}] := \mathsf{defined}$:
- Retrieve $R := \texttt{DKG-pub}[\mathcal{S}][\mathsf{ssid}]$.
- Send $(\texttt{CDKG-pk}, \mathsf{ssid}, R, \texttt{coms}[\mathsf{ssid}])$ to $\mathcal{A}$.

4. On $(\texttt{CDKG-shares}, \mathsf{ssid}, \{(j, s_j, r_j)\}_{P_j \in \mathcal{Q} \cap \mathcal{C}})$ from $\mathcal{A}$:
- If $\texttt{DKG-status}[\mathsf{ssid}] \neq \mathsf{defined}$: ignore.
- For each $P_j \in \mathcal{Q} \cup \mathcal{C}$:
  - Define $D_j := s_j \cdot G + r_j \cdot H$.
  - Record $\texttt{DKG-pub}[\mathsf{ssid}][j] := D_j$
  - Record $\texttt{DKG-sec}[j][\mathsf{ssid}] := s_j$.
  - Record $\texttt{rands}[j][\mathsf{ssid}] := r_j$.
- Set $\texttt{DKG-status}[\mathsf{ssid}] := \mathsf{ready}$.

5. On $(\texttt{CDKG-iabort}, \mathsf{ssid}, j)$ from $\mathcal{A}$:
- If $\texttt{DKG-status}[\mathsf{ssid}] \notin \{\mathsf{defined}, \mathsf{ready}\}$: ignore.
- Record $\texttt{DKG-status}[\mathsf{ssid}] := (\perp, j)$
- If $\mathsf{ssid} = 0$:

  Go to Step 10 to abort Schnorr key generation.
- Else:

  Go to Step 14 to abort Schnorr signing session.

6. On $(\texttt{CDKG-deliver}, \mathsf{ssid})$ from $\mathcal{A}$:
- If $\texttt{DKG-status}[\mathsf{ssid}] \neq \mathsf{ready}$: ignore.
- Record $\texttt{DKG-status}[\mathcal{S}][\mathsf{ssid}] := \top$.
- If $\mathsf{ssid} = 0$:

  Go to Step 11 to simulate delivery of Schnorr public key.
- Else:

  Go to Step 15 to simulate delivery of Schnorr nonce.

KEY GENERATION.

On $(\texttt{SCH-init}, i)$ **from** $\mathcal{F}_{\mathsf{sch}}$ **for** $P_i \in \mathcal{P} \setminus \mathcal{C}$**:**

7. Emulate $\mathcal{F}_{\mathsf{cdkg}}$ as in Step 1:

   Receipt of $(\texttt{CDKG-init}, i)$.
8. Once $(\texttt{SCH-init}, i)$ received from $\mathcal{F}_{\mathsf{sch}}$ for all $P_i \in \mathcal{P} \setminus \mathcal{C}$:
- Receive $(\texttt{SCH-pk}, \mathsf{pk})$ from $\mathcal{F}_{\mathsf{sch}}$.
- Record $\texttt{DKG-pub}[\mathcal{S}][0] := \mathsf{pk}$.
- Emulate $\mathcal{F}_{\mathsf{cdkg}}$ as in Step 2:

  Open interfaces $(\texttt{CDKG-newCK})$ and $(\texttt{CDKG-setCK})$ to $\mathcal{A}$.

9. Once $(\texttt{CDKG-CK}, \tau, H)$ recorded:
- Emulate $\mathcal{F}_{\mathsf{cdkg}}$ as in Step 3c:

  Receipt of $(\texttt{CDKG-gen}, 0, i, \mathcal{P})$ from all $P_i \in \mathcal{P} \setminus \mathcal{C}$.

10. Else if $\texttt{DKG-status}[0] = (\perp, j)$:
    - Query $(\texttt{SCH-abortkey}, j)$ to $\mathcal{F}_{\mathsf{sch}}$.
11. Once $\texttt{DKG-status}[0] = \top$:
    - Query $(\texttt{SCH-deliverPK})$ to $\mathcal{F}_{\mathsf{sch}}$.

Threshold Signing.

On $(\texttt{SCH-sign}, \mathsf{ssid}, \mathcal{Q}, m, i)$ **from** $\mathcal{F}_{\mathsf{sch}}$ **for** $P_i \in \mathcal{Q} \setminus \mathcal{C}$**:**

12. Emulate $\mathcal{F}_{\mathsf{cdkg}}$ in Step 3:
    Receipt of $(\texttt{CDKG-gen}, \mathsf{ssid}, i, \mathcal{Q})$.
13. On $(\texttt{SCH-sig}, \mathsf{ssid}, (s, e))$ from $\mathcal{F}_{\mathsf{sch}}$:
    - Compute $R := s \cdot G - e \cdot \mathsf{pk}$.
    - Define $\texttt{DKG-pub}[\mathcal{S}][\mathsf{ssid}] := R$.
14. If $\texttt{DKG-status}[\mathsf{ssid}] = (\perp, j)$:
    - Query $\mathcal{F}_{\mathsf{sch}}$ with $(\texttt{SCH-abortsig}, \mathsf{ssid}, j)$.
15. Once $\texttt{DKG-status}[\mathsf{ssid}][\mathcal{S}] = \top$:
    (a) For each $P_j \in \mathcal{Q} \cap \mathcal{C}$:
        - Retreive $\mathsf{sk}_j := \texttt{DKG-sec}[j][0]$, $r_j := \texttt{rands}[j][0]$.
        - Retreive $k_j := \texttt{DKG-sec}[j][\mathsf{ssid}]$, $r'_j := \texttt{rands}[j][\mathsf{ssid}]$.
        - Define $\sigma_j := k_j + e \cdot \mathsf{sk}_j$, $\rho_j := r'_j + e \cdot r_j$.
        - Record $\texttt{sigs}[\mathsf{ssid}][j] := (\sigma_j, \rho_j)$.
    (b) Interpolate $S(X)$ of degree $t-1$ such that $S(0) = s$ and $S(j) = \sigma_j$ for all $P_j \in \mathcal{Q} \cap \mathcal{C}$.
    (c) For each $P_i \in \mathcal{Q} \setminus \mathcal{C}$:
        - Define $\sigma_i := S(i)$.
        - Retreive $d_i := \texttt{rands}[i][0]$, $d'_i := \texttt{rands}[i][\mathsf{ssid}]$.
        - Define $\rho_j := \tau^{-1}(d'_i + e \cdot d_i - \sigma_i)$.
        - Record $\texttt{sigs}[\mathsf{ssid}][i] := (\sigma_i, \rho_i)$.
        - Broadcast $(\mathsf{ssid}, i, \sigma_i, \rho_i)$ to $\mathcal{Q}$ via $\mathcal{F}_{\mathsf{bc}}$.

On $(\mathsf{ssid}, j, \sigma_j^*, \rho_j^*)$ **from** $\mathcal{A}$ **for** $P_j \in \mathcal{Q} \cap \mathcal{C}$**:**

16. On the first message define $\mathcal{V} := \mathcal{Q}$.
17. Retreive $\mathsf{pk}_j := \texttt{DKG-pub}[0][j]$, $R_j := \texttt{DKG-pub}[\mathsf{ssid}][j]$.
18. If $(\sigma_j^*, \rho_j^*) \neq \texttt{sigs}[j][\mathsf{ssid}]$: remove P_j from $\mathcal{V}$.
19. If $|\mathcal{V}| < t$:
    - Query $(\texttt{SCH-abortsig}, \mathsf{ssid}, j)$ for $P_j \in \mathcal{Q} \setminus \mathcal{V}$ with smallest j.
20. On t verifying messages from $P \in \mathcal{V}$ (including those from $\mathcal{Q} \setminus \mathcal{C}$ released by $\mathcal{A}$ via $\mathcal{F}_{\mathsf{bc}}$):
    - Query $(\texttt{SCH-deliverSIG}, \mathsf{ssid})$ to $\mathcal{F}_{\mathsf{sch}}$.

Adaptive Corruption.

On $(\texttt{SCH-corrupt}, i)$ **from** $\mathcal{A}$ **for** $P_i \in \mathcal{P} \setminus \mathcal{C}$**:**

21. Send $(\texttt{SCH-corrupt}, i)$ to $\mathcal{F}_{\mathsf{sch}}$.
22. If $\texttt{DKG-status}[0][\mathcal{S}] \notin \{\texttt{ready}, \perp, \top\}$ or if $|\mathcal{C}| \geq t$: go to Step 25.
23. If $|\mathcal{C}| < t - 1$:

(a) Sample $\mathsf{sk}_i \leftarrow_\$ \mathbb{Z}_q$.
(b) Retrieve $s_i \leftarrow \texttt{rands}[i][0]$.
(c) Record $\texttt{DKG-sec}[i][0] := \mathsf{sk}_i$.
(d) Overwrite $\texttt{rands}[i][0] := r_i = \tau^{-1}(s_i - \mathsf{sk}_i)$.
(e) For each ssid such that
- $\texttt{DKG-pub}[\mathsf{ssid}][i] \neq \emptyset$, and
- $\texttt{DKG-status}[\mathsf{ssid}][\mathcal{S}] \in \{\texttt{ready}, \bot, \top\}$:

i. Retrieve $R_i := \texttt{DKG-pub}[\mathsf{ssid}][i]$.
ii. Retrieve $d_i := \texttt{rands}[i][\mathsf{ssid}]$.
iii. If $\texttt{sigs}[\mathsf{ssid}][i] \neq \emptyset$:
- Retrieve $(\sigma_i, \rho_i) \leftarrow \texttt{sigs}[\mathsf{ssid}][i]$.
- Define $k_i := \sigma_i - e\mathsf{sk}_i$.

iv. Else: $k_i \leftarrow_\$ \mathbb{Z}_q$.
v. Set $\texttt{DKG-sec}[i][\mathsf{ssid}] := k_i$.
vi. Set $\texttt{rands}[i][\mathsf{ssid}] := r'_i = \tau^{-1}(d_i - k_i)$.

24. Else if $|\mathcal{C}| = t - 1$:
(a) Receive sk from $\mathcal{F}_{\mathsf{sch}}$.
(b) Define $\texttt{DKG-sec}[\mathcal{S}][0] := \mathsf{sk}$.
(c) Interpolate $S(X)$ such that $S(0) = \mathsf{sk}$ and $S(j) = \mathsf{sk}_j := \texttt{DKG-sec}[j][0]$ for all $P_j \in \mathcal{C}$.
(d) For all $P_k \in \mathcal{P} \setminus \mathcal{C}$:
i. Retrieve $s_k \leftarrow \texttt{rands}[k][0]$.
ii. Define $\texttt{DKG-sec}[k][0] := S(k)$.
iii. Record $\texttt{rands}[k][0] := \tau^{-1}(s_k - S(k))$.
(e) For each ssid such that $\texttt{sigs}[\mathsf{ssid}][\mathcal{S}]$ is non-empty:
i. Retrieve $(\sigma, e) \leftarrow \texttt{sigs}[\mathsf{ssid}][\mathcal{S}]$.
ii. Record $\texttt{DKG-sec}[\mathcal{S}][\mathsf{ssid}] := k = \sigma - e \cdot \mathsf{sk}$.
iii. For all $P_j \in \mathcal{C}$: Retrieve $k_j := \texttt{DKG-sec}[j][\mathsf{ssid}]$.
iv. Interpolate $R(X)$ such that $R(0) = k$ and $R(j) = k_j$ for all $P_j \in \mathcal{C}$.
v. For all $P_k \in \mathcal{P} \setminus \mathcal{C}$:
- Retrieve $d_k \leftarrow \texttt{rands}[k][\mathsf{ssid}]$.
- Record $\texttt{DKG-sec}[k][\mathsf{ssid}] := P(k)$.
- Record $\texttt{rands}[k][\mathsf{ssid}] := \tau^{-1}(d_k - P(k))$.

25. Add $P_i \to \mathcal{C}$.
26. Output $\mathsf{st}_i := (\texttt{CDKG-corrupt}, i, \texttt{DKG-sec}[i], \texttt{rands}[i])$ to $\mathcal{A}$.

Hybrid Argument. In the following we argue indistinguishability of the real and ideal worlds with a hybrid argument.

Hybrid H_0: *Real World.* The real execution of $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$, where the environment $\mathcal{Z}$ runs the protocol of Figure 19 with the n parties in $\mathcal{P}$ and the adversary $\mathcal{A}$, all with access to ideal functionalities $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$. The random variable $\mathsf{EXEC}_{\Pi^{\mathsf{ia}}_{\mathsf{sch}},\mathcal{A},\mathcal{Z}}(z)$ denotes the output of an execution of $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ with $\mathcal{Z}$ and $\mathcal{A}$ on input z. Executions of the protocol in H_0 corresponds to the ensemble:

$$\{\mathsf{EXEC}_{\Pi^{\mathsf{ia}}_{\mathsf{sch}},\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}.$$

Hybrid H_1: *Ideal Layout.* The existing ITIs $(\mathcal{A}, \mathcal{P}, \mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathsf{cdkg}})$ are grouped into a single machine called the simulator $\mathcal{S}$ that forwards communication between $\mathcal{A}$ and $\mathcal{Z}$ unchanged. Between $\mathcal{Z}$ and $\mathcal{S}$ are new dummy party nodes for each party in $\mathcal{P}$ and a functionality node $\mathcal{F}$ that, for now, forward inputs and outputs between $\mathcal{S}$ and the dummy parties, unchanged. This change is syntactic, so the view of $\mathcal{Z}$ is unchanged; H_1 is perfectly indistinguishable from H_0.

Hybrid H_2: *Functionality Record Keeping & Queries on Behalf of Corrupted Parties.* The simulator generates queries to $\mathcal{F}$ on behalf of the corrupted parties from the messages received from $\mathcal{A}$ in the simulated protocol executions. The functionality and the simulator now make all relevant records. The changes to $\mathcal{F}$ and $\mathcal{S}$ are listed below.

– *Modifications to $\mathcal{F}$:*

- Record keeping is done as in Figure 18.
- As in the `SCH-init`, `SCH-sign` interfaces of $\mathcal{F}_{\mathsf{sch}}$: the functionality $\mathcal{F}$ collects and records queries made by the honest dummy parties and $\mathcal{A}^*$. In H_2 these interfaces are not yet fully implemented; the functionality merely forwards inputs and outputs between the dummy parties and $\mathcal{S}$.
- As in the `SCH-deliverPK` interface of $\mathcal{F}_{\mathsf{sch}}$: the functionality $\mathcal{F}$ delivers the public key pk upon receipt of the query. In H_2 the interface is not yet fully implemented; the query $(\texttt{SCH-deliverPK}, \mathsf{pk})$ is merely forwarded by $\mathcal{F}$ as output to the dummy parties from $\mathcal{S}$.
- As in the `SCH-deliverSIG` interface of $\mathcal{F}_{\mathsf{sch}}$: the functionality $\mathcal{F}$ delivers the signature upon receipt of the query. In H_2 the interface is not yet fully implemented; the signature (σ, e) is merely forwarded by $\mathcal{F}$ from $\mathcal{S}$.
- As in the `SCH-abortkey` interface of $\mathcal{F}_{\mathsf{sch}}$: the functionality $\mathcal{F}$ forwards an abort key generation query from $\mathcal{S}$ to the set of parties $\mathcal{P}$. In H_2 the interface is fully implemented.
- As in the `SCH-abortsig` interface of $\mathcal{F}_{\mathsf{sch}}$: the functionality $\mathcal{F}$ forwards an abort signature query from $\mathcal{S}$ to the set of parties $\mathcal{Q}$. In H_2 the interface is fully implemented.

– *Modifications to $\mathcal{S}$:*

- *Record Keeping.* Tables are initialized and maintained as in Figure 20.
- *Key Delivery and Abortion Queries.* As in $\mathcal{S}_{\mathsf{sch}}$ Step 10, if $\mathcal{A}$ aborts subsession 0 of $\mathcal{F}_{\mathsf{cdkg}}$, the simulator $\mathcal{S}$ sends a query $(\texttt{SCH-abortkey}, j)$ to $\mathcal{F}_{\mathsf{sch}}$. As in $\mathcal{S}_{\mathsf{sch}}$ Step 11, upon receipt of $\mathcal{A}$ delivering the output of $\mathcal{F}_{\mathsf{cdkg}}$ for subsession 0, the simulator $\mathcal{S}$ queries $(\texttt{SCH-deliverPK})$ to $\mathcal{F}_{\mathsf{sch}}$, but includes the public key computed in the simulated protocol.
- *Signature Delivery and Abortion Queries.* As in $\mathcal{S}_{\mathsf{sch}}$ Step 19 $\mathcal{S}$ sends a query $(\texttt{SCH-abortsig}, [\cdots])$ to $\mathcal{F}_{\mathsf{sch}}$ when necessary. As in $\mathcal{S}_{\mathsf{sch}}$ Step 20, the simulator $\mathcal{S}$ issues `SCH-deliverSIG` queries to $\mathcal{F}_{\mathsf{sch}}$, but includes the signature computed in the simulated protocol.

This change is syntactic, so the view of $\mathcal{Z}$ is unchanged; H_2 is perfectly indistinguishable from H_1.

Hybrid H_3: *Functionality Samples Public Key & Generates Signatures.* The functionality samples pk, and generates the signature in subsession ssid. In H_3 functionality does not keep secrets; $\mathcal{F}$ leaks the secret key sk, and the discrete log k of $R = k \cdot G$ to $\mathcal{S}$ for every ssid.

– *Modifications to $\mathcal{F}$:*

- *Samples Keys.* As in the SCH-init interface of $\mathcal{F}_{\mathsf{sch}}$: Upon receipt of an initialization query from every $P \in \mathcal{P}$ the functionality samples the key pair $(\mathsf{sk}, \mathsf{pk} = \mathsf{sk}G)$ as in Figure 18. In H_3 the interface is not yet fully implemented; the functionality forwards $(\mathtt{SCH\text{-}pk}, \mathsf{pk}, \mathsf{sk})$ to $\mathcal{S}$.
- *Samples Signatures.* As in the SCH-sign interface of $\mathcal{F}_{\mathsf{sch}}$: Upon receipt of a signing query for subsession ssid from all $P_k \in \mathcal{Q} \setminus \mathcal{C}$ the functionality samples a nonce pair $(k, R = k \cdot G)$ and computes the signature $\sigma = k + e \cdot \mathsf{sk}$. In H_3 this interface is fully implemented.

– *Modifications to $\mathcal{S}$:*

- *Receive Key from $\mathcal{F}$.* The simulator computes the output of $\mathcal{F}_{\mathsf{cdkg}}$ for subsession 0 such that the secret key sk received from $\mathcal{F}$ is shared.
- *Receive Nonces in Subsession ssid from $\mathcal{F}$.* The simulator computes the output of $\mathcal{F}_{\mathsf{cdkg}}$ for ssid from the nonce $k := \sigma - e \cdot \mathsf{sk}$ for the signature (σ, e) received from $\mathcal{F}$.

The distribution of pk and $\{(\mathsf{ssid}, R_{\mathsf{ssid}})\}$ in H_3 is exactly the distribution of pk and $\{(\mathsf{ssid}, R_{\mathsf{ssid}})\}$ in H_2; the functionality $\mathcal{F}$ samples these values in H_3 by the same processes as the simulated $\mathcal{F}_{\mathsf{cdkg}}$ in H_2; this change is perfectly indistinguishable.

Hybrid H_4: *Reject Wrong Partial Signatures.* In this hybrid, $\mathcal{S}$ computes the expected corrupted party partial signatures $\mathtt{sigs}[\mathsf{ssid}][j] := (\sigma_i, \rho_i)$ from the emulated $\mathcal{F}_{\mathsf{cdkg}}$ as in Step 18 of $\mathcal{S}_{\mathsf{sch}}$, and disqualifies any corrupted party that sends $(\sigma_i^*, \rho_i^*) \neq \mathtt{sigs}[\mathsf{ssid}][j]$ as in Step 15a of $\mathcal{S}_{\mathsf{sch}}$.

An adversary that can distinguish H_{11} from H_{12} with non-negligible probability implies an adversary $\mathcal{B}$ against the discrete logarithm problem in $\mathcal{G}$ as follows $\mathcal{B}(H)$:

1. Sample $b \leftarrow_{\$} \{0, 1\}$.
2. Define $\mathcal{D} := \emptyset$.
3. If $b = 0$: execute H_{11} with $\mathcal{A}$, with the following changes:
   (a) On (σ_j^*, ρ_j^*) from $\mathcal{A}$ on behalf of $P_j \in \mathcal{Q} \cap \mathcal{C}$:
      – If $(\sigma_j^*, \rho_j^*) \neq \mathtt{sigs}[\mathsf{ssid}][j]$ and $\sigma_j^* \cdot G + \rho_j^* \cdot H = R_j + e \cdot \mathsf{pk}_j$: add $(\mathsf{ssid}, j, \sigma_j^*, \rho_j^*) \to \mathcal{D}$.
4. Else $(b = 1)$: execute H_{12} with $\mathcal{A}$ with the changes to the emulation of $\mathcal{F}_{\mathsf{cdkg}}$ as below:
   (a) On $(\mathtt{CDKG\text{-}newCK}, c)$: sample $\tau_c \leftarrow_{\$} \mathbb{Z}_q$, define $H_c := \tau_c \cdot H$.
   (b) On $(\mathtt{CDKG\text{-}setCK}, c)$: also record $(\mathtt{mult}, \tau_c)$.
   (c) On (σ_j^*, ρ_j^*) from $\mathcal{A}$ on behalf of $P_j \in \mathcal{Q} \cap \mathcal{C}$:
      – If $(\sigma_j^*, \rho_j^*) \neq \mathtt{sigs}[\mathsf{ssid}][j]$ and $\sigma_j^* \cdot G + \rho_j^* \cdot H = R_j + e \cdot \mathsf{pk}_j$: add $(\mathsf{ssid}, j, \sigma_j^*, \rho_j^*) \to \mathcal{D}$.
5. Await $b' \leftarrow \mathcal{A}$.
6. If $b' = 0$, $b = 0$ or $\mathcal{D}$ is empty output $\perp$.
7. Else ($b' = b = 1$ and $\mathcal{D}$):
   (a) Pick any $(\mathsf{ssid}, j, \sigma_j^*, \rho_j^*) \in \mathcal{D}$. Retreive $(\sigma_j, \rho_j) \leftarrow \mathtt{sigs}[\mathsf{ssid}][j]$. Compute

$$x := \frac{\sigma_j^* - \sigma_j}{\rho_j - \rho_j^*}$$

   (b) Output $\mathsf{log}_G(H) := \tau_c^{-1} x$.

Observe that the view of $\mathcal{A}$ is independent of b if $\mathcal{D}$ is non-empty. By assumption the advantage of $\mathcal{A}$ in distinguishing H_{12} from H_{11} is some non-negligible function $\epsilon(\lambda)$:

$$\begin{aligned} \Pr[b' = b] &= \frac{1}{2} + \epsilon(\lambda) \\ &= \Pr[b' = b \wedge \mathcal{D} \neq \emptyset] + \Pr[b' = b \wedge \mathcal{D} = \emptyset] \\ &= \frac{1}{2} \cdot \Pr[\mathcal{D} = \emptyset] + \Pr[b' = b \wedge \mathcal{D} \neq \emptyset]. \end{aligned} \tag{5}$$

Rearranging, we compute the probability that $\mathcal{A}$ is correct and $\mathcal{D} \neq \emptyset$ as:

$$\begin{aligned}\Pr[b' = b \wedge \mathcal{D} \neq \emptyset] &\overset{(5)}{=} \left(\frac{1}{2} + \epsilon(\lambda)\right) - \frac{1}{2} \cdot \underbrace{\Pr[\mathcal{D} = \emptyset]}_{=1-\Pr[\mathcal{D}\neq\emptyset]} \\ &= \frac{1}{2} \cdot \Pr[\mathcal{D} \neq \emptyset] + \epsilon(\lambda).\end{aligned} \tag{6}$$

Observe that $\mathcal{B}$ succeeds in outputting $\mathsf{log}_G(H)$ whenever $b' = b = 1$ and $\mathcal{D} \neq \emptyset$. We compute the advantage $\epsilon_{\mathsf{DL}}(\mathcal{G})$ of $\mathcal{B}$ as:

$$\begin{aligned}\mathsf{Adv}(\mathcal{B}) &:= \Pr[b' = b \wedge b = 1 \wedge \mathcal{D} \neq \emptyset] \\ &= \Pr[b' = b \wedge \mathcal{D} \neq \emptyset] - \Pr[b' = b \wedge \mathcal{D} \neq \emptyset \wedge b = 0] \\ &= \Pr[b' = b \wedge \mathcal{D} \neq \emptyset] - \underbrace{(1 - \Pr[b' \neq b \vee \mathcal{D} = \emptyset \vee b \neq 0])}_{=\Pr[b'=b\wedge\mathcal{D}\neq\emptyset\wedge b=0]} \\ &\geq \Pr[b' = b \wedge \mathcal{D} \neq \emptyset] - \underbrace{(1 - \Pr[b = 0])}_{=\Pr[b=1]} \\ &\overset{(6)}{=} \frac{1}{2} \cdot (1 - \Pr[\mathcal{D} \neq \emptyset]) + \epsilon(\lambda) \geq \epsilon(\lambda).\end{aligned} \tag{7}$$

This implies that the advantage of $\mathcal{B}$ is non-negligible as well. Therefore, under the DL assumption in $\mathcal{G}$ no such $\mathcal{A}$ that can distinguish H_4 from H_3 exists: $\mathsf{H}_4 \approx \mathsf{H}_3$.

Hybrid H_5: *Random Commitments.* In every subsession, $\mathcal{S}$ samples the commitments R_i to the secret shares of every honest $P_i \in \mathcal{Q}$ randomly.

– *Modifications to $\mathcal{S}$:*

- As in $\mathcal{S}_{\mathsf{sch}}$ Step 3, the simulator uniformly samples the exponent d_i and defines the commitments $D_i := d_i \cdot G$.
- As in $\mathcal{S}_{\mathsf{sch}}$ Step 23d and Step 23(e)vi, on an adaptive corruption $\mathcal{S}$ computes $\mathtt{rands}[i][\mathsf{ssid}] := \tau^{-1}(d_i - k_i)$ by equivocating the commitments with the trapdoor of the commitment key τ.
- As in $\mathcal{S}_{\mathsf{sch}}$ Step 15c, the simulator computes the second component of each honest party signature ρ_i such that the verification equation $\sigma_i \cdot G + \rho_i \cdot H = D_i + e \cdot \mathsf{pk}_i$ is satisfied.

The distribution of the commitments output by $\mathcal{F}_{\mathsf{cdkg}}$ is uniform in $\mathbb{G}$ in both H_5 and H_4, and there is a single ρ_i possible for fixed $(\sigma_i, e, \mathsf{pk}_i, D_i)$; the distribution of ρ_i is equal in H_5 and H_4.

Hybrid H_6: *Interpolated Honest Party Signatures.* In every subsession, $\mathcal{S}$ computes a random threshold sharing of the signature from the signature $\sigma = k + e \cdot \mathsf{sk}$ and the corrupt party signatures.

– *Modifications to $\mathcal{S}$:* As in $\mathcal{S}_{\mathsf{sch}}$ Step 15 the simulator computes the partial signatures $\sigma_j = r_j + e\mathsf{sk}_j$ of the corrupt parties and interpolates the honest party signatures from these points and the signature received from $\mathcal{F}$.

In subsession ssid the distribution of the partial signatures σ_i in H_6 is exactly the distribution of the partial signatures σ_i in H_5; in H_5 the sharing of k was random of degree $t-1$, and in H_6 the sharing of σ is random of degree $t-1$ so this change is perfectly indistinguishable.

Hybrid H_7: *Functionality Does not Leak Secret Key.* In this hybrid, $\mathcal{F}$ no longer leaks sk to $\mathcal{S}$.

– *Modifications to $\mathcal{F}$:* The functionality sends $(\mathtt{SCH\text{-}pk}, \mathsf{pk})$, eliminating leakage of sk. In H_8, the interface is fully implemented and therefore the functionality $\mathcal{F} = \mathcal{F}_{\mathsf{sch}}$ is fully implemented.

As of H_6, the simulator only uses knowledge of sk upon the final corruption. Therefore, the output of the simulator in H_6 is the same as H_7 and this change is perfectly indistinguishable.

Hybrid H_8: *Ideal World.* The ideal execution, where the environment $\mathcal{Z}$ invokes $\mathcal{F}_{\mathsf{sch}}$ of Figure 18 with $\mathcal{S}_{\mathsf{sch}}$. This change is syntactic; as of H_7 both $\mathcal{S} = \mathcal{S}_{\mathsf{sch}}$ and $\mathcal{F} = \mathcal{F}_{\mathsf{sch}}$ are fully implemented. For the random variable $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{sch}},\mathcal{S}_{\mathsf{sch}},\mathcal{Z}}(z)$ which is the output of an interaction with $\mathcal{F}_{\mathsf{sch}}$, the $\mathcal{Z}$ and $\mathcal{S}_{\mathsf{sch}}$ on input z, this hybrid corresponds to the ensemble:

$$\{\mathsf{EXEC}_{\mathcal{F}_{\mathsf{sch}},\mathcal{S}_{\mathsf{sch}},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}.$$

Combining all of the steps above, we have that $\mathsf{H}_8 \approx \mathsf{H}_0$ and $\Pi^{\mathsf{ia}}_{\mathsf{sch}}$ is secure under the DL assumption in $\mathcal{G}$.

Output Guarantees. Guaranteed output delivery holds assuming an honest majority ($t < n/2$) assuming that $\mathcal{F}_{\mathsf{bc}}$ guarantees eventual delivery of all honest party messages and $\mathcal{F}_{\mathsf{cdkg}}$ has guaranteed output delivery. For key generation, if $n - |\mathcal{C}|$ activate $\mathcal{F}_{\mathsf{sch}}$ then $\mathcal{S}_{\mathsf{sch}}$ emulates the generation and delivery of pk by $\mathcal{F}_{\mathsf{cdkg}}$, which has guaranteed output delivery by assumption. For signing, guarnteed delivery of nonces follows by the same argument. Upon delivery of the nonce in subsession ssid the simulator simulates the honest party partial signatures. As $\mathcal{F}_{\mathsf{bc}}$ eventually delivers all messages by assumption, when $\mathcal{Q} = \mathcal{P}$ at least $n - |\mathcal{C}| \geq n - t > n - \frac{n}{2} > t$ partial signatures are delivered and $\mathcal{S}_{\mathsf{sch}}$ queries `SCH-deliverSIG` to deliver the signature.

Identifiable abort holds if $\mathcal{F}_{\mathsf{cdkg}}$ satisfies identifiable abort and $\mathcal{F}_{\mathsf{bc}}$ eventually delivers all broadcast messages, assuming that $t < n$. For key generation, if $n - |\mathcal{C}|$ honest (dummy) parties activate $\mathcal{F}_{\mathsf{sch}}$ then $\mathcal{S}_{\mathsf{sch}}$ simulates the $n - |\mathcal{C}|$ honest queries to $\mathcal{F}_{\mathsf{cdkg}}$ and either a key is output or a corrupted party is identified, by assumption. Therefore, key generation always completes with a query of either `SCH-abortkey` or `SCH-deliverPK`. For signing, nonce generation follows by the same argument; $\mathcal{F}_{\mathsf{cdkg}}$ always delivers a nonce or results in a query of `SCH-abortsig` by assumption. Upon delivery of the nonce in subsession ssid the simulator simulates the honest party partial signatures. As $\mathcal{F}_{\mathsf{bc}}$ eventually delivers all broadcast messages and parties are disqualified if they send an empty partial signature, either $|\mathcal{V}| < t$ or $|\mathcal{V}| \geq t$ and a signature can be reconstructed; the simulator always queries `SCH-abortsig` with a disqualified party or a signature can be computed and $\mathcal{S}_{\mathsf{sch}}$ queries `SCH-deliverSIG`.

□

References

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. "Simplified Threshold RSA with Adaptive and Proactive Security". In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. St. Petersburg, Russia: Springer Berlin Heidelberg, Germany, May 2006, pp. 593–611. DOI: 10.1007/11761679_35.

[AF04] Masayuki Abe and Serge Fehr. "Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography". In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 2004, pp. 317–334. DOI: 10.1007/978-3-540-28628-8_20.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures". In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 2004, pp. 41–55. DOI: 10.1007/978-3-540-28628-8_3.

[BDLR25a] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. "Adaptively Secure Three-Round Threshold Schnorr Signatures from DDH". In: *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VI*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Vol. 16005. Lecture Notes in Computer Science. Springer, 2025, pp. 390–422. DOI: 10.1007/978-3-032-01887-8_13. URL: https://doi.org/10.1007/978-3-032-01887-8%5C_13.

[BDLR25b] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. "Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security". In: *Advances in Cryptology – EUROCRYPT 2025, Part II*. Ed. by Serge Fehr and Pierre-Alain Fouque. Vol. 15602. Lecture Notes in Computer Science. Madrid, Spain: Springer, Cham, Switzerland, May 2025, pp. 304–334. DOI: 10.1007/978-3-031-91124-8_11.

[BK25] Renas Bacho and Alireza Kavousi. "SoK: Dlog-Based Distributed Key Generation". In: *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*. Ed. by Marina Blanton, William Enck, and Cristina Nita-Rotaru. IEEE, 2025, pp. 614–632. DOI: 10.1109/SP61157.2025.00127. URL: https://doi.org/10.1109/SP61157.2025.00127.

[BL22] Renas Bacho and Julian Loss. "On the Adaptive Security of the Threshold BLS Signature Scheme". In: *ACM CCS 2022: 29th Conference on Computer and Communications Security*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. Los Angeles, CA, USA: ACM Press, Nov. 2022, pp. 193–207. DOI: 10.1145/3548606.3560656.

[BLSW24] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. "HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures". In: *Advances in Cryptology – ASIACRYPT 2024, Part III*. Ed. by Kai-Min Chung and Yu Sasaki. Vol. 15486. Lecture Notes in Computer Science. Kolkata, India: Springer, Singapore, Singapore, Dec. 2024, pp. 104–140. DOI: 10.1007/978-981-96-0891-1_4.

[BLTWZ24] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. "Twinkle: Threshold Signatures from DDH with Full Adaptive Security". In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*. Ed. by Marc Joye and Gregor Leander. Vol. 14651. Lecture Notes in Computer Science. Springer, 2024, pp. 429–459. DOI: 10.1007/978-3-031-58716-0_15. URL: https://doi.org/10.1007/978-3-031-58716-0%5C_15.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme". In: *Journal of Cryptology* 16.3 (June 2003), pp. 185–215. DOI: 10.1007/s00145-002-0120-1.

[Bol03] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer Berlin Heidelberg, Germany, Jan. 2003, pp. 31–46. DOI: 10.1007/3-540-36288-6_3.

[BP25] Luís T. A. N. Brandão and Rene Peralta. *Fully-Adaptive Two-Round Threshold Schnorr Signatures from DDH*. NIST Computer Security Resource Center. 2025. URL: https://csrc.nist.gov/pubs/ir/8214/c/2pd.

[Bra+24] Lennart Braun, Guilhem Castagnos, Ivan Damgård, Fabien Laguillaumie, Kelsey Melissaris, Claudio Orlandi, and Ida Tucker. "An Improved Threshold Homomorphic Cryptosystem Based on Class Groups". In: *SCN 24: 14th International Conference on Security in Communication Networks, Part II*. Ed. by Clemente Galdi and Duong Hieu Phan. Vol. 14974. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Cham, Switzerland, Sept. 2024, pp. 24–46. DOI: 10.1007/978-3-031-71073-5_2.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd Annual Symposium on Foundations of Computer Science*. Las Vegas, NV, USA: IEEE Computer Society Press, Oct. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[CGGMP20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. Virtual Event, USA: ACM Press, Nov. 2020, pp. 1769–1787. DOI: 10.1145/3372297.3423367.

[CGJKR99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Adaptive Security for Threshold Cryptosystems". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 1999, pp. 98–115. DOI: 10.1007/3-540-48405-1_7.

[Che25] Yanbo Chen. "Round-Efficient Adaptively Secure Threshold Signatures with Rewinding". In: *IACR Commun. Cryptol.* 2.2 (2025), p. 16. DOI: 10.62056/AH893Z10K. URL: https://doi.org/10.62056/ah893z10k.

[CS25] Elizabeth C. Crites and Alistair Stewart. "A Plausible Attack on the Adaptive Security of Threshold Schnorr Signatures". In: *Advances in Cryptology – CRYPTO 2025, Part VI*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Vol. 16005. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, Aug. 2025, pp. 457–479. DOI: 10.1007/978-3-032-01887-8_15.

[Des88] Yvo Desmedt. "Society and Group Oriented Cryptography: A New Concept". In: *Advances in Cryptology – CRYPTO'87*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 1988, pp. 120–127. DOI: 10.1007/3-540-48184-2_8.

[DF90] Yvo Desmedt and Yair Frankel. "Threshold Cryptosystems". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, New York, USA, Aug. 1990, pp. 307–315. DOI: 10.1007/0-387-34805-0_28.

[DKLs24] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. "Threshold ECDSA in Three Rounds". In: *2024 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2024, pp. 3053–3071. DOI: 10.1109/SP54263.2024.00178.

[DR24] Sourav Das and Ling Ren. "Adaptively Secure BLS Threshold Signatures from DDH and co-CDH". In: *Advances in Cryptology – CRYPTO 2024, Part VII*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14926. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, Aug. 2024, pp. 251–284. DOI: 10.1007/978-3-031-68394-7_9.

[Fel87] Paul Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *28th Annual Symposium on Foundations of Computer Science*. Los Angeles, CA, USA: IEEE Computer Society Press, Oct. 1987, pp. 427–437. DOI: 10.1109/SFCS.1987.4.

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 2005, pp. 152–168. DOI: 10.1007/11535218_10.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. "On the Non-malleability of the Fiat-Shamir Transform". In: *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*. Ed. by Steven D. Galbraith and Mridul Nandi. Vol. 7668. Lecture Notes in Computer Science. Kolkata, India: Springer Berlin Heidelberg, Germany, Dec. 2012, pp. 60–79. DOI: 10.1007/978-3-642-34931-7_5.

[FMT24] Hanwen Feng, Tiancheng Mai, and Qiang Tang. "Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing". In: *ACM CCS 2024: 31st Conference on Computer and Communications Security*. Ed. by Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie. Salt Lake City, UT, USA: ACM Press, Oct. 2024, pp. 2636–2650. DOI: 10.1145/3658644.3690253.

[FMY99a] Yair Frankel, Philip D. MacKenzie, and Moti Yung. "Adaptively-Secure Distributed Public-Key Systems". In: *Algorithms - ESA '99, 7th Annual European Symposium, Prague, Czech Republic, July 16-18, 1999, Proceedings*. Ed. by Jaroslav Nesetril. Vol. 1643. Lecture Notes in Computer Science. Springer, 1999, pp. 4–27. DOI: 10.1007/3-540-48481-7_2. URL: https://doi.org/10.1007/3-540-48481-7%5C_2.

[FMY99b] Yair Frankel, Philip D. MacKenzie, and Moti Yung. "Adaptively-Secure Optimal-Resilience Proactive RSA". In: *Advances in Cryptology – ASIACRYPT'99*. Ed. by Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing. Vol. 1716. Lecture Notes in Computer Science. Singapore: Springer Berlin Heidelberg, Germany, Nov. 1999, pp. 180–194. DOI: 10.1007/978-3-540-48000-6_15.

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology – CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[GCRS25] Paul Gerhart, Davide Li Calsi, Luigi Russo, and Dominique Schröder. *Fully-Adaptive Two-Round Threshold Schnorr Signatures from DDH*. Cryptology ePrint Archive, Paper 2025/1478. 2025. URL: https://eprint.iacr.org/2025/1478.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Secure Applications of Pedersen's Distributed Key Generation Protocol". In: *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*. Ed. by Marc Joye. Vol. 2612. Lecture Notes in Computer Science. Springer, 2003, pp. 373–390. DOI: 10.1007/3-540-36563-X_26. URL: https://doi.org/10.1007/3-540-36563-X%5C_26.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Journal of Cryptology* 20.1 (Jan. 2007), pp. 51–83. DOI: 10.1007/s00145-006-0347-3.

[GJKR96] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Robust Threshold DSS Signatures". In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Saragossa, Spain: Springer Berlin Heidelberg, Germany, May 1996, pp. 354–371. DOI: 10.1007/3-540-68339-9_31.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer Berlin Heidelberg, Germany, May 1999, pp. 295–310. DOI: 10.1007/3-540-48910-X_21.

[Gur+21] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. "Aggregatable Distributed Key Generation". In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Cham, Switzerland, Oct. 2021, pp. 147–176. DOI: 10.1007/978-3-030-77870-5_6.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Zero-knowledge from secure multiparty computation". In: *39th Annual ACM Symposium on Theory of Computing*. Ed. by David S. Johnson and Uriel Feige. San Diego, CA, USA: ACM Press, June 2007, pp. 21–30. DOI: 10.1145/1250790.1250794.

[JL00] Stanislaw Jarecki and Anna Lysyanskaya. "Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures". In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer Berlin Heidelberg, Germany, May 2000, pp. 221–242. DOI: 10.1007/3-540-45539-6_16.

[Kat24] Jonathan Katz. "Round-Optimal, Fully Secure Distributed Key Generation". In: *Advances in Cryptology – CRYPTO 2024, Part VII*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14926. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, Aug. 2024, pp. 285–316. DOI: 10.1007/978-3-031-68394-7_10.

[KMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. "Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures". In: *ACM CCS 2020: 27th Conference on Computer and Communications Secu-*

rity. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. Virtual Event, USA: ACM Press, Nov. 2020, pp. 1751–1767. DOI: 10.1145/3372297.3423364.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. "Universally Composable Synchronous Computation". In: *TCC 2013: 10th Theory of Cryptography Conference*. Ed. by Amit Sahai. Vol. 7785. Lecture Notes in Computer Science. Tokyo, Japan: Springer Berlin Heidelberg, Germany, Mar. 2013, pp. 477–498. DOI: 10.1007/978-3-642-36594-2_27.

[Ks22] Yashvanth Kondi and abhi shelat. "Improved Straight-Line Extraction in the Random Oracle Model with Applications to Signature Aggregation". In: *Advances in Cryptology – ASIACRYPT 2022, Part II*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Cham, Switzerland, Dec. 2022, pp. 279–309. DOI: 10.1007/978-3-031-22966-4_10.

[LP01] Anna Lysyanskaya and Chris Peikert. "Adaptive Security in the Threshold Setting: From Cryptosystems to Signature Schemes". In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Gold Coast, Australia: Springer Berlin Heidelberg, Germany, Dec. 2001, pp. 331–350. DOI: 10.1007/3-540-45682-1_20.

[LR22] Anna Lysyanskaya and Leah Namisa Rosenbloom. *Efficient and Universally Composable Non-Interactive Zero-Knowledge Proofs of Knowledge with Security Against Adaptive Corruptions*. Cryptology ePrint Archive, Report 2022/1484. 2022. URL: https://eprint.iacr.org/2022/1484.

[Mak22] Nikolaos Makriyannis. "On the Classic Protocol for MPC Schnorr Signatures". In: *IACR Cryptol. ePrint Arch.* (2022), p. 1332. URL: https://eprint.iacr.org/2022/1332.

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. "MuSig2: Simple Two-Round Schnorr Multi-signatures". In: *Advances in Cryptology – CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. Lecture Notes in Computer Science. Virtual Event: Springer, Cham, Switzerland, Aug. 2021, pp. 189–221. DOI: 10.1007/978-3-030-84242-0_8.

[Ped91] Torben P. Pedersen. "A Threshold Cryptosystem without a Trusted Party (Extended Abstract) (Rump Session)". In: *Advances in Cryptology – EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. Lecture Notes in Computer Science. Brighton, UK: Springer Berlin Heidelberg, Germany, Apr. 1991, pp. 522–526. DOI: 10.1007/3-540-46416-6_47.

[Ped92] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer Berlin Heidelberg, Germany, Aug. 1992, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (Jan. 1991), pp. 161–174. DOI: 10.1007/BF00196725.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

[SS01] Douglas R. Stinson and Reto Strobl. "Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates". In: *Information Security and Privacy, 6th Australasian Conference, ACISP 2001, Sydney, Australia, July 11-13, 2001, Proceedings*. Ed. by Vijay Varadharajan and Yi Mu. Vol. 2119. Lecture Notes in Computer Science. Springer, 2001, pp. 417–434. DOI: 10.1007/3-540-47719-5_33. URL: https://doi.org/10.1007/3-540-47719-5%5C_33.

[Syt+17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. "Scalable Bias-Resistant Distributed Randomness". In: *2017 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, May 2017, pp. 444–460. DOI: 10.1109/SP.2017.45.