A Formal Treatment of Distributed Key Generation, and New Constructions

Chelsea Komlo, Ian Goldberg, Douglas Stebila

Abstract. In this work, we present a novel generic construction for a Distributed Key Generation (DKG) scheme. Our generic construction relies on three modular cryptographic building blocks. The first is an aggregatable Verifiable Secret Sharing (AgVSS) scheme, the second is a Non-Interactive Key Exchange (NIKE) scheme, and the third is a secure hash function. We give formal definitions for the AgVSS and NIKE schemes, as well as concrete constructions. The utility of this generic construction is *flexibility*; i.e., any aggregatable VSS and NIKE scheme can be employed, and the construction will remain secure.

To prove the security of our generic construction, we introduce formalized game-based notions of security for DKGs, building upon existing notions in the literature. However, these prior security notions either were presented informally, omitted important requirements, or assumed certain algebraic structure of the underlying scheme. Our security notions make no such assumption of underlying algebraic structure, and explicitly consider details such as participant consistency, communication patterns, and key validity. Further, our security notions imply simulatability with respect to a target key generation scheme without rewinding. Hence, any construction that is proven secure using our security notions additionally achieves UC security.

We then present STORM, a concrete instantiation of our generic construction that is secure in the discrete logarithm setting in the random oracle model. STORM is more efficient than related DKG schemes in the literature. Because of its simple design and composability, it is a practical choice for real-world settings and standardization efforts.

Table of Contents

1 Introduction

Distributed Key Generation. Distributed Key Generation (DKG) schemes underlie many multi-party cryptographic primitives in use today, such as threshold signature schemes [5,34,37] and distributed randomness beacons [36]. A DKG allows a set of n participants to cooperate to generate a keypair, such that the public key represents the entire set of participants, and each participant holds a share of the corresponding secret key. Importantly, DKGs ensure that the public key is agreed upon by all (honest) parties at the end of the protocol, but no single party knows the corresponding secret key. However, DKGs ensure that this secret key can be recovered given a threshold t number of cooperating parties, where $t \leq n$.

Despite widespread interest in DKGs, existing constructions fall short in either their complexity or composability. Most glaringly is Pedersen-DKG [40]. While Pedersen-DKG is appealing because of its simplicity and efficiency, its security to date has only been proven in the context of specific applications [5, 27, 30], as it cannot be proven secure in a standalone fashion [28]. More specifically, it is possible that the use of Pedersen-DKG may be secure in some settings, but not in others. As such, the risk exists that Pedersen-DKG might be misused in contexts for which it cannot been proven secure. While many alternative DKG constructions have been presented in the literature [1, 3, 9, 21, 28–30, 34, 37, 39], these constructions either are not proven secure with respect to a target key generation scheme, incur additional protocol complexity, or require pairings.

Conversely, growing interest in using threshold schemes in practice points towards the need for DKGs that are friendly to standardization efforts and securely composable across a range of use cases. For example, the FROST Schnorr threshold signature scheme [34], whose security was demonstrated at CRYPTO '22 [5], and for which an IETF draft exists [19], is written in a way to be agnostic to any particular key generation scheme. Ideally, a candidate DKG to be standardized should be applicable not only to threshold signature schemes like FROST that are compatible with single-party EdDSA verification [10, 11], but also to other applications that require secret key material to be distributed among a set of trusted parties [22].

Proving the Security of DKGs. However, challenges arise when proving the security of a DKG in a standalone manner; i.e., that the DKG is secure in *any* setting in place of its target (single-party) key generation scheme. Although existing simulatability-based notions for DKGs exist [3, 6, 9, 28, 30, 37], these notions prove to be either incomplete or insufficient. For example, these simulatability-based notions assume characteristics of the underlying algebraic structure of the scheme, by expecting a unique correspondence between secret and public keys. While such an assumption holds for schemes that reduce to discrete logarithm assumptions, this assumption is not universally held. For example, in lattice-based cryptosystems, public keys have a many-to-one relation with secret keys.

Further, many simulatability-based notions assume important details such as the consistency of honest participants' state or that adversarial players do not abort. However, such assumptions are *not* guaranteed in a distributed, multi-party setting. Additionally, critical security issues can arise when adversaries are allowed to abort at key times in the protocol. For example, allowing participants to abort the protocol during the last round can be employed as a vector for a key-bias attack [28]. Under the hood, many definitions require the protocol to be robust; i.e., that the protocol will always complete successfully in spite of misbehaving parties. However, doing so increases the complexity for the protocol by requiring additional complaint and voting rounds, which are difficult to implement correctly in practice.

1.1 Our Contributions

A Composable and Generic DKG. In this work, we present a novel generic DKG construction. Our construction relies on three simple cryptographic building blocks: a secure hash function, an *Aggregatable Verifiable Secret Sharing* (AgVSS) scheme, and a *Non-Interactive Key Exchange* (NIKE) scheme. The ability to aggregate Feldman's VSS [23] has been extensively employed in prior DKG constructions [28, 30, 34, 37, 40]; we simply formalize the notion of a VSS that is aggregatable, and present new security notions to reflect this property. We additionally provide formal definitions and notions of security for a NIKE, as well as concrete AgVSS and NIKE constructions.

	GJKR [28]	PedPop [5,34]	STORM
Composable?	Yes	No	Yes
Rounds	3	2	3
Bandwidth Efficiency	$2nt + 3n$	$nt + 3n$	$nt + 5n$
Computational Efficiency	$2nt + 2n + t$	$nt + 2n$	$nt + 2n$
Security Model	DLP	KoE/AGM	CDH
Assumed Threshold	$n \geq 2t - 1$	$n \geq t$	$n \geq 2t - 1$

Table 1. Comparison between STORM and related synchronous constructions that are secure in the random oracle model, do not require pairings, and have formal proofs of security. Composable denotes if the construction has a standalone proof of security and can be used in any context that its target (single-party) key generation algorithm is used. The number of estimated rounds do not consider sub-protocols to identify and exclude cheating parties. Bandwidth efficiency denotes the number of group and field elements sent and received by each participant throughout the protocol. Computational efficiency denotes the number of exponentiations required (i.e., scalar multiplications in an elliptic curve group). n is the total number of participants in the protocol, t is the threshold number required to recover the secret key, and the number of corrupted parties is assumed to be *at most* $t - 1$. k is security parameters for the respective scheme [21]. DLP is the Discrete Logarithm Problem; KoE is the Knowledge of Exponent model; AGM is the Algebraic Group Model; CDH is the Computational Diffie-Hellman Problem.

Why a Generic Construction? The goal for presenting a generic DKG as opposed to a monolithic construction is to enable flexibility for implementations and future standardization efforts. For example, while we define concrete AgVSS and NIKE constructions, future implementations may instead wish to employ a standardized NIKE [4] instead. Further, as NIST is currently in the process of standardizing threshold primitives [11], it may be worthwhile to leverage future VSS standards, which may possibly diverge from the concrete VSS defined in this work. Or, implementations may wish to define a construction that employs publicly verifiable randomness, or derives secret values deterministically, such as from a seed phrase. The utility of a generic DKG is that so long as the chosen building blocks are secure, any instantiation of our generic DKG using these building blocks will remain secure.

A Concrete Instantiation: STORM. We present a concrete instantiation of our generic construction that we call STORM (Synchronous, disTributed, and Optimized geneRation of key Material). STORM employs the concrete AgVSS and NIKE constructions defined in this paper. The security of STORM reduces to the Computational Diffie-Hellman (CDH) problem in the random oracle model. STORM is securely composable with existing threshold cryptosystems such as threshold Schnorr signatures [19,34] and similar discrete-log based systems. We compare in Table 1 the practicality and security of STORM compared to related constructions in the literature that require only standard (i.e., non-pairing based) groups.

New Notions of Security for DKGs. Our goal is to prove the security of our generic construction in such a way that (i) demonstrates the DKG can be securely used in any setting where its target key generation scheme is used, (ii) captures implicit notions such as handling adversarial aborts and honest participant consistency, and (iii) assumes nothing about the algebraic structure of the scheme. Towards this end, we introduce formal game-based security notions for DKGs that generalize and extend prior notions in the literature. Our games make explicit important details such as communication patterns between honest and adversarial players, where prior notions have omitted some details. For example, our games explicitly model the adversary as *rushing*, in that it can always query for honest player's contributions before producing its own. We then prove the security of our generic DKG with respect to these notions.

For a DKG to be secure, we require that the DKG be strongly or weakly robust, zero-knowledge, and indistinguishable. Strong versus weak robustness captures the distinction between a protocol that always

succeeds assuming some threshold of honest participants (strong robustness), and one which allows for non-fatal aborts, so long as all participants end in a consistent state (weak robustness). This distinction allows us to define a much simpler DKG construction. Similar to prior simulation-based notions, zero-knowledge requires that the DKG is simulatable with respect to a public key. Indistinguishability requires that the output of a DKG be indistinguishable from its target (single-party) key generation scheme.

By fulfilling the notion of indistinguishability, we can guarantee that the DKG is composable in any setting where the single-party key generation scheme could be used. Prior simulation-based notions of security for DKGs assumed that zero-knowledge implied indistinguishability, but such an assumption holds only under cryptosystems where secret and public keys have a unique correspondence. This is because simulatability-based experiments require the environment to simulate its participation with respect to a public challenge, and hence assert conditions *only* on public key material.

These notions could perhaps be captured by an alternative model such as Universal Composability [12]. However, we opt for a game-based approach, so that we can explicitly and independently define each notion. However, because our notions of security capture *perfect simulatability* with respect to a target key generation algorithm without rewinding, we can guarantee that DKGs that achieve our notions of security in fact achieve UC security [35].

What We Do Not Do. Our definitions and constructions assume a strictly synchronous setting; i.e., that every participant in the protocol terminates a round before beginning the next. While asynchronous DKGs exist [2, 21, 31, 33, 39], these constructions require the tradeoff of increased complexity. Instead, we target the setting where key generation is performed infrequently or in a more controlled environment, such as generating a long-lived keypair among a set of signing parties.

Additionally, while some prior DKGs build upon a publicly verifiable secret sharing scheme (PVSS) to reduce network round complexity [9, 29], we explicitly do not, due to the tradeoff in increased protocol complexity. For similar reasons, we do not consider the large-scale setting [15].

Open Questions. Our generic construction assumes a one-to-one correspondence between public and private keys. While STORM is secure assuming the hardness of the Computational Diffie-Hellman (CDH) problem, it is possible our generic construction can be instantiated by quantum-secure primitives that define a homomorphism between the secret and public domains. For example, performing rounds in a sequential manner may allow our generic construction to be instantiated by CSIDH-based primitives [17, 18]. However, because of the difficulty of instantiating a NIKE using lattice-based primitives, it is an open question as to how our generic construction can be extended to lattice-based primitives.

Summary of Our Contributions. In this work, we present the following contributions.

- We introduce the notion of an aggregated Verifiable Secret Sharing (AgVSS) scheme, building upon its implicit use in prior literature. We give concrete AgVSS and Non-Interactive Key Exchange (NIKE) constructions, and prove their security.
- We present formal game-based definitions for the security of a DKG, that makes explicit details such as communication patterns between honest and corrupted players, expectations of participant consistency and requirements of key validity. Further, our separation of the notion of robustness into a strong and weak variant allows for a simpler and more efficient DKG construction.
- We give a generic DKG construction, and prove its security using our new notions. Our generic construction requires a secure hash function, an AgVSS, and a NIKE.
- We introduce STORM, a concrete instantiation of our generic construction. The security of STORM reduces to the Computational Diffie-Hellman (CDH) problem in the random oracle model.

1.2 Related Work

Synchronous Constructions. Pedersen-DKG [41] is a two-round protocol among n parties, such that each party plays the role of the dealer in a Feldman Verifiable Secret Sharing (VSS) protocol [40]. Gennaro, Jarecki,

Krawczyk, and Rabin [28] describe an attack against Pedersen-DKG, which we refer to as the Key-Influence attack, that considers a *rushing adversary*, who is allowed to speak. In this attack, an adversary can influence the distribution of key material output by the protocol by adaptively choosing their contribution. While the authors later demonstrate unforgeability of a Schnorr threshold signature scheme with Pedersen-DKG employed for key generation [27], this proof of security does *not* extend to other applications. Gurkan, Jovanovic, Maller, Meiklejohn, Stern, and Tomescu [30] show that Pedersen-DKG can be used securely in any setting where the cryptographic scheme is "rekeyable". In other words, if the scheme produces some cryptographic output o_1 (such as a ciphertext or signature) that is valid with respect to a secret key sk_1, it is possible to obtain a second value $o_2 \leftarrow f_1(o_1, \mathsf{sk}', \alpha)$, such that o_2 is valid with respect to a related secret key $\mathsf{sk}_2 \leftarrow f_2(\mathsf{sk}_1, \mathsf{sk}', \alpha)$, where α is a side input and f_1, f_2 are efficient update functions. While some constructions such as BLS signatures [8] do support rekeyability, others such as Schnorr signatures do not.

Gennaro et al. [28] present a three-round construction that is secure against the Key-Influence attack. While their description of their construction implies that a round to broadcast failure messages is optional, their construction is insecure without this round. For this reason, our construction includes a mandatory "complaint or no complaint" round (see KeyGen_3 in Figure 11). Our construction employs similar techniques as Gennaro et al. [28], but offers improved performance.

Komlo and Goldberg [34] introduce a variant of Pedersen-DKG that we refer to as PedPop, where the dealer additionally sends a Schnorr proof of possession to prevent rogue-key attacks. Bellare, Crites, Komlo, Maller, Tessaro, and Zhu [5] present an alternative proof of security for PedPop, in the context of the FROST threshold signature, but this proof does not extend to the use of PedPop in other settings.

Lindell [37] introduced a DKG that builds upon Schnorr proofs of knowledge and online-extractable zero-knowledge proofs of knowledge. However, the protocol incurs computational costs due to the requirement of online extractors [24]. Further, the security of the scheme is only generally discussed without reference to a concrete notion of security.

Many efficient pairing-based DKGs are described in the literature [1, 29, 30]. Further, concrete DKGs have been presented in the ECDSA and RSA settings [13, 20, 25, 38]. This work focuses presenting a generic construction that can be instantiated without pairings, and describes STORM, a concrete construction that can be employed in the context of EdDSA. We leave extending our generic construction to the ECDSA or RSA setting for future work.

Definitions of DKG Security. Gurkan et al. [30] present a definition of DKG security, in which the DKG is secure if its use within some larger protocol does not weaken the security of the overall scheme from the centralized (single-party) setting. We build upon this notion by defining the security of a DKG with respect to a target key generation scheme. Bacho and Loss [3] introduce the security notions of consistency and oracle-aided algebraic simultatability for a DKG. Their notion of consistency is encompassed by our notions of strong and weak robustness. Because their notion of oracle-aided simultability assumes a discrete logarithm oracle, it is not applicable to schemes without such algebraic structure. Boneh and Shoup [9] present a generic definition for the security of a distributed key generation protocol using a simulation-based approach, with respect to a target trusted secret sharing scheme. However, their definition implicitly assumes certain algebraic structure of the underlying scheme; whereas our notions makes no such assumptions.

2 Preliminaries

Notation. Let $\lambda \in \mathbb{Z}$ represent the security parameter in unary representation. We denote the assignment of an element y to the value x as $y \leftarrow x$, and sampling an element from some set S uniformly at random as $x \xleftarrow{\$} S$. For a randomized algorithm A, we write $x \xleftarrow{\$} A()$ to indicate the random variable x that is output from the execution of A. Let $\mathbb{G}$ be a cyclic group of prime order q, and $\mathbb{Z}_q$ be the field of integers modulo q. Let g be a generator of $\mathbb{G}$. Let $\mathbb{F}$ denote a field. We use $[n]$ to represent the set $\{1, \ldots, n\}$.

Polynomial Interpolation. A polynomial of degree $t-1$ over a field $\mathbb{F}$ can be interpolated by t (or more) points. Let η be the list of t distinct indices $\eta \subset [n]$ corresponding to the x-coordinates $x_i \in \mathbb{F}, i \in \eta$. Then,

$L_i(x)$ is the Lagrange polynomial defined by η, of the form

$$L_i(x) = \prod_{j \in \eta; j \neq i} \frac{x - x_j}{x_i - x_j}$$

where $L_i(x) = \sum_{k=0}^{t-1} L_{k,i} x^k$ has value 1 where $x = x_i$ and 0 otherwise. Given a set of t points $(x_i, f(x_i))$, any point $f(x_\ell)$ on the degree $t-1$ polynomial f can be determined by Lagrange interpolation:

$$f(x_\ell) = \sum_{j \in \eta} f(x_j) \cdot L_j(x_\ell) \ .$$

The polynomial f can also be described as a Vandermonde matrix:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}$$

Given the set of t points $(x_i, f(x_i))$, each coefficient of the polynomial f can be solved for via

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{t-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \dots & x_t^{t-1} \end{pmatrix}^{-1} \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_t) \end{pmatrix} = \begin{pmatrix} L_{0,1} & L_{0,2} & \dots & L_{0,t} \\ L_{1,1} & L_{1,2} & \dots & L_{1,t} \\ \vdots & \vdots & \ddots & \vdots \\ L_{t-1,1} & L_{t-1,2} & \dots & L_{t-1,t} \end{pmatrix} \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_t) \end{pmatrix}$$

Where $L_{k,i}$ is the k^{th} coefficient of the Lagrange polynomial $L_i(x)$.

2.1 Threshold Schemes

A (t, n) threshold scheme is a multi-party protocol for a set of n parties. The scheme splits a secret s into n shares, which are distributed one to each of the n participants. To recover s, at least t participants are required to cooperate by pooling their shares. A common example of a threshold scheme is Shamir secret sharing [42],

Security Assumptions. The security of threshold schemes depend critically on the assumed number of corrupted parties with respect to the threshold t, as follows:

- *Honest Minority.* The adversary is assumed to control up to $t-1$ parties, and at minimum one honest party is assumed.
- *Honest Majority.* Here, the adversary is still allowed to control up to $t-1$ parties. However, at minimum, at least t honest parties are assumed.

Remark 1 (Adaptive Security). A threshold scheme can be *statically* secure or *adaptively* secure. Static security assumes the set of parties that an adversary corrupts is fixed, whereas adaptive security allows the adversary to corrupt any $t-1$ parties. In this work, we model the adversary as having the power to choose n, t, the set of honest parties honest, and the set of corrupted parties $\mathsf{corrupt}$. This modeling can easily be translated to the adaptive setting, where the adversary is given a corruption oracle instead of receiving state for the set of players $\mathsf{corrupt}$ at the beginning of the game. Consequently, our security definitions model the "single inconsistent party" notion of adaptive security [14], where the environment can handle adaptive

$\mathsf{Exp}^{\mathrm{rec}}_{\mathsf{NK},\mathcal{A}}(\lambda)$

$(\mathsf{sk}_1, \mathsf{pk}_1) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda)$

$(\mathsf{sk}_2, \mathsf{pk}_2) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda)$

$\psi \leftarrow \mathsf{NK.SharedKey}(\mathsf{sk}_1, \mathsf{pk}_2)$

$\psi' \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_1, \mathsf{pk}_2)$

return 1 **if** $\psi' = \psi$

return 0

Fig. 1. Unrecoverability experiment defining the advantage of an adversary $\mathcal{A}$ against a non-interactive key exchange (NIKE) NK.

corruptions by the adversary, with the exception of a single "inconsistent" honest player. Hence, our security notions and proofs imply adaptive security, but with exponential tightness loss. For small choices of n and t, this tightness loss may be acceptable. Techniques to achieve tight proofs of adaptive security in stronger security models have recently been demonstrated in concurrent work [1, 3]; we expect these techniques to similarly apply to our constructions.

2.2 Non-Interactive Key Exchange (NIKE)

Our generic scheme additionally builds upon a Non-Interactive Key Exchange (NIKE) [7, 16, 26] scheme. We additionally define a verification algorithm that checks whether a tuple is in fact a valid keypair.

Definition 1. *A* **Non-Interactive Key Exchange (NIKE)** NK *with verification is the tuple* $\mathsf{NK} = (\mathsf{KeyGen}, \mathsf{Verify}, \mathsf{SharedKey})$*, where*

- $\mathsf{KeyGen}(\lambda) \to (\mathsf{sk}, \mathsf{pk})$*: Generate a secret key* sk *and corresponding public key* pk *with respect to the security parameter* λ*.*
- $\mathsf{Verify}(\mathsf{sk}, \mathsf{pk}) \to \{0, 1\}$*: Verify that* $(\mathsf{sk}, \mathsf{pk})$ *is a valid output from* NK.KeyGen*.*
- $\mathsf{SharedKey}(\mathsf{sk}_1, \mathsf{pk}_2) \to \psi$*: Output the shared key* ψ *generated by the combination of one party's secret key* sk_1 *and the other party's public key* pk_2*.*
- *For correctness, we require that, for all* $(\mathsf{sk}_1, \mathsf{pk}_1) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda), (\mathsf{sk}_2, \mathsf{pk}_2) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda)$*, the following conditions hold:*

$$\mathsf{NK.Verify}(\mathsf{sk}_i, \mathsf{pk}_i) = 1, \text{ for } i \in \{1, 2\} \text{ and}$$

$$\mathsf{NK.SharedKey}(\mathsf{sk}_1, \mathsf{pk}_2) = \mathsf{NK.SharedKey}(\mathsf{sk}_2, \mathsf{pk}_1)$$

In addition to the requirement that a NIKE be *session-key unrecoverable*, we additionally require that a NIKE be *binding*. We discuss both properties next.

Session-Key Recovery. A NIKE that is *session-key unrecoverable* ensures that given the public keys of two parties $\mathsf{pk}_1, \mathsf{pk}_2$, it should be hard for an adversary to obtain the shared secret. While prior security notions for non-interactive key exchange exist in the literature [16, 26], these notions allow an adversary to act as an active participant, and assume a distinguishing game. Our constructions require only the more restricted model where the adversary learns key material but does not contribute itself, and must compute the shared secret directly. We present this restricted notion in Figure 1.

The advantage of an adversary $\mathcal{A}$ against NK in the unrecoverability experiment as defined in Figure 1 is

$$\mathsf{Adv}^{\mathrm{rec}}_{\mathsf{NK},\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\mathrm{rec}}_{\mathsf{NK},\mathcal{A}}(\lambda) = 1] \,.$$

Definition 2. *A NIKE is* **session-key unrecoverable** *if for all probabilistic polynomial time adversaries* $\mathcal{A}$*, the function* $\mathsf{Adv}^{\mathrm{rec}}_{\mathsf{NK},\mathcal{A}}(\lambda)$ *is negligible.*

$\mathsf{Exp}^{\mathrm{bind}}_{\mathsf{NK},\mathcal{A}}(\lambda)$

1 : $(\mathsf{sk}_1^*, \mathsf{pk}_1^*, \mathsf{sk}_2^*, \mathsf{pk}_2^*) \xleftarrow{\$} \mathcal{A}(\lambda)$
2 : **return** 0 **if** $\mathsf{NK.Verify}(\mathsf{sk}_1^*, \mathsf{pk}_1^*) \neq 1$
3 : **return** 0 **if** $\mathsf{NK.Verify}(\mathsf{sk}_2^*, \mathsf{pk}_2^*) \neq 1$
4 : **return** 1 **if** $\mathsf{NK.SharedKey}(\mathsf{sk}_1^*, \mathsf{pk}_2^*) \neq \mathsf{NK.SharedKey}(\mathsf{sk}_2^*, \mathsf{pk}_1^*)$
5 : **return** 0

Fig. 2. Binding experiment defining the advantage of an adversary $\mathcal{A}$ against a non-interactive key exchange (NIKE) NK. Here, the adversary can perform all NK operations directly.

Binding. We introduce an additional security property that must hold when the adversary is allowed to generate both sides of the key exchange. Informally, a NIKE is *binding* when $\mathsf{NK.SharedKey}(\mathsf{sk}_1, \mathsf{pk}_2)$ must equal $\mathsf{NK.SharedKey}(\mathsf{sk}_2, \mathsf{pk}_1)$, even when the adversary is allowed to generate $(\mathsf{sk}_1, \mathsf{pk}_1), (\mathsf{sk}_2, \mathsf{pk}_2)$ itself. The only restriction to the adversary is that both keypairs must be valid. We formalize this requirement in Figure 2.

The advantage of an adversary $\mathcal{A}$ against NK in the binding experiment as defined in Figure 2 is

$$\mathsf{Adv}^{\mathrm{bind}}_{\mathsf{NK},\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\mathrm{bind}}_{\mathsf{NK},\mathcal{A}}(\lambda) = 1] \,.$$

Definition 3. *A NIKE is* **binding** *if for all probabilistic polynomial time adversaries $\mathcal{A}$, the function $\mathsf{Adv}^{\mathrm{bind}}_{\mathsf{NK},\mathcal{A}}(\lambda)$ is negligible.*

A Concrete NIKE. We now describe a concrete NIKE which is a non-interactive Diffie–Hellman key exchange, with additionally a verification check to ensure that $\mathsf{pk} = g^{\mathsf{sk}}$.

- $\mathsf{NK.KeyGen}(\lambda) \to (\mathsf{sk}, \mathsf{pk})$: Sample secret key $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$; derive public key $\mathsf{pk} \in \mathbb{G}$ as $\mathsf{pk} = g^{\mathsf{sk}}$. Output $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{NK.Verify}(\mathsf{sk}, \mathsf{pk}) \to \{0, 1\}$: Output 1 if $\mathsf{pk} = g^{\mathsf{sk}}$; otherwise, output 0.
- $\mathsf{NK.SharedKey}(\mathsf{sk}_1, \mathsf{pk}_2) \to \psi$: Derive the shared key $\psi \in \mathbb{G}$ as $\psi \leftarrow \mathsf{pk}_2^{\mathsf{sk}_1}$. Output ψ.

We demonstrate in Appendix A that the concrete scheme is unrecoverable assuming the Computational Diffie-Hellman (CDH) problem is hard, and is unconditionally binding.

2.3 Network Model

Underlying any DKG is the requirement that participants send and receive messages over a network channel. We now describe these channels and assumptions.

Synchronicity. In this work, we assume a DKG that operates purely in the synchronous model. In other words, we assume all honest parties wait to proceed to a subsequent round until they have received the expected inputs from all other participants from a prior round.

Broadcast channels. We assume an idealized broadcast channel. We represent messages sent and received over this idealized channel as inb_i, which is the set of broadcast messages that is received as input for the i^{th} protocol round. inb_i consists of broadcast messages $\mathsf{bmsg}_1, \ldots, \mathsf{bmsg}_n$, sent by participants $1, \ldots, n$. Participant k is the sender of a broadcast message bmsg_k, and all other participants are the receivers of this message.

Broadcast channels have the following properties:

1. *Consistent.* Each participant has the same view of the message sent over the channel.
2. *Authenticated.* Players know that the message was in fact sent by the claimed sender. In practice, this requirement is often fulfilled by a PKI.
3. *Reliable Delivery.* Player i knows that the message it sent was in fact received by the intended participants.
4. *Unordered.* The channel does *not* guarantee ordering of messages.

Peer to peer channels. We assume an idealized peer-to-peer channel. We represent messages sent and received over this channel as inp_i and outp_i, inp_i is the vector of peer-to-peer messages that is received as input for protocol round i, and outp_i is the vector of peer-to-peer messages that is output at the end of network round i. inp_i and outp_i consist of peer-to-peer messages which we denote as $\mathsf{pmsg}_{i,j}$, where here, participant i is the sender and participant j is the receiver.

Peer-to-peer channels are authenticated, reliable, and unordered, per the definitions above. Additionally, peer-to-peer channels are *confidential*; i.e., only participants i and j are allowed to know the contents of $\mathsf{pmsg}_{i,j}$.

Remark 2 (Rushing Adversaries). A **rushing adversary** is one that "speaks last;" i.e., it waits to receive inputs from honest participants before publishing its own. Importantly, a rushing adversary may be able to choose its input *adaptively* after observing inputs from all other participants. Our security definitions in Section 4 consider such an adversary, by allowing the adversary to query for honest participants' contributions in a round before publishing its own.

3 A Formalization of Verifiable Secret Sharing

We now more formally introduce Verifiable Secret Sharing (VSS). We then extend the notion of a VSS to one that is aggregatable.

3.1 Verifiable Secret Sharing (VSS)

Verifiable Secret Sharing (VSS) allows a dealer to share a secret s in such a way that participants can ensure that combining their shares allows for recovery of a secret that corresponds to a public commitment to s, without learning s in the process. Participants verify that their shares can correctly recover s using this commitment. While prior definitions of VSS schemes [32] define algorithms to issue and verify shares, and recover the secret, we additionally define an algorithm to derive shares in a public domain. This notion has been employed implicitly in prior DKGs [34], and we formalize this construct here. Under the hood, our definition of a VSS encompasses both a secret-sharing primitive and a polynomial commitment primitive. It is possible breaking these notions into separate primitives may be useful for future work, as is the approach taken by Abraham et al. [1].

Definition 4. *A* **Verifiable Secret Sharing (VSS) scheme** S *is the tuple* $(\mathsf{Share}, \mathsf{Recover}, \mathsf{Verify}, \mathsf{GetPub})$ *and parameterized by a one-way map* $\mathsf{SecretToPublic} : \hat{S} \to \hat{P}$ *that maps elements in the secret domain* $\hat{S}$ *to the public domain* $\hat{P}$*, where*

- $\mathsf{Share}(\lambda, s, n, t) \to (\{(1, w_1), \ldots, (n, w_n)\}, D)$*: A probabilistic algorithm performed by a dealer that accepts as input the security parameter* λ*, the secret* $s \in \hat{S}$ *from the secret domain* $\hat{S}$*, the number of participants* n*, and the threshold* t*, where* n *and* t *are positive integers, and* $n \geq t$*. Outputs the list of shares* $\{(1, w_1), \ldots, (n, w_n)\}$ *and commitment* D*.*
- $\mathsf{Verify}(i, w_i, D) \to \{0, 1\}$*: A deterministic algorithm performed by a recipient of a share that accepts an identifier* i*, a share* w_i*, and a commitment* D*. Outputs* 1 *if the share is valid with respect to* D*, otherwise outputs* 0*.*
- $\mathsf{Recover}(t, \mathsf{M}) \to s/\mathsf{fail}$*: A deterministic algorithm that accepts a recovery set* $\mathsf{M} = \{(j, w_j)\}_{j \in C}$*, where* C *is a set of participant identifiers such that* $|C| \geq t$*. Employ* M *to obtain* s*, or output* fail *in the case of failure.*

$\mathsf{Exp}^{\mathrm{sec}}_{\mathsf{S},\mathcal{A},\mathsf{SimShare}}(\lambda, s)$

$b \xleftarrow{\$} \{0,1\}$
$(n, t, \mathsf{corrupt}, \mathsf{st}_A) \xleftarrow{\$} \mathcal{A}(\lambda)$
return 0 **if** $t > n$ **or** $\mathsf{corrupt} \not\subseteq [n]$ **or** $|\mathsf{corrupt}| \geq t$
if $b = 0$
  $(\{(j, w_j)\}_{j\in[n]}, D) \leftarrow \mathsf{S.Share}(\lambda, s, n, t)$
else // $b = 1$ *case*
  $\Delta \xleftarrow{\$} \mathsf{S.SecretToPublic}(s)$
  $(\{(j, w_j)\}_{j\in\mathsf{corrupt}}, D) \xleftarrow{\$} \mathsf{SimShare}(\Delta, n, t, \mathsf{corrupt})$
  for $j \in \mathsf{corrupt}$ **do**
    return 1 **if** $\mathsf{S.Verify}(j, w_j, D) \neq 1$
    // *The simulation must produce* $|\mathsf{corrupt}|$ *valid shares*
  return 1 **if** $\mathsf{S.GetPub}(0, D) \neq \Delta$
    // *The commitment must be valid with respect to* Δ
$b' \xleftarrow{\$} \mathcal{A}(\mathsf{st}_A, \{(j, w_j)\}_{j\in\mathsf{corrupt}}, D)$
return 1 **if** $b' = b$
return 0

Fig. 3. Secrecy experiment defining the advantage of an adversary $\mathcal{A}$ against a Verifiable Secret Sharing scheme (VSS) S that is zero-indexed.

– $\mathsf{GetPub}(i, D) \to W_i$*: A deterministic algorithm that accepts an identifier* i *and a commitment* D*. Outputs a share in the public domain* W_i *associated with identifier* i*.*

For correctness, for all security parameters λ, for all $s \in \hat{S}$, and n, t, C, such that $n \geq t$, $t \geq 1$, $C \subseteq [n]$, and $|C| \geq t$, the following must hold:

$$\mathsf{S.Verify}(i, w_i, D) = 1 \text{ for all } i \in [n], \text{ and}$$
$$\mathsf{S.GetPub}(i, D) = \mathsf{S.SecretToPublic}(w_i) \text{ for all } i \in [n], \text{ and}$$
$$\mathsf{S.Recover}(t, \mathsf{M}) = s, \text{ where}$$
$$\mathsf{S.Share}(\lambda, s, n, t) \to (\{(i, w_i)\}_{i\in[n]}, D) \text{ and } \mathsf{M} = \{(i, w_i)\}_{i\in C}$$

Zero-Indexing. For the purposes of our definitions, we additionally require that a VSS be *zero-indexed*, which we define next.

Definition 5. *A* **zero-indexed VSS** *is one which satisfies the correctness condition augmented with the following line:*

$$\mathsf{S.Verify}(0, s, D) = 1 \text{ and } \mathsf{S.GetPub}(0, D) = \mathsf{S.SecretToPublic}(s)\ . \tag{1}$$

A prototypical example of a zero-indexed VSS is Feldman's secret sharing [23], which itself builds on Shamir secret sharing [42].

We now employ the notion of a zero-indexed VSS to define the security notions of secrecy and uniqueness for VSS schemes, building upon prior definitions in the literature [9, 32], but formalized in game-based notation.

Secrecy. A VSS scheme is *secret* if an adversary that is allowed to corrupt up to $t-1$ players has a negligible probability of learning the secret s. This notion can be also expressed in terms of *simulatability*; i.e., an

$\mathsf{Exp}^{\mathrm{unq}}_{\mathsf{S},\mathcal{A}}(\lambda)$

$(t^*, \mathsf{M}_1^*, \mathsf{M}_2^*, D^*) \xleftarrow{\$} \mathcal{A}(\lambda)$
// *$\mathcal{A}$ chooses two recovery sets and a commitment*
// *with respect to the same threshold*
for $(i, w_i) \in \mathsf{M}_1^*$ **do**
  return 0 **if** $\mathsf{S.Verify}(i, w_i, D^*) \neq 1$
for $(i', w_i') \in \mathsf{M}_2^*$ **do**
  return 0 **if** $\mathsf{S.Verify}(i', w_i', D^*) \neq 1$
return 0 **if** $\mathsf{S.Recover}(t^*, \mathsf{M}_1^*) = \mathsf{fail} \vee \mathsf{S.Recover}(t^*, \mathsf{M}_2^*) = \mathsf{fail}$
return 1 **if** $\mathsf{S.Recover}(t^*, \mathsf{M}_1^*) \neq \mathsf{S.Recover}(t^*, \mathsf{M}_2^*)$
// *$\mathcal{A}$ wins if it can generate two different recovery sets for the same*
// *commitment, but which recover distinct secrets*
return 0

Fig. 4. Uniqueness experiment defining the advantage of an adversary $\mathcal{A}$ against a Verifiable Secret Sharing scheme (VSS) S.

environment that can simulate secret sharing for an unknown challenge ensures that the adversary learns no additional information than is known by the environment. We formalize this notion in Figure 3, with respect to an algorithm $\mathsf{SimShare}$ that is defined in the context of the concrete construction. The experiment takes as input any secret s from the secret space $\hat{S}$. Importantly, we do not restrict secrets to be sampled uniformly at random; even degenerate cases should hold.

The advantage of an adversary $\mathcal{A}$ against a VSS scheme S in the secrecy experiment as defined in Figure 3 is

$$\mathsf{Adv}^{\mathrm{sec}}_{\mathsf{S},\mathcal{A},\mathsf{SimShare}}(\lambda) = \max_{s \in \hat{S}} \left(\left| \Pr[\mathsf{Exp}^{\mathrm{sec}}_{\mathsf{S},\mathcal{A},\mathsf{SimShare}}(\lambda, s) = 1] - 1/2 \right| \right) .$$

Definition 6. *A VSS scheme* S *is* **secret** *if there exists an algorithm* $\mathsf{SimShare}$ *such that for all probabilistic polynomial time adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{sec}}_{\mathsf{S},\mathcal{A},\mathsf{SimShare}}(\lambda)$ *is a negligible function of* λ.

Uniqueness. Informally, a VSS scheme S that is *unique* is one where the public commitment D uniquely determines the output from $\mathsf{S.Recover}$. We formalize this notion in Figure 4.

The advantage of an adversary $\mathcal{A}$ against S in the uniqueness experiment as defined in Figure 4 is

$$\mathsf{Adv}^{\mathrm{unq}}_{\mathsf{S},\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\mathrm{unq}}_{\mathsf{S},\mathcal{A}}(\lambda) = 1] .$$

Definition 7. *A VSS scheme is* **unique** *if for all probabilistic polynomial time adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{unq}}_{\mathsf{S},\mathcal{A}}(\lambda)$ *is a negligible function of* λ.

A Concrete VSS. We now recall Feldman's Verifiable Secret Sharing [23]. Feldman's VSS itself is an augmentation to Shamir's secret sharing [42]. Here, the map $\mathsf{S.SecretToPublic}$ is simply $a \mapsto g^a$, where g is the generator of the group $\mathbb{G}$, and $a \in \mathbb{Z}_q$.

- $\mathsf{S.Share}(\lambda, s, n, t) \to (\{(1, w_1), \ldots, (n, w_n)\}, D)$. Accepts as input the security parameter λ, a secret $s \in \mathbb{Z}_q$, the number of participants $n \in \mathbb{N}$, and the threshold $t \in \mathbb{N}$ such that $1 \leq t \leq n$. Perform the following steps:
  1. Sample $t-1$ coefficients at random: $(a_1, \ldots, a_{t-1}) \xleftarrow{\$} \mathbb{Z}_q^{t-1}$.
  2. Using s as the constant term and $a_1, \ldots, a_{t-1}$ as the remaining coefficients, define the $(t-1)$-degree polynomial $f(x) = s + \sum_{i=1}^{t-1} a_i x^i$.

3. Generate n shares $w_j \in \mathbb{Z}_q$ by deriving $w_j \leftarrow f(j),\ j \in [n]$.
4. Define $A_0 = g^s$ and $A_i = g^{a_i}$, for $i \in [t-1]$.
5. Define the VSS commitment D as the vector of commitments to the coefficients of f: $D \leftarrow (A_0, \ldots, A_{t-1})$.
6. Output $(\{(i, w_i)\}_{i\in[n]}, D)$, where each i represents the identifier of the recipient of w_i.

- $\mathsf{S.Verify}(i, w_i, D) \to \{0,1\}$: Accepts as input a participant identifier $i \in [n]$, the share w_i that belongs to participant i, and a VSS commitment D that is a tuple of group elements, such that $|D| = t$. Verifies that (i, w_i) is a valid point on f as follows:
  1. Parse $\langle A_0, \ldots, A_{t-1}\rangle \leftarrow D$.
  2. Output 1 if Equation 2 holds; otherwise, output 0.

$$g^{w_i} \stackrel{?}{=} \prod_{k=0}^{t-1} A_k^{i^k} \tag{2}$$

- $\mathsf{S.Recover}(t, \mathsf{M}) \to s/\mathsf{fail}$: Accepts as input the threshold t, a recovery set $\mathsf{M} = \{(j, w_j)\}_{j\in C}$ consisting of secret shares for a coalition $C \subseteq [n]$ where $|C| \geq t$. If $|C| < t$, output fail. Using the inputs, perform the following steps:
  1. Derive Lagrange coefficients for each identifier $\{L_j(0)\}_{j\in C}$.
  2. Derive $s = \sum_{j\in C} w_j \cdot L_j(0)$.
  3. Output s.

– $\mathsf{S.GetPub}(i, D) \to W_i$: Accepts as input a participant identifier i and a VSS commitment. Outputs $W_i = g^{f(i)}$ by computing

$$W_i \leftarrow \prod_{k=0}^{|D|-1} A_k^{i^k}\ .$$

Correctness and security. For Feldman's VSS, correctness is straightforward to verify. For completeness, we discuss its security in Appendix B.

3.2 Aggregatable Verifiable Secret Sharing (AgVSS)

We now augment the notion of a verifiable secret sharing scheme as presented in Definition 4 to define an *aggregatable* verifiable secret sharing scheme (AgVSS) AV. The ability to aggregate Feldman VSS has been demonstrated before [30], and is employed implicitly in nearly every DKG that has been defined in the literature [27, 29, 37, 40] but we formalize the notion of an AgVSS more generally.

An AgVSS builds upon a base VSS scheme, and is defined with respect to the same map $\mathsf{SecretToPublic} : \hat{S} \to \hat{P}$ as the base VSS that maps elements in the secret space $\hat{S}$ to the public space $\hat{P}$. It is also defined with respect to a random oracle H, and an auxiliary distribution AX that has guessing entropy linear to the security parameter λ.

Definition 8. *An* **Aggregatable Verifiable Secret Sharing (AgVSS) scheme** $\mathsf{AV}[\mathsf{H}]$ *is the tuple* $(\mathsf{Share}, \mathsf{Recover}, \mathsf{Verify}, \mathsf{GetPub}, \mathsf{AggPriv}, \mathsf{AggPub}, \mathsf{GetTweak})$*, where*
$\mathsf{Share}, \mathsf{Recover}, \mathsf{Verify}, \mathsf{GetPub}$ *are identical to the base VSS scheme as in Definition 4, and where*

– $\mathsf{GetTweak}(O, \mathsf{aux}) \to v$*: A deterministic algorithm that accepts as input a public aggregation set* O *consisting of* ℓ *VSS commitments* $\{D_1, \ldots, D_\ell\}$*, and an auxiliary string* aux *drawn from* AX*. Output a tweak* $v \in \hat{S}$*.*

– $\mathsf{AggPriv}(P, v) \to \hat{w}_i$*: A deterministic algorithm that accepts as input a private aggregation set* P *consisting of* ℓ *shares* $\{w_{ji}\}_{j\in[\ell]}$*, and a tweak* v*. Output the aggregated share* $\hat{w}_i$*.*

– $\mathsf{AggPub}(O, v) \to C$*: A deterministic algorithm that accepts as input a public aggregation set* O *consisting of* ℓ *VSS commitments* $\{D_1, \ldots, D_\ell\}$*, and a tweak* v*. Output the aggregated commitment* C*.*

Similarly to the VSS setting, we require that the AgVSS be zero-indexed.

$\mathsf{Exp}^{\mathrm{corr}}_{\mathsf{AV}[\mathsf{H}]}(\lambda, n, t, C, \ell)$

1 : **return** 0 **if** $t \geq n \vee C \not\subseteq [n] \vee |C| < t$
2 : $\mathsf{aux} \xleftarrow{\$} \mathsf{AV.AX}$
3 : **for** $j \in [\ell]$ **do**
4 : $\quad s_j \xleftarrow{\$} \hat{S};\ (\{(i, w_{ji})\}_{i\in[n]}, D_j) \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{Share}(\lambda, s_j, n, t)$
5 : $v \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{GetTweak}(O, \mathsf{aux})$
6 : **for** $i \in [n]$ **do**
7 : $\quad \hat{w}_i \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{AggPriv}(\{w_{ji}\}_{j\in[\ell]}, \{D_j\}_{j\in[\ell]}, v)$
8 : $C \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{AggPub}(\{D_j\}_{j\in[\ell]}, v)$
9 : **for** $i \in [n]$ **do**
10 : $\quad$ **return** 0 **if** $\mathsf{AV}[\mathsf{H}].\mathsf{Verify}(i, \hat{w}_i, C) \neq 1$
11 : $\hat{s} \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{Recover}(t, \{(k, \hat{w}_k)\}_{k\in C}, C)$
12 : **return** 0 **if** $\mathsf{AV}[\mathsf{H}].\mathsf{Verify}(0, \hat{s}, C) \neq 1$
13 : $\quad$ // *This check assumes the AgVSS is zero-indexed*
14 : **return** 1

Fig. 5. Notion of correctness for an AgVSS.

Aggregation Correctness. An AgVSS must fulfill the notion of correctness for a VSS scheme. In addition, an AgVSS must fulfill the notion of correctness when the scheme is aggregatable; i.e., that for all n, t, ℓ, C, where $n, t, \ell \in \mathbb{N}$, $C \subseteq [n]$, and $|C| \geq t$,

$$\Pr\left[\mathsf{Exp}^{\mathrm{corr}}_{\mathsf{AV}[\mathsf{H}]}(\lambda, n, t, C, \ell) = 1\right] = 1$$

where $\mathsf{Exp}^{\mathrm{corr}}_{\mathsf{AV}}(\lambda, n, t, C, \ell)$ is presented in Figure 5.

An AgVSS must satisfy aggregated secrecy and uniqueness, as discussed next.

Aggregated Secrecy. Unlike in the plain VSS setting, an AgVSS allows for aggregating shares and commitments that could both be honestly as well as maliciously generated. Hence, we require that the AgVSS fulfills the notion of *aggregated secrecy*, which we formalize in Figure 6.

Remark 3 (Aggregated secrecy for an AgVSS is a generalization of VSS secrecy). The secrecy game for a VSS scheme as shown in Figure 3 only allows the adversary to receive a single set of shares and their corresponding commitment from the environment. Hence, the aggregated secrecy game in Figure 6 is a strict generalization of the VSS secrecy game, as the adversary in the secure aggregation game can not only query for many shares and commitments, but submit its own. As a consequence, any AgVSS that fulfills aggregated secrecy is also secret.

When executing the experiment shown in Figure 6, the environment simulates one secret sharing operation with respect to a challenge Δ when the random bit $b = 1$. The environment the allows the adversary to query for corrupted parties' shares by $\mathcal{O}^{\mathsf{GetShare}}$, and submit its own shares for honest parties by $\mathcal{O}^{\mathsf{RecvShare}}$. The adversary is then provided with the auxiliary string aux. The adversary automatically wins if the aggregated commitment AV does not fulfill the check $\mathsf{AV}[\mathsf{H}].\mathsf{GetPub}(0, C) \neq \Delta$. Otherwise, $\mathcal{A}$ outputs a guess b' to guess if it is in the real or simulated environment.

The advantage of an adversary $\mathcal{A}$ against AV in the aggregated secrecy experiment as defined in Figure 6 with respect to algorithms $\mathsf{SimShare}, \mathsf{SimTweak}$ is

$$\mathsf{Adv}^{\mathrm{asec}}_{\mathsf{AV}[\mathsf{H}],\mathcal{A},\mathsf{SimShare},\mathsf{SimTweak}}(\lambda) = \max_{s\in\hat{S}} \left|\Pr[\mathsf{Exp}^{\mathrm{asec}}_{\mathsf{AV}[\mathsf{H}],\mathcal{A},\mathsf{SimShare},\mathsf{SimTweak}}(\lambda, s) = 1] - 1/2\right| .$$

$\mathsf{Exp}^{\mathrm{asec}}_{\mathsf{AV}[\mathsf{H}],\mathcal{A},\mathsf{SimShare}}(\lambda, s)$

$b \xleftarrow{\$} \{0,1\}$
$\Delta \xleftarrow{\$} \mathsf{AV.SecretToPublic}(s)$
$\mathsf{issuedchal} \leftarrow 0$
$\mathsf{aux} \xleftarrow{\$} \mathsf{AV.AX}$
// *Pick random auxiliary string*
$\ell' \leftarrow 0$
$Q_1 \leftarrow \emptyset, Q_2 \leftarrow \emptyset$
$(n, t, \mathsf{corrupt}, \mathsf{st}_A) \xleftarrow{\$} \mathcal{A}(\lambda)$
if $t > n \vee \mathsf{corrupt} \not\subseteq [n]$
or if $|\mathsf{corrupt}| \geq t$
  $z \xleftarrow{\$} \{0,1\}$; **return** z
$\mathsf{honest} \leftarrow [n] \setminus \mathsf{corrupt}$
$\mathsf{st}'_A \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{GetShare}}, \mathcal{O}^{\mathsf{RecvShare}}, \mathcal{O}^{\mathsf{GetTweak}}}(\mathsf{st}_A)$
if $\mathsf{issuedchal} = 0$
  $z \xleftarrow{\$} \{0,1\}$; **return** z
if $b = 1$
  $\mathsf{in} \leftarrow ((\{D_j\}_{j \in [\ell']} \cup \{D^*\}), \mathsf{aux})$
  $C \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{AggPub}(\mathsf{in})$
  if $\mathsf{AV}[\mathsf{H}].\mathsf{GetPub}(0, C) \neq \Delta$
    return 1
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{GetTweak}}}(\mathsf{st}'_A, \mathsf{aux})$
// $\mathcal{A}$ *performs*
// *aggregation directly*
return 1 **if** $b' = b$
return 0

$\mathcal{O}^{\mathsf{GetShare}}()$ // *Performs secret sharing*

if $\mathsf{issuedchal} = 0$
  $\mathsf{issuedchal} \leftarrow 1$
  if $b = 0$
    $(\{(j, w_j)\}_{j \in [n]}, D) \xleftarrow{\$} \mathsf{AV}[\mathsf{H}].\mathsf{Share}(\lambda, s, n, t)$
    $Q_2 \leftarrow Q_2 \cup \{(\{(j, w_j)\}_{j \in \mathsf{honest}}, , D)\}$
    return $(\{(j, w_j)\}_{j \in \mathsf{corrupt}}, D)$
  else // $b = 1$ *case*
    out $\xleftarrow{\$} \mathsf{SimShare}(\lambda, \mathsf{corrupt}, \Delta)$
    $(\alpha^*, \{(j, w_j)\}_{j \in \mathsf{corrupt}}, D^*) \leftarrow$ **out**
    $Q_1 \leftarrow (\alpha^*, D^*)$
  return $(\{(j, w_j)\}_{j \in \mathsf{corrupt}}, D^*)$
$\ell' \leftarrow \ell' + 1$
$s_{\ell'} \xleftarrow{\$} \hat{S}$
$(\{(j, w_{\ell' j})\}_{j \in [n]}, D_{\ell'}) \xleftarrow{\$} \mathsf{AV}[\mathsf{H}].\mathsf{Share}(\lambda, s_{\ell'}, n, t)$
$Q_2 \leftarrow Q_2 \cup \{(\{(j, w_{\ell' j})\}_{j \in \mathsf{honest}}, , D_{\ell'})\}$
return $(\{(j, w_{\ell' j})\}_{j \in \mathsf{corrupt}}, D_{\ell'})$

$\mathcal{O}^{\mathsf{RecvShare}}(\{(j, w_{\ell' j})\}_{j \in \mathsf{honest}}, D_{\ell'})$

// *Receives shares from* $\mathcal{A}$
$\ell' \leftarrow \ell' + 1$
for $j \in \mathsf{honest}$
  if $\mathsf{AV}[\mathsf{H}].\mathsf{Verify}(j, w_{\ell' j}, D_{\ell'}) \neq 1$
    return $\perp$
$Q_2 \leftarrow Q_2 \cup \{(\{(j, w_{\ell' j})\}_{j \in \mathsf{honest}}, D_{\ell'})\}$

$\mathcal{O}^{\mathsf{GetTweak}}(O_i, \mathsf{aux}_i)$

if $b = 1$
  $v \leftarrow \mathsf{SimTweak}(O_i, \mathsf{aux}_i, Q_1, Q_2)$
else
  $v \leftarrow \mathsf{AV.GetTweak}(O_i, \mathsf{aux}_i)$
return v

Fig. 6. Aggregated secrecy experiment defining the advantage of an adversary $\mathcal{A}$ against a zero-indexed Aggregatable Verifiable Secret Sharing scheme (AgVSS) AV.

Definition 9. *An AgVSS scheme* AV *achieves* **aggregated secrecy** *if there exist algorithms* SimShare, SimTweak *such that for all probabilistic polynomial time adversaries* $\mathcal{A}$*, the function* $\mathsf{Adv}^{\mathrm{asec}}_{\mathsf{AV}[\mathsf{H}],\mathcal{A},\mathsf{SimShare},\mathsf{SimTweak}}(\lambda)$ *is a negligible function of* λ*.*

Uniqueness. The uniqueness property for an AgVSS is the same as that of the base VSS, as presented in Figure 4. Notably, the adversary in this experiment can perform aggregation directly, and so the experiment does not change.

A Concrete AgVSS. We now present a concrete AgVSS scheme that builds upon Feldman's VSS, and assumes an honest majority of participants. The algorithms $\mathsf{Share}, \mathsf{Recover}, \mathsf{Verify}, \mathsf{GetPub}$ are identical to Feldman's VSS, as presented in Section 3.1.

Let $\mathsf{H} : \mathbb{G}^* \times \mathbb{Z}_q \to \mathbb{Z}_q$ be a cryptographically secure hash function, whose inputs are a tuple consisting of a vector of group elements in $\mathbb{G}$, and an element in $\mathbb{Z}_q$. Finally, let the distribution AX be the uniform distribution on $\mathbb{Z}_q$.

We now define the aggregation algorithms for this concrete AgVSS.

- $\mathsf{AV.GetTweak}(O, \mathsf{aux})$: Let $O = \{D_j\}_{j\in[\ell]}$ be the public aggregation set of the corresponding ℓ Feldman VSS commitments to P, such that $|O| = t$, and O is sorted in a canonical order. Generate $v \leftarrow \mathsf{H}(O, \mathsf{aux})$. Output v.
- $\mathsf{AV.AggPriv}(P_i, v)$: Let $P = \{w_{ji}\}_{j\in[\ell]}$ be a private aggregation set of ℓ Shamir secret shares, and $v \in \mathbb{Z}_q$ be an auxiliary string.
  1. Derive the aggregate share via Equation 3.

$$\hat{w}_i \leftarrow v + \sum_{j\in[\ell]} w_{ji} \tag{3}$$

  2. Output $\hat{w}_i$
- $\mathsf{AV.AggPub}(O, v)$: Perform the following:
  1. Derive the commitment to the aggregated constant coefficient by Equation 4:

$$\hat{A}_0 = g^v \cdot \prod_{j\in[\ell]} D_j[0] \tag{4}$$

  2. Derive the commitment to the $k \in \{1, \ldots, t-1\}$ aggregated coefficients by Equation 5:

$$\hat{A}_k = \prod_{j\in[\ell]} D_j[k] \tag{5}$$

  Let $C = \langle \hat{A}_0, \ldots, \hat{A}_{t-1} \rangle$
  3. Output C

We demonstrate the correctness and security properties of the concrete scheme in Appendix C.

4 Distributed Key Generation (DKG)

Informally, a *Distributed Key Generation* (DKG) scheme allows a set of n players to jointly generate a public key pk and a secret key sk, such that all parties learn the public key pk but no single party learns sk. Instead, each party id_i has a share sk_i of the secret key, such that a threshold t number of shares are required to recover sk, where $t \leq n$. A DKG generates secret and public keys that are compatible with some *target* key generation scheme which operates in the centralized (single-party) setting.

We now formalize the definition of a DKG. We begin by defining a target single-party key generation scheme TK that the DKG implements in a distributed manner.

Definition 10. *A* **target key generation scheme** TK *is the tuple* $\mathsf{TK} = (\mathsf{KeyGen}, \mathsf{Verify})$*, where:*

- $\mathsf{KeyGen}(\lambda) \to (\mathsf{sk}, \mathsf{pk})$*: A probabilistic algorithm that accepts as input a security parameter* λ *and outputs a secret and public keypair* $(\mathsf{sk}, \mathsf{pk})$*, where* sk *is an element of a secret domain* $\hat{S}$ *and* pk *is an element of the public domain* $\hat{P}$*.*
- $\mathsf{Verify}(\mathsf{sk}, \mathsf{pk}) \to \{0, 1\}$*: A deterministic algorithm that accepts as input a secret and public keypair* $(\mathsf{sk}, \mathsf{pk})$*, and outputs* 0 *if the keypair is invalid; otherwise, outputs* 1*.*

– *For correctness, we require that for all* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{TK.KeyGen}(\lambda)$*, we have*

$$\mathsf{TK.Verify}(\mathsf{sk}, \mathsf{pk}) = 1 \ .$$

Definition 11. *A* **distributed key generation scheme***, or* **DKG***, is the tuple of the protocols* $\mathsf{D}[\mathsf{H}] = (\mathsf{KeyGen}, \mathsf{Recover})$ *parameterized by the number of rounds* numrounds*. A DKG is defined with respect to a target key generation scheme* TK*, and initialized with the hash function* H*, whose domain and codomain depend on* D*, where:*

– $\mathsf{KeyGen}(\lambda, n, t) \to \langle(\mathsf{pk}, \mathsf{qual}, \mathsf{aux}), (\mathsf{status}_i, \mathsf{sk}_i)_{i\in[n]}\rangle$*: An interactive protocol between* n *participants. Accepts as input the security parameter* λ*, and positive integers* $n \geq t \geq 1$*, where* n *is the number of participants and* t *is the threshold.*
The protocol outputs a public value $(\mathsf{pk}, \mathsf{qual}, \mathsf{aux})$ *where:*
- pk *is the public key representing the group; it will be* $\perp$ *if the protocol terminated unsuccessfully.*
- qual $\subseteq [n]$ *is the set of qualified players remaining at the end of the protocol; it will be* $\perp$ *if the protocol terminated unsuccessfully.*
- aux *is any auxiliary information that may be required for a particular application, such as participant public keys,*
- status_i *indicates if participant* i *completed the protocol successfully, and can be either* abort*,* fail*, or* accept*. Here,* abort *indicates the protocol terminated early,* fail *indicates a critical fail state has been reached, and* accept *indicates that the protocol completed successfully.*
- sk_i *is participant* i*'s secret key share; it will be* $\perp$ *if the protocol terminated unsuccessfully.*

Internally to the KeyGen *protocol, each participant performs the following:*
- $\mathsf{PerformRound}_0(\lambda, n, t, \mathsf{id}_i) \to (\mathsf{state}_i, \mathsf{outp}, \mathsf{bmsg}_i)$*: The algorithm run by the participant to initiate the protocol. Accepts the security parameter* λ*, the number of participants* n*, the threshold* t*, and the participant identifier* id_i*. Outputs the state* state_i *for that participant, the vector* outp *of any outgoing peer-to-peer messages to other parties, and a broadcast message* bmsg_i *to all other parties. See Section 2.3 for definitions of these network channels.*
- $\mathsf{PerformRound}_k(\mathsf{state}_i, \mathsf{inp}, \mathsf{inb}) \to (\mathsf{state}_i, \mathsf{outp}, \mathsf{bmsg}_i)$*: The algorithm run for each intermediate round* $k \in \{1, \ldots, \mathsf{numrounds} - 1\}$ *in the protocol. Accepts as input the participant's internal state* state_i*, the vector* inp *of any peer-to-peer messages sent by other participants in the previous round, and the set* inb *of any broadcast messages sent by other participants in the previous round. Outputs the tuple* $(\mathsf{state}_i, \mathsf{outp}, \mathsf{bmsg}_i)$*.*
- $\mathsf{Finalize}(\mathsf{state}_i, \mathsf{inp}, \mathsf{inb}) \to \langle(\mathsf{pk}, \mathsf{qual}, \mathsf{aux}), (\mathsf{status}_i, \mathsf{sk}_i)_{i\in[n]}\rangle$*: Accepts participant state* state_i*, peer-to-peer messages* outp*, and broadcast messages* inb*. Outputs a public output* $(\mathsf{pk}, \mathsf{qual}, \mathsf{aux})$ *and a private output* $(\mathsf{status}_i, \mathsf{sk}_i)$*.*

– $\mathsf{Recover}(\mathsf{pk}, t, \{(i, \mathsf{sk}_i)\}_{i\in C}) \to \mathsf{sk}/\mathsf{fail}$*: The algorithm run by the participant to finish its protocol execution. Accepts as input the public key* pk*, the threshold* t*, and a set of secret key shares* $\{(i, \mathsf{sk}_i)\}_{i\in C}$*, where* $|C| \geq t$*. Derive* sk *from* $\{\mathsf{sk}_i\}_{i\in C}$*. If* $\mathsf{TK.Verify}(\mathsf{sk}, \mathsf{pk}) \neq 1$*, output* fail*. Otherwise, output* sk*.*

– *For correctness, we require that, when all participants follow the protocol:*

$$\begin{aligned} &\textit{if } \mathsf{D}[\mathsf{H}].\mathsf{KeyGen}(\lambda, n, t) \to \big((\mathsf{pk}, \mathsf{qual}), \langle(\mathsf{status}_i, \mathsf{sk}_i)\rangle_{i\in[n]}\big) \\ &\textit{then } \mathsf{D}[\mathsf{H}].\mathsf{Recover}(\mathsf{pk}, t, \{(i, \mathsf{sk}_i)\}_{i\in C}) \to \mathsf{sk} \textit{ for all } C \subseteq [n], |C| \geq t, \\ &\textit{and } \mathsf{TK.Verify}(\mathsf{sk}, \mathsf{pk}) \to 1 \end{aligned}$$

A DKG must be correct, and fulfill the security notions of zero-knowledge and indistinguishability. Additionally, a DKG may fulfill the notion of either strong or weak robustness. We explain these concepts next.

4.1 Robustness

We now review the notion of robustness, building upon the formalism presented by Gurkan et al. [30]. In the literature, robustness today refers to the property that so long as the number of corrupted and honest participants meet some pre-defined threshold, then the protocol will complete successfully. The definition by Gurkan et al. deems a DKG to be robust if honest parties' shares combine to output a secret key that is valid with respect to a public key. However, the notion by Gurkan et al. does not consider player's ending states, and also assumes a one-to-one unique mapping between elements in the secret domain and elements in the public domain.

We present several clarifications to the notion of robustness using a game-based definition in Figure 7. First, our definition is generalizable beyond discrete-logarithm based constructions. Second, our definition encodes two different forms of robustness. *Strong robustness* defines the existing notion of robustness in the literature. We additionally introduce the notion of *weak robustness*, which allows the protocol to non-fatally abort, so long as all honest participants maintain the same status. In particular, participants can exit either with an abort status or a fail status, where fail indicates a critical failure. Prior notions in the literature considered only strong robustness; however, introducing a weaker variant allows us to define a more efficient construction that fulfills our other notions of secrecy and indistinguishability.

Remark 4 (Number of Honest Players). The robustness experiment in Figure 7 assumes an honest majority of players. If an adversary were allowed to control the majority of participants, it could split the view of honest participants, unless some additional structure is assumed, such as a public bulletin board.

Strong Robustness. We show the strong robustness attack experiment in Figure 7, where its differences with weak robustness are outlined in a box. The advantage of an adversary $\mathcal{A}$ against D in the strong robustness experiment is

$$\mathsf{Adv}^{\text{str-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\text{str-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda) = 1] \ .$$

Definition 12. *A DKG* D *is* **strongly robust** *if for all probabilistic polynomial time adversaries* $\mathcal{A}$*, the function* $\mathsf{Adv}^{\text{str-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda)$ *is negligible.*

The strong robustness attack game allows $\mathcal{A}$ to participate in key generation and control up to $t-1$ players, the set of which we denote by $\mathsf{corrupt}$. We denote the set of honest players as honest. The robustness attack game has oracles $\mathcal{O}^{\mathsf{PerformRound}_r}, r \in [\mathsf{numrounds}-1], \mathcal{O}^{\mathsf{Finalize}}$, and RO in common with other games, so we show these oracles in Figure 8. The adversary participates in the protocol by exchanging peer-to-peer and broadcast messages with honest players in each round r via the oracle $\mathcal{O}^{\mathsf{PerformRound}_r}$. The experiment is responsible for exchanging messages between parties and enforcing assumptions of the underlying network channels. Additionally, the experiment ensures that rounds must be queried in order, and that no round is repeated.

$\mathcal{A}$ wins the strong robustness game under three conditions. First, if it causes any honest player to end in a non-accepting state. Second, if it causes any honest players' view of $\mathsf{pk}, \mathsf{qual}$, or aux to be different from any other player's. Finally, $\mathcal{A}$ wins if it can cause the resulting keypair to be invalid with respect to the target key generation algorithm (recall that $\mathsf{D.Recover}$ tests for validity of the keypair, and outputs fail if sk is invalid).

Weak Robustness. We similarly present the weak robustness attack experiment in Figure 7, where its differences with strong robustness are outlined in a dashed box. Intuitively, weak robustness differs from strong robustness in that it allows honest parties to end in an aborted state abort, so long as *all* honest parties end in the same state. Otherwise, if all honest parties end in an accepting state, it is identical to the notion of strong robustness.

The advantage of an adversary $\mathcal{A}$ against the DKG D in the weak robustness experiment is

$$\mathsf{Adv}^{\text{wk-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\text{wk-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda) = 1] \ .$$

$\mathsf{Exp}^{\text{str-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda)$ | $\mathsf{Exp}^{\text{wk-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda)$

$\mathsf{rc} \leftarrow 0$ // *Ensure rounds are queried in order*

$\mathsf{inp}_{11} \leftarrow \emptyset, \ldots, \mathsf{inp}_{1n} \leftarrow \emptyset$ // *Simulate peer-to-peer channels*

$(n, t, \mathsf{corrupt}, \mathsf{st}_A) \leftarrow \mathcal{A}(\lambda)$; **return** 0 **if** $n < t \vee \mathsf{corrupt} \not\subset [n] \vee |\mathsf{corrupt}| \geq t$

$\mathsf{honest} \leftarrow [n] \setminus \mathsf{corrupt}$

return 0 **if** $|\mathsf{honest}| < t$

for $i \in \mathsf{honest}$ **do**

  $(\mathsf{state}_i, \mathsf{outp}_{0i}, \mathsf{bmsg}_i) \leftarrow \mathsf{PerformRound}_0[\mathsf{H}](\lambda, n, t, i)$

  $\mathsf{inp}_{1k}[i] \leftarrow \mathsf{outp}_{0i}[k],\ \forall\ k \in [n]$

  // *Set party i peer-to-peer messages for all other parties*

$\mathcal{A}^{\mathcal{O}^{\mathsf{PerformRound}_r}, \mathcal{O}^{\mathsf{Finalize}}, \mathsf{RO}}(\mathsf{st}_A, \{\mathsf{inp}_{1i}\}_{i \in \mathsf{corrupt}}, \{\mathsf{bmsg}_i\}_{i \in \mathsf{honest}})$

return 0 **if** $\mathsf{rc} \neq \mathsf{numrounds}$ // *Prevents trivial win by forcing coordination*

if $\mathsf{honest} \neq \{i : \mathsf{status}_i = \mathsf{accept}\}_{i \in \mathsf{honest}}$

  return 1

for $i, j \in \mathsf{honest}$ **do**

  return 1 **if** $\mathsf{status}_i \neq \mathsf{status}_j$

  // *$\mathcal{A}$ wins if any honest party ends in a disjoint state*

if $\mathsf{honest} = \{i : \mathsf{status}_i = \mathsf{abort}\}_{i \in \mathsf{honest}}$

  return 0

  // *$\mathcal{A}$ loses if all honest parties end in a consistent aborted state*

for $i, j \in \mathsf{honest}$ **do**

  return 1 **if** $\mathsf{pk}_i \neq \mathsf{pk}_j$

  return 1 **if** $\mathsf{qual}_i \neq \mathsf{qual}_j$

  return 1 **if** $\mathsf{aux}_i \neq \mathsf{aux}_j$

$\mathsf{pk} = \mathsf{pk}_i, i \in \mathsf{honest}$ // *All parties at this point have the same view of* pk

$\mathsf{sk} \leftarrow \mathsf{D}[\mathsf{H}].\mathsf{Recover}(\mathsf{pk}, \{(i, \mathsf{sk}_i)\}_{i \in \mathsf{honest}})$

return 1 **if** $\mathsf{sk} = \mathsf{fail}$

// *$\mathcal{A}$ wins if honest parties' shares do not recover a valid secret*

return 0

Fig. 7. Game defining the advantage of an adversary $\mathcal{A}$ to disrupt robustness for a DKG. We show the *strong robustness* experiment in a box, and the *weak robustness* experiment in a dashed box. The oracles $\mathcal{O}^{\mathsf{PerformRound}_r}, r \in [\mathsf{numrounds}-1]$, $\mathcal{O}^{\mathsf{Finalize}}$, and RO are defined in Figure 8.

Definition 13. *A DKG* D *is* **weakly robust** *if for all probabilistic polynomial time adversaries $\mathcal{A}$, the function* $\mathsf{Adv}^{\text{wk-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda)$ *is negligible.*

$\mathcal{A}$ wins the weak robustness game under three conditions. First, $\mathcal{A}$ wins if honest players do not all finish with the same status. Otherwise, if all honest players have status accept, then $\mathcal{A}$ wins if the honest players' view of pk or qual are not all the same or if the resulting keypair is invalid with respect to the target key generation algorithm.

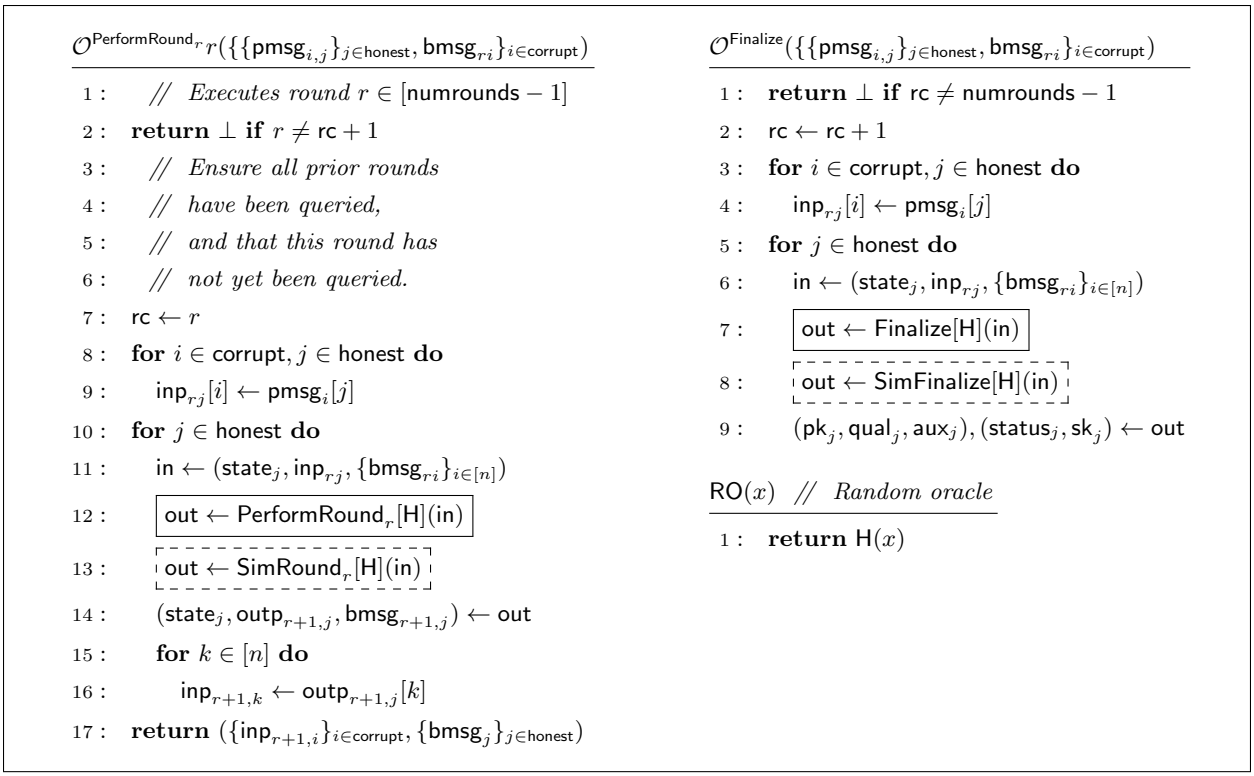

Fig. 8. Common oracles used within the games defining the security of a distributed key generation algorithm. Boxed algorithms are used by the robustness and indistinguishability games; dashed boxed algorithms are used by the zero-knowledge game.

4.2 Zero Knowledge

We next turn to defining zero-knowledge for a DKG. Zero knowledge requires that a probabilistic polynomial-time adversary $\mathcal{A}$ allowed to control up to $t-1$ participants does not gain any additional advantage in its effort to learn sk than it would against the target key generation algorithm. This notion is formalized by the requirement of *simulatability*: that the DKG can be simulated to an adversary with respect to some challenge $\hat{\Delta} \in \hat{P}$, where $\hat{\Delta}$ is a valid public key for the target key generation algorithm. We formalize this notion in Figure 9.

Our definition of zero-knowledge is distinct from the notion of secrecy introduced by Gennaro et al. [28] in several ways. First, we define the notion with respect to a target key generation algorithm, whereas the definition by Gennaro et al. is specifically within the discrete-logarithm setting. Second, Gennaro et al. assume that indistinguishability of the resulting keypair is an implicit consequence of the ability to simulate with respect to a public key. However, simulatability does not necessarily imply indistinguishability, because the zero-knowledge game *only* guarantees the indistinguisahbility of public key material between the real and simulated environments. If there is a one-to-one relation between public and secret keys, proving zero-knowledge implies indistinguishability. However, for cryptosystems where a one-to-many relation exists between public and private keys, such as in lattice-based cryptosystems, proving that a DKG is zero-knowledge does *not* imply indistinguishability from its target key generation scheme. For this reason, we instead present the notions of zero-knowledge and indistinguishability as separate notions.

We now define this notion of zero-knowledge more formally. The advantage of an adversary $\mathcal{A}$ against a DKG D with respect to the simulation algorithms $(\mathsf{SimRound}_i, i \in \{0, \ldots, \mathsf{numrounds}-1\}, \mathsf{SimFinalize})$ in the

$\mathsf{Exp}^{\mathsf{zk}}_{\mathsf{D},\mathcal{A},\mathsf{SimRound}_i,\mathsf{SimFinalize}}(\lambda)$

$b \xleftarrow{\$} \{0,1\}$
$z \xleftarrow{\$} \{0,1\}$ // *Random coin for losing conditions*
$\mathsf{rc} \leftarrow 0$ // *Ensure rounds are queried in order*
$\mathsf{inp}_{11} \leftarrow \emptyset, \ldots, \mathsf{inp}_{1n} \leftarrow \emptyset$ // *Simulate peer-to-peer channels*
$(n, t, \mathsf{corrupt}, \mathsf{st}_A) \leftarrow \mathcal{A}(\lambda)$; $\mathsf{honest} \leftarrow [n] \setminus \mathsf{corrupt}$
if $n < t \vee \mathsf{corrupt} \not\subset [n] \vee |\mathsf{corrupt}| \geq t$
  return z
$(\cdot, \hat{\Delta}) \xleftarrow{\$} \mathsf{TK.KeyGen}(\lambda)$
for $i \in \mathsf{honest}$ **do**
  if $b = 0$ **then** $(\mathsf{state}_i, \mathsf{outp}_i, \mathsf{bmsg}_i) \leftarrow \mathsf{PerformRound}_0[\mathsf{H}](\lambda, n, t, i)$
  else $(\mathsf{state}_i, \mathsf{outp}_i, \mathsf{bmsg}_i) \leftarrow \mathsf{SimRound}_0[\mathsf{H}](\lambda, n, t, i, \hat{\Delta})$
  $\mathsf{inp}_{1k}[i] \leftarrow \mathsf{outp}_{0i}[k], \ \forall\ k \in [n]$
  // *Set party i peer-to-peer messages for all other parties*
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{PerformRound}_r}, \mathcal{O}^{\mathsf{Finalize}}, \mathsf{RO}}(\mathsf{st}_A, \{\mathsf{inp}_{1j}\}_{j \in \mathsf{corrupt}}, \{\mathsf{bmsg}_i\}_{i \in \mathsf{honest}})$
if $\mathsf{rc} \neq \mathsf{numrounds}$ // *Prevents trivial win by forcing coordination*
  return z
if $\{j : \mathsf{status}_j = \mathsf{abort}\}_{j \in \mathsf{honest}} = \mathsf{honest}$
  return z
  // *$\mathcal{A}$ cannot win if all honest parties non-fatally abort*
if $b = 1$
  return 1 **if** $\exists\ i \in \mathsf{honest} : \mathsf{pk}_i \neq \hat{\Delta}$
  // *$\mathcal{A}$ wins when $b = 1$ if the game is not simulated with respect to $\hat{\Delta}$*
return 1 **if** $b \stackrel{?}{=} b'$
return 0

Fig. 9. Zero-knowledge experiment defining the advantage of an adversary $\mathcal{A}$ that controls up to $t-1$ players in a DKG. The oracles $\mathcal{O}^{\mathsf{PerformRound}_r}$ and RO are defined in Figure 8.

zero-knowledge experiment as defined in Figure 9 is

$$\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{D},\mathcal{A},\mathsf{SimRound}_i,\mathsf{SimFinalize}}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathsf{zk}}_{\mathsf{D},\mathcal{A},\mathsf{SimRound}_i,\mathsf{SimFinalize}}(\lambda) = 1] - 1/2\right| .$$

Definition 14. *A DKG* D *is* **zero-knowledge** *if there exist algorithms* $(\mathsf{SimRound}_i, i \in \{0, \ldots, \mathsf{numrounds} - 1\}, \mathsf{SimFinalize})$ *such that for all probabilistic polynomial time adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{D},\mathcal{A},\mathsf{SimRound}_i,\mathsf{SimFinalize}}(\lambda)$ *is negligible.*

The zero-knowledge attack game in Figure 9 requires $\mathcal{A}$ to successfully guess if the environment that it is in is real or simulated with respect to a challenge $\hat{\Delta}$, where $\hat{\Delta}$ is a public key generated by $\mathsf{TK.KeyGen}$. The game accepts as input the security parameter λ, and internally to the experiment, samples a bit $b \in \{0,1\}$ at random. In the simulated setting (when $b = 1$), the environment simulates the honest participants with respect to $\hat{\Delta}$ by employing the simulation algorithms $(\mathsf{SimRound}_i, i \in \{0, \ldots, \mathsf{numrounds} - 1\}, \mathsf{SimFinalize})$. The adversary wins by default if the environment cannot simulate the protocol in such a way that the resulting group public key pk does not equal $\hat{\Delta}$. Otherwise, the adversary wins if it successfully guesses whether it is playing against the real or simulated protocol.

$\mathsf{Exp}^{\mathrm{ind}}_{\mathsf{D},\mathsf{TK},\mathcal{A},\mathcal{E}}(\lambda)$

$b \xleftarrow{\$} \{0,1\}$
$z \xleftarrow{\$} \{0,1\}$ // *Random coin for losing conditions*
$\mathsf{rc} \leftarrow 0$ // *Ensure rounds are queried in order*
$\mathsf{inp}_{11} \leftarrow \emptyset, \ldots, \mathsf{inp}_{1n} \leftarrow \emptyset$ // *Simulate peer-to-peer channels*
$(n, t, \mathsf{corrupt}, \mathsf{st}_A) \leftarrow \mathcal{A}(\lambda)$
if $n < t \vee \mathsf{corrupt} \not\subset [n] \vee |\mathsf{corrupt}| \geq t$ **then** $z \xleftarrow{\$} \{0,1\}$; **return** z
$\mathsf{honest} \leftarrow [n] \setminus \mathsf{corrupt}$
for $i \in \mathsf{honest}$ **do**
  $(\mathsf{state}_i, \mathsf{outp}_{0i}, \mathsf{bmsg}_i) \leftarrow \mathsf{PerformRound}_0[\mathsf{H}](\lambda, n, t, i)$
  $\mathsf{inp}_{1k}[i] \leftarrow \mathsf{outp}_{0i}[k], \ \forall\ k \in [n]$
  // *Set party i peer-to-peer messages for all other parties*
$\mathcal{A}^{\mathcal{O}^{\mathsf{PerformRound}_r}, \mathcal{O}^{\mathsf{Finalize}}, \mathsf{RO}}(\{\mathsf{inp}_{1i}\}_{i \in \mathsf{corrupt}}, \{\mathsf{bmsg}_i\}_{i \in \mathsf{honest}}, \mathsf{st}_A)$
if $\mathsf{rc} \neq \mathsf{numrounds}$ // *Prevents trivial win by forcing coordination*
  return z
if $\exists i \in \mathsf{honest} : \mathsf{status}_i \neq \mathsf{accept}$
  return z
  // *Honest parties must complete successfully*
for $i, j \in \mathsf{honest}$ **do**
  return 1 **if** $\mathsf{pk}_i \neq \mathsf{pk}_j$
  return 1 **if** $\mathsf{qual}_i \neq \mathsf{qual}_j$
$\mathsf{sk}_0 \leftarrow \mathsf{D}[\mathsf{H}].\mathsf{Recover}(\mathsf{pk}_0, t, \{(i, \mathsf{sk}_i)\}_{i \in \mathsf{honest}})$
  // *All parties at this point have the same view of* pk *and* qual
return 1 **if** $\mathsf{sk}_0 = \mathsf{fail}$
  // *The output DKG keypair given to the distinguisher must be valid*
$(\mathsf{sk}_1, \mathsf{pk}_1) \xleftarrow{\$} \mathsf{TK}.\mathsf{KeyGen}(\lambda)$
$b' \leftarrow \mathcal{E}((\mathsf{pk}_b, \mathsf{sk}_b), (\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}))$
return 1 **if** $b' = b$
return 0

Fig. 10. Game defining the advantage of an adversary $\mathcal{A}$ to compromise indistinguishability of a DKG. The distinguisher $\mathcal{E}$ is challenged to distinguish between valid keypairs generated via the DKG and the target key generation algorithm. The oracles $\mathcal{O}^{\mathsf{PerformRound}_r}$ and RO are defined in Figure 8.

4.3 Indistinguishability

The security of any key generation protocol requires that the secret key be hard to guess or learn from the public transcript. In the single-party setting, the adversary does not have any influence over the sampling of key material, and so guaranteeing this property is straightforward. However, because the adversary is an active participant in a distributed key generation protocol, we must ensure that a subset of at most $t - 1$ colluding players cannot bias key material.

To establish this property, we require that a DKG must generate keys that are *indistinguishable* from keys output by the target key generation algorithm. We formalize this requirement in the attack game in Figure 10.

The advantage of a distinguishing adversary $\mathcal{E}$ against a DKG D in the indistinguishability experiment in Figure 10 with respect to the target key generation algorithm TK and participating adversary $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathrm{ind}}_{\mathsf{D},\mathsf{TK},\mathcal{A},\mathcal{E}}(\lambda) = \left|\Pr[\mathsf{Exp}^{\mathrm{ind}}_{\mathsf{D},\mathsf{TK},\mathcal{A},\mathcal{E}}(\lambda) = 1] - 1/2\right| .$$

Definition 15. *A DKG* D *is* **indistinguishable** *from its target key generation algorithm if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *and computationally unbounded distinguishers* $\mathcal{E}$, $\mathsf{Adv}^{\mathrm{ind}}_{\mathsf{D},\mathsf{TK},\mathcal{A},\mathcal{E}}(\lambda)$ *is negligible.*

The attack game in Figure 10 allows for two distinct adversaries, $\mathcal{A}$ and $\mathcal{E}$. $\mathcal{A}$ participates in the protocol and controls up to $t-1$ players. $\mathcal{E}$ is simply given two sets of keypairs after the protocol completes, and is required to guess which keypair was generated by the distributed key generation protocol and which by the target key generation protocol. Note that $\mathcal{A}$ and $\mathcal{E}$ are assumed to be black box and act arbitrarily, but are not allowed to share state. Otherwise, the game could trivially be won simply by $\mathcal{A}$ informing $\mathcal{E}$ which public key was output from the distributed key generation protocol. To similarly prevent this trivial win condition, the keypair output from the distributed key generation execution must be valid, otherwise $\mathcal{A}$ could force the protocol to output an invalid keypair, which $\mathcal{E}$ could trivially distinguish from the output of $\mathsf{TK.KeyGen}$.

5 A Generic DKG Construction

We now introduce a weakly robust generic construction that assumes an honest majority, shown in Figure 11. We present a concrete construction STORM of this generic construction that is secure in the discrete log setting in Section 6.

Our generic construction employs the following building blocks:

- A secure hash function H, where H_1 and H_2 are domain-separated instances of H, such that:
  - $\mathsf{H}_1 : (\{D_j\}_{j\in[n]}, \mathsf{aux}) \mapsto v$: Accepts as input a tuple, where the first element in the tuple is a set of n AgVSS commitments ordered canonically ordered, and the second element is an auxiliary element aux as output by H_2 below. The output is the tweak employed by the AgVSS aggregation algorithm.
  - $\mathsf{H}_2 : \{\psi_j\}_{j\in[n]} \mapsto \mathsf{aux}$: Accepts as input a set of n elements, and outputs an auxiliary element aux.
- An aggregatable verifiable secret sharing scheme $\mathsf{AV}[\mathsf{H}_1] = (\mathsf{Share}, \mathsf{Verify}, \mathsf{Recover}, \mathsf{GetPub}, \mathsf{AggPriv}, \mathsf{AggPub})$ that is zero-indexed.
- A non-interactive key exchange (NIKE) $\mathsf{NK} = (\mathsf{KeyGen}, \mathsf{Verify}, \mathsf{SharedKey})$.

The secret and public domains of the AgVSS and NIKE must be the same as that of the target key generation scheme, and the distribution over the secret domain for the NIKE must be the same as that of the target key generation scheme. Further, we assume that the NIKE outputs secret and public keys with strictly a one-to-one relation.

In addition to these building blocks, our construction requires that these primitives are algebraically composable. Intuitively, the secret key output from $\mathsf{NK.KeyGen}$ should be a valid secret that can be shared via $\mathsf{AV.Share}$, and the AgVSS commitment should be verifiable with respect to the public NIKE key. Towards this latter point, we require Assumption 1 to hold.

Assumption 1 *For any* $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda)$*, and any* $(n, t) \in \mathbb{N}$ *such that* $n \geq t$*; and any* $C \subseteq [n], |C| \geq t$*, Equation 6 holds:*

$$\begin{aligned} \mathsf{NK.Verify}(\mathsf{sk}, \mathsf{pk}) = 1, \ \textit{where} \\ (\{(i, w_i)\}_{i\in[n]}), D) \xleftarrow{\$} \mathsf{AV[H].Share}(\lambda, \mathsf{sk}, n, t) \ \textit{and} \\ \mathsf{pk} \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(0, D) \ \textit{and} \\ \mathsf{sk} \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{Recover}(t, \{(i, w_i)\}_{i\in C}) \end{aligned} \tag{6}$$

$\mathsf{PerformRound}_0[\mathsf{H}](\lambda, n, t, i)$

$(\alpha_i, A_i) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda)$
$\mathsf{in} \leftarrow (\lambda, \alpha_i, n, t)$
$(\{(j, w_{ij})\}_{j\in[n]}, D_i) \xleftarrow{\$} \mathsf{AV}[\mathsf{H}_1].\mathsf{Share}(\mathsf{in})$
// The NIKE secret key α_i is
// secret shared via the AgVSS.
$(\beta_i, B_i) \xleftarrow{\$} \mathsf{NK.KeyGen}(\lambda)$
// Broadcast commitments
$\mathsf{bmsg}_{0,i} \leftarrow (A_i, B_i, D_i)$
for $j \in [n], j \neq i$ **do**
// Send shares via peer-to-peer channel
$\mathsf{pmsg}_{0,ij} \leftarrow (j, w_{ij})$
$\mathsf{state}_i \leftarrow (\beta_i, w_{ii}, D_i, B_i)$
$\mathsf{outp} \leftarrow \{\mathsf{pmsg}_{0,ij}\}_{j\in[n],j\neq i}$
return $(\mathsf{state}_i, \mathsf{out}_p, \mathsf{bmsg}_{0,i})$

$\mathsf{PerformRound}_1[\mathsf{H}](\mathsf{state}_i, \mathsf{inp}_1, \mathsf{inb}_1)$

$\mathsf{parse}\ \{(i, w_{ji})\}_{j\in[n],j\neq i} \leftarrow \mathsf{inp}_1$
$\mathsf{parse}\ \{(A_j, B_j, D_j)\}_{j\in[n],j\neq i} \leftarrow \mathsf{inb}_1$
for $j \in [n], j \neq i$ **do**
if $\mathsf{AV}[\mathsf{H}_1].\mathsf{Verify}(i, w_{ji}, D_j) \neq 1$
or if $\mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(0, D_j) \neq A_j$
$\mathsf{state}_i.\mathsf{status} = \mathsf{abort}$
$\mathsf{bmsg}_{1,i} \leftarrow \mathsf{fail}$
// If any verification check fails, abort
return $(\mathsf{state}_i, \perp, \mathsf{bmsg}_{1,i})$
$\mathsf{state}_i \leftarrow \mathsf{state}_i \cup \{(j, w_{ji})\}_{j\in[n]}$
$\mathsf{bmsg}_{1,i} \leftarrow \mathsf{accept}$
return $(\mathsf{state}_i, \perp, \mathsf{bmsg}_{1,i})$

$\mathsf{PerformRound}_2[\mathsf{H}](\mathsf{state}_i, \emptyset, \mathsf{inb}_2)$

if $\mathsf{fail} \in \mathsf{inb}_2$
$\mathsf{state}_i.\mathsf{status} = \mathsf{abort}$
// If any participant fails, abort
return $(\mathsf{state}_i, \perp, \perp)$
$\mathsf{bmsg}_{2,i} \leftarrow \beta_i$
return $(\mathsf{state}_i, \perp, \mathsf{bmsg}_{2,i})$

$\mathsf{Finalize}[\mathsf{H}](\mathsf{state}_i, \emptyset, \mathsf{inb}_3)$

if $\mathsf{state}_i.\mathsf{status} = \mathsf{abort}$
return $(\perp, \perp, \perp), (\mathsf{abort}, \perp)$
$\mathsf{parse}\ \{\beta_j\}_{j\in[n],j\neq i} \leftarrow \mathsf{inb}_3$
$\mathsf{qual} \leftarrow \emptyset;\ \mathsf{corrupt} \leftarrow \emptyset$
for $j \in [n], j \neq i$ **do**
if $\mathsf{NK.Verify}(\beta_j, B_j) = 1$
$\psi_j \leftarrow \mathsf{NK.SharedKey}(\beta_j, A_j)$
// Qualified participants send
// the correct opening β_j
$\mathsf{qual} \leftarrow \mathsf{qual} \cup \{j\}$
else $\mathsf{corrupt} \leftarrow \mathsf{corrupt} \cup \{j\}$
for $j \in \mathsf{corrupt}$ **do**
// Perform secret recovery
// sub-protocol for
// misbehaving parties
$\mathsf{M}_j = \{(k, w_{jk})\}_{k\in\mathsf{qual}}$
// All remaining players
// combine their shares
$\alpha_j \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{Recover}(t, \mathsf{M}_j)$
$\psi_j \leftarrow \mathsf{NK.SharedKey}(\alpha_j, B_j)$
$v \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{GetTweak}(\{D_j\}_{j\in[n]}, \{\psi_j\}_{j\in[n]})$
$\mathsf{sk}_i \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{AggPriv}(i, \{w_{ji}\}_{j\in[n]}, v)$
$C \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{AggPub}(\{D_j\}_{j\in[n]}, v)$
$\mathsf{pk} \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(0, C)$
// Derive public keys for all
// remaining participants
for $i \in \mathsf{qual}$ **do**
$\mathsf{pk}_i = \mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(i, C)$
$\mathsf{aux} \leftarrow \{\mathsf{pk}_i\}_{i\in\mathsf{qual}}$
return $(\mathsf{pk}, \mathsf{qual}, \mathsf{aux}), (\mathsf{accept}, \mathsf{sk}_i)$

Fig. 11. A generic DKG key generation protocol as defined in Section 4, initialized with a NIKE NK as defined in Section 2.2, an AgVSS AV as defined in Section 3.2, and a hash function H. Each $\mathsf{PerformRound}_r$ operation is defined with respect to participant i. The peer-to-peer channel inputs for round r are inp_r and outputs are outp_r, and are comprised of peer-to-peer messages pmsg_j. The broadcast channel inputs are inb_r and are comprised of broadcast messages bmsg_j, sent by participant $j, j \in [n]$.

Key Generation Protocol. As shown in Figure 11, the key generation protocol is completed in three network rounds.

First, in $\mathsf{PerformRound}_0$, each participant i performs $\mathsf{NK.KeyGen}$ twice; first to generate a secret key α_i and its commitment A_i, and second to generate a random value β_i and its commitment B_i. Then, each participant i performs an AgVSS secret sharing of α_i, outputting n secret shares $\{(j, w_{ij})\}_{j\in[n]}$, and an AgVSS commitment D_i. Finally, each participant i broadcasts its NIKE and AgVSS commitments to all other players via a broadcast message $\mathsf{bmsg}_{0,i}$. Additionally, each participant i sends one secret share to every other player $j \neq i$ via a peer-to-peer message $\mathsf{pmsg}_{0,ij}$, keeping one share for itself.

In $\mathsf{PerformRound}_1$, each participant i receives shares $\{(i, w_{ji})\}_{j\in[n],j\neq i}$ from all other players via the peer-to-peer channel input inp_1 and commitments $\{(A_j, B_j, D_j)\}_{j\in[n],j\neq i}$ from all other players via the broadcast channel input inb_1. Each participant then verifies the correctness of their received share with respect to its corresponding commitment. Additionally, each participant i verifies that the secret committed to in D_i is the same secret committed to in A_i, by checking that $\mathsf{AV.GetPub}(0, D_i) = A_i$. If any check fails, the participant will broadcast a fail message, and set their status to abort. Any participant that sets their status to abort results in that participant terminating (immediately exiting) the protocol. Else, it will broadcast an accept message.

In $\mathsf{PerformRound}_2$, participants receive as input status messages from all other participants via the broadcast channel input inb_2. If any participant received a fail message, that participant immediately exits $\mathsf{PerformRound}_2$ after setting their status to abort[1]. Otherwise, each participant i broadcasts β_i to all other participants (which is the opening to the commitment B_i sent in $\mathsf{PerformRound}_0$).

Finally, in $\mathsf{Finalize}$, each participant i receives as input the openings $\{\beta_j\}_{j\in[n],j\neq i}$ from all other participants via the broadcast channel inb_3. For the values they received from every other player $j \neq i$, they first verify the correctness of the opening β_j with respect to B_j. Depending on the output of this check, each participant i does the following:

- If the opening β_j is valid, participant i then derives the blinding factor ψ_j by performing $\psi_j \leftarrow \mathsf{NK.SharedKey}(\beta_j, A_j)$.
- If the opening β_j is invalid, the set of participants qual that followed the protocol honestly cooperate to derive the correct ψ_j, excluding the cheating participant. They do so by pooling their respective shares in order to recover α_j. We discuss this secret recovery sub-protocol further below. After obtaining this set qual of at least t shares, each remaining participant then performs $\mathsf{NK.SharedKey}(\alpha_j, B_j)$, obtaining ψ_j as the result. Due to the assumption that at least t honest participants exist, this step will always complete successfully.

Next, each participant derives $v \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{GetTweak}(O, \{\psi_1, \ldots, \psi_n\})$, where $O = \{D_j\}_{j\in[n]}$ is the set of all AgVSS commitments for each participant. Each participant i then performs $\mathsf{AV.AggPriv}$ and $\mathsf{AV.AggPub}$, using as input the set of all their received shares $P = \{w_{ji}\}_{j\in[n]}$, the set of AgVSS commitments O, and v. Each participant i employs the output of $\mathsf{AV.AggPriv}$ and $\mathsf{AV.AggPub}$ to determine their final secret key share sk_i and the aggregated commitment C. The group public key pk is then derived using C. Similarly, the set of each participant's individual public key pk_i is derived using C, and then output as the auxiliary data.

Participants finally complete $\mathsf{Finalize}$ by setting their status to accept, updating their internal state, and exiting the protocol. The public output at the end of $\mathsf{PerformRound}_3$ is $(\mathsf{pk}, \mathsf{qual}, \mathsf{aux})$. The private output to each participant i is $(\mathsf{status}, \mathsf{sk}_i)$.

Secret Recovery Sub-Protocol. When performing the $\mathsf{Finalize}$ step in Figure 11, in case any party misbehaves and sends an incorrect opening β_j, all remaining honest participants broadcast the shares sent by the malicious party j in order to recover the secret s_j. We rely on existing definitions of this sub-protocol in prior related DKG constructions; see Gennaro et al. [28] for details of this sub-protocol, which is only performed in the event of a misbehaving party.

[1] Because the fail message is sent on a broadcast channel, if any participant receives a fail message, then all participants receive that message

Recovery Algorithm. While most practical DKG applications do not require the participants to actually come together to recover the secret key, it is possible that some applications may wish to do so. We define the recovery algorithm now.

– $\mathsf{Recover}(\mathsf{pk}, t, \{(i, \mathsf{sk}_i)\}_{i \in C}) \to \mathsf{sk}/\mathsf{fail}$: Accepts the public key $\mathsf{pk} \in \mathbb{G}$ and $|C| \geq t$ tuples $\{(i, \mathsf{sk}_i)\}_{i \in C}$ consisting of participant i's identifier and their secret key. Using these inputs, perform the following steps:

1. Derive $\mathsf{sk} = \mathsf{AV.Recover}(t, \{(i, \mathsf{sk}_i)\}_{i \in C})$.
2. If $\mathsf{TK.Verify}(\mathsf{sk}, \mathsf{pk}) \neq 1$, output fail.
3. Output sk.

5.1 Security of the Generic Protocol

We next demonstrate that the generic protocol in Figure 11 is weakly robust, zero-knowledge, and indistinguishable.

Theorem 1. *The generic construction is weakly robust in the honest majority setting against an adversary $\mathcal{A}$ controlling up to $t-1$ participants in the random oracle model, a zero-indexed and secure AgVSS, a binding NIKE.*

Concretely, for every adversary $\mathcal{A}$ that attacks the scheme, there exist adversaries $(\mathcal{B}_1, \mathcal{B}_2)$ that run in about the same time as $\mathcal{A}$ such that

$$\mathsf{Adv}^{\text{wk-rbst}}_{\mathsf{D},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{bind}}_{\mathsf{NK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{unq}}_{\mathsf{AV}[\mathsf{H}_1],\mathcal{B}_2}(\lambda). \tag{7}$$

We prove Theorem 1 in Appendix D.1.

Theorem 2. *The generic construction is zero-knowledge in the honest majority setting in the random oracle model, against a PPT adversary $\mathcal{A}$ controlling up to $t-1$ participants, assuming the generic construction itself is weakly robust, the AgVSS achieves aggregated secrecy, and the NIKE is unrecoverable.*

Concretely, for every adversary $\mathcal{A}$ that attacks the construction, there exist adversaries $(\mathcal{B}_1, \mathcal{B}_3, \mathcal{B}_4)$ that run in about the same time as $\mathcal{A}$, and algorithms $\mathsf{SimRound}_i, i \in \{0, \ldots, 2\}, \mathsf{SimFinalize}$, such that

$$\begin{aligned}\mathsf{Adv}^{\text{zk}}_{\mathsf{D},\mathcal{A},\mathsf{SimRound}_i,\mathsf{SimFinalize}}(\lambda) \leq\ & \mathsf{Adv}^{\text{sec}}_{\mathsf{AV}[\mathsf{H}_1],\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{rec}}_{\mathsf{NK},\mathcal{B}_3}(\lambda) \\ & + \mathsf{Adv}^{\text{asec}}_{\mathsf{AV}[\mathsf{H}_1],\mathcal{B}_4,\mathsf{SimShare},\mathsf{SimTweak}}(\lambda).\end{aligned} \tag{8}$$

where $\mathsf{SimShare}, \mathsf{SimTweak}$ are as defined for aggregated secrecy of the AgVSS, as in Definition 9.

We prove Theorem 2 in Appendix D.2.

Theorem 3. *The generic construction is indistinguishable in the honest majority setting in the random oracle model, against a PPT adversary $\mathcal{A}$ controlling up to $t-1$ participants, assuming the generic construction itself is weakly robust, the AgVSS achieves aggregated secrecy, and the NIKE is unrecoverable.*

Concretely, for every adversary $\mathcal{A}$ that attacks the construction, there exist adversaries $(\mathcal{B}_1, \mathcal{B}_3, \mathcal{B}_4)$ that run in about the same time as $\mathcal{A}$, such that

$$\begin{aligned}\mathsf{Adv}^{\text{ind}}_{\mathsf{D},\mathsf{TK},\mathcal{A},\mathcal{E}}(\lambda) \leq\ & \mathsf{Adv}^{\text{sec}}_{\mathsf{AV}[\mathsf{H}_1],\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{rec}}_{\mathsf{NK},\mathcal{B}_3}(\lambda) \\ & + \mathsf{Adv}^{\text{asec}}_{\mathsf{AV}[\mathsf{H}_1],\mathcal{B}_4,\mathsf{SimShare},\mathsf{SimTweak}}(\lambda).\end{aligned} \tag{9}$$

where $\mathsf{SimShare}, \mathsf{SimTweak}$ are as defined for aggregated secrecy of the AgVSS, as in Definition 9.

Because our generic construction requires a one-to-one mapping between secret and public keys, Theorem 3 follows from Theorem 2.

6 STORM, a Concrete DKG Construction

Our generic DKG presented in Section 5 must be instantiated with concrete building blocks to be usable. We now present one possible concrete construction, which we call STORM (Synchronous, disTributed, and Optimized geneRation of key Material). We refer to Section 5 for the details of the DKG key generation and recovery steps, but describe the building blocks and security assumptions for STORM here.

STORM is secure under the Computational Diffie-Hellman (CDH) problem in the random oracle model, in the honest majority setting. STORM can be securely composed with any non-pairing based scheme that relies on these assumptions, as the distributed form of the following target key generation scheme.

Target Key Generation. The target key generation scheme TK for STORM is defined as follows. The secret domain $\hat{S}$ is the field $\mathbb{Z}_q$ and the public domain $\hat{P}$ is the group $\mathbb{G}$ generated by g.

- $\mathsf{KeyGen}(\lambda) \to (\mathsf{sk}, \mathsf{pk})$: Sample $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$. Generate $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. Output $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Verify}(\mathsf{sk}, \mathsf{pk}) \to \{0, 1\}$: Output 1 if $g^{\mathsf{sk}} = \mathsf{pk}$. Otherwise, output 0.

Building Blocks. STORM is instantiated with the following cryptographic building blocks:

1. the concrete NIKE scheme presented in Section 2.2,
2. the concrete AgVSS scheme presented in Section 3.2, and
3. a secure hash function H, where H_1, and H_2 domain-separated instances of H, defined as follows:
   - $\mathsf{H}_1 : (\mathbb{G}^t)^n \times \mathbb{Z}_q \to \mathbb{Z}_q$: Accepts as input a vector, where the first element is a tuple of n AgVSS commitments $D_j \in \mathbb{G}^t, j \in [n]$ which are each a tuple of group elements of size t, and the second element is an auxiliary element $\mathsf{aux} \in \mathbb{Z}_q$.
   - $\mathsf{H}_2 : (\mathbb{G})^n \to \mathbb{Z}_q$: Accepts as input a vector of n blinding factors such that $\psi_j \in \mathbb{G}, j \in [n]$, and outputs an auxiliary element $\mathsf{aux} \in \mathbb{Z}_q$.

Correctness and Security. Because STORM is an instantiation of the generic construction presented in Section 5, its correctness and security automatically follow from the proofs for the generic scheme.

7 Conclusion

In this work, we present a generic construction for a DKG that can be securely employed in any setting in place of its target (single-party) key generation scheme. To prove its security, we require that the DKG be strongly or weakly robust, zero-knowledge, and indistinguishable from its target key generation scheme. We formalize these notions using a game-based approach, and use these notions to prove the security of our generic construction. We then introduce STORM, a concrete instantiation of our generic construction that is secure assuming the Computational Diffie-Hellman problem is hard, in the random oracle model.

8 Acknowledgments

We thank Dan Shumow, Greg Zaverucha, and Chris Wood for their helpful discussion, and Jack Grigg and Deirdre Connolly for their review and feedback. Many thanks to Olivier Sanders and Jonathan Katz for their discussion on simulation-based notions of security. Chelsea Komlo was supported in part by Microsoft Research in the process of completing this work. Douglas Stebila was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2022-0318. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program.

References

1. I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2022/1759, 2022. URL: https://eprint.iacr.org/2022/1759.
2. I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation. In A. Miller, K. Censor-Hillel, and J. H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing*, pages 363–373. ACM, 2021.
3. R. Bacho and J. Loss. On the adaptive security of the threshold BLS signature scheme. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *CCS 2022*, pages 193–207. ACM, 2022.
4. R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. Hybrid public key encryption. https://datatracker.ietf.org/doc/html/rfc9180, 2022.
5. M. Bellare, E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. Better than advertised security for non-interactive threshold signatures. In Y. Dodis and T. Shrimpton, editors, *CRYPTO 2022*, volume 13510 of *LNCS*, pages 517–550. Springer, 2022.
6. R. Bendlin, S. Krehbiel, and C. Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In M. J. J. Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *Applied Cryptography and Network Security (ACNS) 2013*, volume 7954 of *LNCS*, pages 218–236. Springer, 2013.
7. D. J. Bernstein. Curve25519: New Diffie–Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006.
8. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.
9. D. Boneh and V. Shoup. A graduate course in applied cryptography, 2023. URL: http://toc.cryptobook.us/book.pdf.
10. L. Brandão and M. Davidson. Notes on threshold eddsa/schnorr signatures. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8214B.ipd.pdf, 2022.
11. L. Brandão and R. Peralta. Nist first call for multi-party threshold schemes. https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8214C.ipd.pdf, 2023.
12. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.
13. R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1769–1787. ACM, 2020.
14. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In M. J. Wiener, editor, *CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999*, volume 1666 of *LNCS*, pages 98–115. Springer, 1999.
15. J. F. Canny and S. Sorkin. Practical large-scale distributed key generation. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2004.
16. D. Cash, E. Kiltz, and V. Shoup. The twin Diffie–Hellman problem and applications. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 127–145. Springer, 2008.
17. W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: An efficient post-quantum commutative group action. Cryptology ePrint Archive, Paper 2018/383, 2018. URL: https://eprint.iacr.org/2018/383.
18. W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: an efficient post-quantum commutative group action. In T. Peyrin and S. D. Galbraith, editors, *ASIACRYPT 2018*, volume 11274 of *LNCS*, pages 395–427. Springer, 2018.
19. D. Connolly, C. Komlo, I. Goldberg, and C. Wood. Two-round threshold Schnorr signatures with FROST, 2022. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost/.
20. I. Damgård and G. L. Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. In D. Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2010.
21. S. Das, T. Yurek, Z. Xiang, A. K. Miller, L. Kokoris-Kogias, and L. Ren. Practical asynchronous distributed key generation. In *43rd IEEE Symposium on Security and Privacy, SP 2022*, pages 2518–2534. IEEE, 2022.
22. A. Davidson, S. Sahib, P. Snyder, and C. Wood. Star: Distributed secret sharing for private threshold aggregation reporting, 2022. URL: https://datatracker.ietf.org/doc/draft-dss-star.

23. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437. IEEE Computer Society, 1987.
24. M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, 2005.
25. Y. Frankel, P. D. MacKenzie, and M. Yung. Robust efficient distributed rsa-key generation. In J. S. Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 663–672. ACM, 1998.
26. E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In K. Kurosawa and G. Hanaoka, editors, *Public-Key Cryptography - PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, 2013.
27. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure applications of Pedersen's distributed key generation protocol. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 373–390. Springer, 2003.
28. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, 2007.
29. J. Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339, 2021. URL: https://eprint.iacr.org/2021/339.
30. K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Aggregatable distributed key generation. In A. Canteaut and F. Standaert, editors, *EUROCRYPT 2021*, volume 12696 of *LNCS*, pages 147–176. Springer, 2021.
31. A. Kate, Y. Huang, and I. Goldberg. Distributed key generation in the wild. Cryptology ePrint Archive, Paper 2012/377, 2012. URL: https://eprint.iacr.org/2012/377.
32. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.
33. E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20*, pages 1751–1767. ACM, 2020.
34. C. Komlo and I. Goldberg. FROST: flexible round-optimized Schnorr threshold signatures. In O. Dunkelman, M. J. J. Jr., and C. O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, 2020.
35. E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.
36. League of Entropy. Threshold partially-oblivious PRFs with applications to key management. https://docs.keep.network/random-beacon/dkg.html, 2019.
37. Y. Lindell. Simple three-round multiparty Schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://ia.cr/2022/374.
38. Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1837–1854. ACM, 2018.
39. W. Neji, K. B. Sinaoui, and N. B. Rajeb. Distributed key generation protocol with a new complaint management strategy. *Secur. Commun. Networks*, 9(17):4585–4595, 2016.
40. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.
41. T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In D. W. Davies, editor, *EUROCRYPT 1991*, volume 547 of *LNCS*, pages 522–526. Springer, 1991.
42. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

$\mathsf{SimShare}(\Delta, n, t, \mathsf{corrupt})$

1 : $A_0 = \Delta$
2 : **for** $j \in \mathsf{corrupt}$ **do**
3 : $\quad w_j \xleftarrow{\$} \mathbb{Z}_q$
4 : **for** $k \in \{1, \ldots, t-1\}$ **do**
5 : $\quad A_k = A_0^{L'_{0,k}} \cdot \prod_{j \in \mathsf{corrupt}} g^{L'_{j,k} \cdot w_j}$
6 : $D = \langle A_0, \ldots, A_{(t-1)} \rangle$
7 : **return** $(\{(j, w_j)\}_{j \in \mathsf{corrupt}}, D)$

Fig. 12. Algorithm to simulate secret sharing in a Feldman VSS to an adversary controlling $t-1$ participants denoted by the set $\mathsf{corrupt}$ with respect to a challenge Δ. Here, $L'_{i,k}$ denotes the k^{th} coefficient of the Lagrange polynomial $L_i(x)'$, which is computed with respect to the set $\{0\} \cup \mathsf{corrupt}$.

A Proofs for Concrete NIKE

In Section 2.2, we introduce a concrete NIKE scheme. We now demonstrate that it is unrecoverable and binding.

Theorem 4. *The concrete NIKE of Section 2.2 is unrecoverable against a probabilistic polynomial-time adversary $\mathcal{A}$, assuming the Computational Diffie–Hellman (CDH) assumption holds in $\mathbb{G}$. More specifically, for any PPT adversary $\mathcal{A}$ that attacks NK, there exists an adversary $\mathcal{B}$ with approximately the same running time as $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{rec}}_{\mathsf{NK},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{cdh}}_{\mathcal{B}}(\lambda)$$

Because breaking session key recovery in our concrete scheme simply requires an adversary to guess a valid Diffie–Hellman shared secret knowing only two public keys, we omit the proof for Theorem 4 as it is a direct reduction to CDH.

Theorem 5. *The concrete NIKE of Section 2.2 is binding against a computationally unbounded adversary $\mathcal{A}$.*

Proof. We now show that any adversary $\mathcal{A}$ cannot win the binding experiment $\mathsf{Exp}^{\text{bind}}$ in Figure 2 for the concrete NIKE of Section 2.2. $\mathcal{A}$ wins when it can produce some $(\mathsf{sk}_1^*, \mathsf{pk}_1^*, \mathsf{sk}_2^*, \mathsf{pk}_2^*)$ such that:

1. $\mathsf{NK.Verify}(\mathsf{sk}_1^*, \mathsf{pk}_1^*)$ outputs 1, and
2. $\mathsf{NK.Verify}(\mathsf{sk}_2^*, \mathsf{pk}_2^*)$ outputs 1, but
3. $\mathsf{NK.SharedKey}(\mathsf{sk}_1^*, \mathsf{pk}_2^*) \neq \mathsf{NK.SharedKey}(\mathsf{sk}_2^*, \mathsf{pk}_1^*)$

However, in the setting of the concrete scheme, this means that $\mathsf{pk}_1^* = g^{\mathsf{sk}_1^*}$ and $\mathsf{pk}_2^* = g^{\mathsf{sk}_2^*}$, yet $(\mathsf{pk}_2^*)^{\mathsf{sk}_1^*} \neq (\mathsf{pk}_1^*)^{\mathsf{sk}_2^*}$, which is impossible.

B Proofs for Feldman's VSS

In Section 3.1, we define Feldman's VSS. We now discuss its security.

B.1 Secrecy

Theorem 6. *Feldman's VSS is perfectly secret.*

Proof. We prove Theorem 6 by showing that the adversary cannot distinguish Game 0 (which corresponds to the VSS secrecy experiment from Figure 3 when $b = 0$ and the protocol is honestly executed) from Game 1 (Figure 3 when $b = 1$ and the protocol is simulated).

Game 0. This is simply the secrecy experiment as shown in Figure 3, instantiated with Feldman's VSS, and with $b = 0$.

Game 1. This is the game shown in Figure 3, when $b = 1$. Recall that when $b = 1$, the environment simulates secret sharing for $\Delta \in \mathbb{G}$ whose discrete logarithm is unknown, employing an algorithm $\mathsf{SimShare}$.

We show $\mathsf{SimShare}$ for our concrete construction in Figure 12. To begin, the algorithm picks $t-1$ shares w_j at random. This number of shares is sufficient for the simulation, as we assume $|\mathsf{corrupt}| = t-1$, without loss of generality. $\mathsf{SimShare}$ then performs polynomial interpolation "in the exponent" using Δ and the $t-1$ shares, to derive $A_k, k \in [t-1]$. The commitment to f is then set as $D = \langle A_0, \ldots, A_{t-1} \rangle$, where $A_0 = \Delta$.

Difference between Game 0 *and Game* 1. We now show that $\mathcal{A}$ has no additional advantage in distinguishing the games.

The simulation is perfect because D in Game 1 simply defines a random polynomial f' "in the exponent", whose constant term $f'(0)$ (the secret) is the discrete logarithm of Δ. Hence, the output of $\mathsf{SimShare}$ is indistinguishable to the adversary from a real output of $\mathsf{S.Share}$.

Finishing the proof. Because $\mathcal{A}$ has identical advantage in Game 0 and Game 1 (i.e., the simulation is perfect), the advantage that $\mathcal{A}$ wins the VSS secrecy game is then zero. This concludes the proof. □

B.2 Uniqueness

We now give a proof sketch for the uniqueness of Feldman's VSS; we omit the full proof as it is a well-known result in the literature [32].

Feldman's VSS is information-theoretically unique because Shamir's secret sharing is information-theoretically unique. Put differently, a polynomial f is uniquely defined by t points. Hence, given some threshold $t^* \geq 1$ chosen by the adversary playing against $\mathsf{Exp}^{\mathrm{unq}}$ in Figure 4, the adversary cannot pick two different recovery sets $\mathsf{M}_1^*, \mathsf{M}_2^*$ and a single D^* where each share in M_1^* and M_2^* is valid with respect to t^* and D^*, but $\mathsf{S.Recover}(t^*, \mathsf{M}_1^*) \neq \mathsf{S.Recover}(t^*, \mathsf{M}_2^*)$.

C Proofs for the Concrete AgVSS

In Section 3.2, we define a concrete AgVSS which builds upon Feldman's VSS. We demonstrate the correctness, aggregated secrecy, and uniqueness of that scheme now.

C.1 Correctness

The concrete AgVSS scheme of Section 3.2 fulfills the notion of correctness for the base VSS scheme, as the algorithms $\mathsf{Share}, \mathsf{Recover}, \mathsf{Verify}, \mathsf{GetPub}$ are identical to Feldman's VSS, as is the map $\mathsf{SecretToPublic}$. We now demonstrate that the concrete AgVSS scheme fulfills the additional notion of correctness for an AgVSS scheme.

Theorem 7. *The concrete AgVSS scheme of Section 3.2 is correct in the sense of Figure 5.*

Proof. We analyze each case where a return value of 0 could occur in Figure 5, and demonstrate that that event will not occur.

Case 1. We first show that for each $i \in [n]$, $\mathsf{AV.Verify}(i, \hat{w}_i, C) = 1$. Recall that $\hat{w}_i$ is simply

$$\hat{w}_i = v + \sum_{j \in [\ell]} w_{ji}$$

by Equation 3. Because each w_{ji} was generated by the $\mathsf{AV.Share}$ algorithm, then $w_{ji} = f_j(i)$ for some polynomials $f_1, \ldots, f_\ell$ each of degree $t-1$. Let $f' = v + \sum_{i\in[\ell]} f_i$. Then $(i, \hat{w}_i)$ is a valid point on f', because

$$\hat{w}_i = v + \sum_{j\in[\ell]} w_{ji} = v + \sum_{j\in[\ell]} f_j(i) = f'(i)$$

Because C is a commitment to the same f' as shown in Equations 4 and 5, then $\mathsf{AV.Verify}(i, \hat{w}_i, C)$ will output 1. Note also that $\mathsf{AV.Recover}$ will output $\hat{s} = f'(0) = v + \sum_{j\in[\ell]} s_j$.

Case 2. We next prove that $\mathsf{AV.Verify}(0, \hat{s}, C) = 1$. As above, after performing $\mathsf{AV.AggPub}$, the resulting aggregated commitment C can be represented as a commitment to the polynomial f'. $\mathsf{AV.Verify}(0, \hat{s}, C)$ checks that $g^{\hat{s}} = \hat{A}_0$ (from Eq. 2 with $i = 0$). From Eq. 4, $\hat{A}_0 = g^v \cdot \prod_{j\in[\ell]} D_j[0] = g^v \cdot \prod_{j\in[\ell]} g^{f_j(0)} = g^v \cdot \prod_{j\in[\ell]} g^{s_j}$. Since $\hat{s} = v + \sum_{j\in[\ell]} s_j$ as above, $\mathsf{AV.Verify}(0, \hat{s}, C)$ will output 1.

This concludes the proof. □

C.2 Aggregated Secrecy

We now show that the concrete scheme of Section 3.2 fulfills the notion of aggregated secrecy from Figure 6. To do so, we first show how the environment simulates $\mathsf{SimShare}$ and H. Without loss of generality, we assume $|\mathsf{corrupt}| = t - 1$.

Lemma 1. *When playing the secure aggregation game as in Figure 6 against the concrete scheme, the output of $\mathcal{O}^{\mathsf{GetShare}}$ when $b = 0$ (i.e., when the environment honestly performs $\mathsf{AV.Share}$) is indistinguishable to an adversary from the output when $b = 1$ (i.e., when the environment performs $\mathsf{SimShare}$ as shown in Figure 13), assuming the adversary controls no more than $|\mathsf{corrupt}| = t - 1$ participants.*

Proof. $\mathsf{SimShare}$ accepts a challenge $\Delta \in \mathbb{G}$, and outputs a randomly sampled secret α, $|\mathsf{corrupt}|$ shares, and an AgVSS commitment. To begin, it picks $\alpha \in \mathbb{Z}_q$ at random, and then generates a blinded commitment A_0 to the (unknown) discrete logarithm of Δ using α as the blinding factor, by Equation 10.

$$A_0 = \Delta \cdot g^\alpha \tag{10}$$

$\mathsf{SimShare}$ then picks $t-1$ shares for the corrupted parties at random, resulting in the list of shares $\langle w_j \rangle_{j\in\mathsf{corrupt}} \xleftarrow{\$} \mathbb{Z}_q^{(t-1)}$ and then generates commitments to coefficients $A_1, \ldots, A_{(t-1)}$ with respect to these shares and A_0, by Equation 11.

$$A_k = A_0^{L'_{0,k}} \cdot \prod_{j\in\mathsf{corrupt}} g^{L'_{j,k}\cdot w_j} \tag{11}$$

Here, $L'_{i,k}$ denotes the k^{th} coefficient of the Lagrange polynomial $L'_i(x)$, which is computed with respect to the set $\{0\} \cup \mathsf{corrupt}$. The commitment D to the $|\mathsf{corrupt}|$ shares is then $D = \langle A_0, \ldots, A_{(t-1)} \rangle$. The simulation is perfect because D simply is committing to a random polynomial f "in the exponent", whose constant term is the discrete logarithm of A_0, (where A_0 is a blinded commitment to the discrete logarithm of Δ) and with $t-1$ random points that are the shares distributed to the adversary. Each additional coefficient $a_k, k \in [t-1]$ of f is committed to "in the exponent" via A_k, which can be determined via Equation 11. Hence, so long as the adversary controls fewer than $t-1$ players, $\mathsf{AV.Verify}(i, w_i, D)$ for all $i \in \mathsf{corrupt}$ will output 1. □

Lemma 2. *The adversary has negligible advantage in distinguishing the environment's simulation of $\mathsf{GetTweak}$ in the concrete scheme of Section 3.2 in the (programmable) random oracle model, assuming an honest majority.*

Proof. Recall that $\mathsf{GetTweak}$ is instantiated by H in the concrete scheme. For any query (O', aux') where $\mathsf{aux}_i \neq \mathsf{aux}$, the environment simply programs H to return a random value. However, when it receives a query for (O, aux), it programs H to return v as in Equation 12. Recall that aux is provided to the adversary

$\mathsf{SimShare}(\lambda, \mathsf{corrupt}, \Delta)$

1 : $\alpha \xleftarrow{\$} \mathbb{Z}_q;\ A_0 = \Delta \cdot g^{\alpha}$
2 : **for** $j \in \mathsf{corrupt}$ **do**
3 : $\quad w_j \xleftarrow{\$} \mathbb{Z}_q$
4 : **for** $k \in \{1, \ldots, t-1\}$ **do**
5 : $\quad A_k = A_0^{L'_{0,k}} \cdot \prod_{j \in \mathsf{corrupt}} g^{L'_{j,k} \cdot w_j}$
6 : $D = \langle A_0, \ldots, A_{(t-1)} \rangle$
7 : **return** $(\alpha, \{(j, w_j)\}_{j \in \mathsf{corrupt}}, D)$

$\mathsf{H}(O_i, \mathsf{aux}_i)$

1 : $\quad$ // Q_1, Q_2, Q_3 *are defined*
2 : $\quad$ // *by the experiment*
3 : **if** $Q_3[O_i, \mathsf{aux}_i] \neq \perp$
4 : $\quad$ **return** $Q_3[O_i, \mathsf{aux}_i]$
5 : $D^* \leftarrow Q_1$
6 : $\{D_j\}_{j \in [\ell']} \leftarrow Q_2$
7 : $O = \{D_j\}_{j \in [\ell']} \cup D^*$
8 : **if** $\mathsf{aux}_i \neq \mathsf{aux}$ **or** $O_i \neq O$
9 : $\quad$ // *Respond honestly if this is*
10 : $\quad$ // *not the challenge query*
11 : $\quad Q_3[O_i, \mathsf{aux}_i] \xleftarrow{\$} \mathbb{Z}_q$
12 : $\quad$ **return** $Q_3[O_i, \mathsf{aux}_i]$
13 : $\alpha \leftarrow Q_1$
14 : **for** $\{(j, w_{kj})\}_{j \in \mathsf{honest}, k \in [\ell']} \in Q_2$
15 : $\quad \mathsf{in} \leftarrow (t, \{(j, w_{kj})\}_{j \in \mathsf{honest}})$
16 : $\quad s_k \leftarrow \mathsf{AV}[\mathsf{H}].\mathsf{Recover}(\mathsf{in})$
17 : $v \leftarrow (-1) \cdot (\alpha + \sum_{i \in [j]} s_k)$
18 : $Q_3[O_i, \mathsf{aux}_i] \leftarrow v$
19 : **return** v

Fig. 13. Algorithms to simulate secret sharing and tweak generation in an AgVSS to an adversary controlling $t-1$ participants denoted by the set $\mathsf{corrupt}$ with respect to a challenge Δ and at least t honest players denoted by the set honest, where $\mathsf{corrupt} \cup \mathsf{honest} = [n]$ and $\mathsf{corrupt} \cap \mathsf{honest} = \emptyset$. Here, $L'_{i,k}$ denotes the k^{th} coefficient of the Lagrange polynomial $L'_i(x)$, which is computed with respect to the set $\{0\} \cup \mathsf{corrupt}$.

only after it has finished querying $\mathcal{O}^{\mathsf{GetShare}}$ and $\mathcal{O}^{\mathsf{RecvShare}}$, and so similarly O is fixed at the time that the adversary can query with aux as input (i.e, the environment will never have to guess which query to program).

$$v \leftarrow -\alpha - \sum_{j \in [\ell']} s_j \tag{12}$$

There are three requirements that must be satisfied for the environment to program H correctly.

1. The programmed output v must be indistinguishable from a random value.
2. The environment must be able to derive $\{s_j\}_{j \in [\ell']}$ without the involvement or detection of $\mathcal{A}$.
3. The environment can correctly program H strictly before $\mathcal{A}$ can query on inputs (O, aux).

We first demonstrate that the output v from Equation 12 is indistinguishable from any other output from H. Recall that α is chosen at random when $\mathsf{SimShare}$ is performed, and published only within the commitment A_0 as shown in Equation 10, serving as a blinding factor for Δ. Because Δ is chosen at uniformly at random from the public domain, and because α is chosen at random, the v value in Equation 12 is indistinguishable from a random element to $\mathcal{A}$. Moreover, even if the adversary submitted inputs to $\mathcal{O}^{\mathsf{RecvShare}}$ in the attempt to cancel out the outputs from $\mathcal{O}^{\mathsf{GetShare}}$, the adversary would be unable to do so, because the adversary sees only the blinded commitment A_0 as shown in Equation 10, and because Δ is chosen uniformly at random. Therefore, programming H at exactly one point with v is indistinguishable to any other point that is honestly programmed with a random value chosen from $\mathbb{Z}_q$.

Second, the environment can derive $\{s_j\}_{j\in[\ell']}$ without the involvement or detection of $\mathcal{A}$, because we operate in the honest majority setting, and assume that the environment simulates at least t honest players. The environment can therefore reconstruct the adversary's s_j values (from Q_2), and it knows its own (from both Q_1 and Q_2).

Finally, the adversary does not learn aux until *after* it has completed its queries to $\mathcal{O}^{\mathsf{GetShare}}$ and $\mathcal{O}^{\mathsf{RecvShare}}$. Before returning aux to $\mathcal{A}$, the environment programs H with v on inputs (O, aux). Because $\mathsf{AX} = \mathbb{Z}_q$ and aux is some uniform element from AX, then $\mathcal{A}$ had negligible probability in learning the output H with input aux until the environment has programmed it. The environment is able to correctly program on these inputs using v because it is guaranteed that $\mathcal{A}$ cannot further influence O. In other words, there is no danger that the environment must make a decision on which input to program using v.

This completes the proof. □

Theorem 8. *The concrete scheme of Section 3.2 fulfills aggregated secrecy against an adversary $\mathcal{A}$ as in Figure 6, assuming an honest majority.*

Proof. Recall that that in the aggregated secrecy game, $\mathcal{A}$ has three opportunities to distinguish the $b = 0$ game from the $b = 1$ game:

1. by how the environment responds to $\mathcal{O}^{\mathsf{GetShare}}$,
2. by how the environment responds to $\mathcal{O}^{\mathsf{GetTweak}}$, and
3. whether or not $\mathsf{AV.GetPub}(0, C)$ evaluates to the challenge Δ when $b = 1$.

For (1), by Lemma 1, $\mathcal{O}^{\mathsf{GetShare}}$ is indistinguishable to the adversary when $b = 0$ (i.e., shares are honestly generated) and when $b = 1$ (i.e., $\mathsf{SimShare}$ is performed), For (2), by Lemma 2, the environment can simulate H in a way that is indistinguishable to the adversary in the $b = 0$ case (the output is chosen at random) and the $b = 1$ case (the output is programmed). We complete the proof by showing that the adversary has zero advantage for (3).

Recall that the concrete AgVSS is zero-indexed, per Definition 5. By Lemma 1, the environment can embed Δ in the first query of the adversary to $\mathcal{O}^{\mathsf{GetShare}}$ in a way that is indistinguishable to $\mathcal{A}$. $\mathcal{A}$ must query $\mathcal{O}^{\mathsf{GetShare}}$ at least once (i.e., if $\mathsf{issuedchal} = 0$ at the end of the experiment, then the experiment returns 0). Hence, it is guaranteed that the public recovery set $\{D_j\}_{j\in[\ell]}$ that is input to $\mathsf{AV.AggPub}$ will include a contribution with respect to the discrete logarithm of Δ.

Equation 13 demonstrates that the commitment to the constant term of $C = \{\hat{A}_0, \ldots, \hat{A}_{t-1}\}$ evaluates to Δ; and so, because the concrete AgVSS is zero-indexed, then $\mathsf{AV.GetPub}(0, C)$ evaluates to Δ.

$$\begin{aligned}\hat{A}_0 &= g^v \cdot \prod_{j\in[\ell]} D_j[0] \\ &= g^v \cdot \Delta \cdot g^\alpha \cdot g^{\sum_{i\in[\ell']} s_i} \\ &= g^{-\alpha-\sum_{i\in[\ell']} s_i} \cdot \Delta \cdot g^\alpha \cdot g^{\sum_{i\in[\ell']} s_i} \\ &= \Delta\end{aligned} \tag{13}$$

Hence, the adversary gains no advantage in its guess of b', regardless of whether it is operating in the real $(b = 0)$ or simulated $(b = 1)$ setting. This concludes the proof. □

C.3 Uniqueness

The uniqueness of the concrete AgVSS also follows from that of Feldman's VSS, as we now show.

Theorem 9. *The concrete AgVSS scheme is unique.*

Proof. We now show that the additional aggregation algorithm in our AgVSS construction gives the adversary no additional power in the uniqueness game than in the plain Feldman's VSS setting.

Recall the VSS uniqueness experiment in Figure 4. The adversary is required to output a valid tuple $(\mathsf{M}_1^*, \mathsf{M}_2^*, D^*)$, for which $\mathsf{S.Recover}(\mathsf{M}_1^*) \neq \mathsf{S.Recover}(\mathsf{M}_2^*)$. As above, we know that the adversary cannot win the uniqueness game in the Feldman's VSS setting, no matter what strategy it employs, including using the AgVSS aggregation algorithm, since the aggregation algorithm uses no information hidden from the adversary. Therefore the adversary cannot win the uniqueness game in the AgVSS setting either. □

D Proofs for Generic Construction

D.1 Robustness

Proof. We prove Theorem 1 by a sequence of games.

Game 0. This is the DKG weak robustness game as in Figure 7, applied to construction in Figure 11.

Let W_0 be the event that $\mathcal{A}$ wins in Game 0. Then,

$$\mathsf{Adv}^{\text{wk-rbst}}_{\mathcal{A},\mathsf{D}}(\lambda) = \Pr[W_0]$$

Let $\mathcal{C}$ be the algorithm that simulates the robustness game to $\mathcal{A}$. We now describe how $\mathcal{C}$ simulates the game.

Setup. To begin, $\mathcal{C}$ initializes $Q_1 \leftarrow \emptyset$ to simulate H_1 and $Q_2 \leftarrow \emptyset$ to simulate H_2, in addition to initializing each honest party as shown in Figure 7. $\mathcal{C}$ handles $\mathcal{A}$'s random oracle queries by lazy sampling, as follow:

H_1 : When the adversary queries H_1 on inputs (O_i, aux_i), the environment checks if it is in Q_1, and if so, returns the corresponding v_i. Otherwise, the environment randomly samples v_i from its respective domain, sets $Q_1[(O_i, \mathsf{aux}_i)] = v_i$, and then returns v_i.

H_2 : When the adversary queries H_2 on inputs $\{\psi_1, \ldots, \psi_n\}$, the environment checks if it is in Q_2, and if so, returns the corresponding aux_i. Otherwise, the environment randomly samples aux_i from its respective domain, sets $Q_2[\{\psi_1, \ldots, \psi_n\}] = \mathsf{aux}_i$, and then returns aux_i.

$\mathcal{C}$ performs all signing oracle queries honestly.

Game 1. The only difference in Game 1 is if $\mathcal{A}$ outputs a tuple $(B_j, \mathsf{pk}_j, D_j, \beta_j), j \in \mathsf{corrupt}$ where $(B_j, \mathsf{pk}_j, D_j)$ are output in $\mathsf{PerformRound}_0$ and β_j is output in $\mathsf{PerformRound}_2$ such that Equation 14 holds, then $\mathcal{C}$ aborts.

$$\begin{aligned} &\mathsf{AV}[\mathsf{H}_1].\mathsf{GetPub}(0, D_j) = \mathsf{pk}, \text{ and} \\ &\mathsf{NK.Verify}(\beta_j, B_j) \to 1, \text{ but} \\ &\mathsf{NK.SharedKey}(\beta_j, \mathsf{pk}_j) \neq \mathsf{NK.SharedKey}(\alpha_j, B_j) \end{aligned} \tag{14}$$

However, if the NIKE is binding and Assumption 1 holds, then Game 1 and Game 0 are indistinguishable to $\mathcal{A}$.

Reduction to NIKE Binding. Let $\mathcal{B}_1$ be an adversary playing against the NIKE binding game. $\mathcal{B}_1$ simulates Game 1 to $\mathcal{A}$ in exactly the same way as Game 0. However, in $\mathsf{Finalize}$, if $\mathcal{A}$ outputs some β_j with respect to the commitments $(\mathsf{pk}_j, B_j, D_j)$ output in $\mathsf{PerformRound}_0$ such that Equation 14 holds, then $\mathcal{B}_1$ recovers α_j, and outputs $(\alpha_j, \mathsf{pk}_j, \beta_j, B_j)$ as its output to the NIKE binding game.

$\mathcal{B}_1$ recovers α_j by performing $\alpha_j \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{Recover}(t, \mathsf{M}_j)$, where $\mathsf{M}_j = \{(k, w_{jk})\}_{k \in \mathsf{honest}}$. $\mathcal{B}_1$ can recover s_j because the checks in $\mathsf{PerformRound}_1$ completed successfully (else the honest participants would have aborted the protocol in $\mathsf{PerformRound}_1$ and $\mathsf{PerformRound}_2$), then each share in M_j is a valid share of the secret α_j committed to by D_j and pk_j. Further, because $\mathcal{B}_1$ simulates at least t honest players, it has a sufficient number of shares to perform this recovery step.

Hence, when $\mathcal{A}$ distinguishes Game 1 from Game 0 (i.e., by causing Game 1 to abort), then $\mathcal{B}_1$ wins.

Difference between Game 0 and Game 1. Let W_1 be the event that $\mathcal{A}$ wins in Game 1. Then,

$$\left|\Pr[W_1] - \Pr[W_0]\right| \leq \mathsf{Adv}^{\mathrm{bind}}_{\mathsf{NK},\mathcal{B}_1}(\lambda) \tag{15}$$

Game 2. The only difference in Game 2 is that in $\mathsf{Finalize}$, if Equation 16 holds, then $\mathcal{C}$ aborts.

$$\begin{aligned} &\mathsf{sk} \leftarrow \mathsf{AV.Recover}(t, \{(i, \mathsf{sk}_i)\}_{i \in \mathsf{honest}}), \text{ but} \\ &\mathsf{NK.Verify}(\mathsf{sk}, \mathsf{pk}) \rightarrow 0 \end{aligned} \tag{16}$$

However, if the AgVSS is unique, then Game 1 and Game 2 are indistinguishable.

Reduction to AgVSS Uniqueness. Let $\mathcal{B}_2$ be an adversary playing against the AgVSS uniqueness game.

$\mathcal{B}_2$ follows the protocol honestly for all parties $i \in \mathsf{honest}$. However, in $\mathsf{PerformRound}_3$, $\mathcal{B}_2$ recovers sk and tests to see if Equation 16 holds. $\mathcal{B}_2$ can recover sk without the involvement of $\mathcal{A}$ because it simulates at least t honest parties.

Recall that $\mathsf{sk}_i \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{AggPriv}(i, \{w_{i,j}\}_{j\in[n]}, \{D_j\}_{j\in[n]}, \mathsf{aux})$, and $C \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{AggPub}(\{D_j\}_{j\in[n]}, \mathsf{aux})$. The only values that differ between each honest party when performing this aggregation step are the sets $\{w_{i,j}\}_{j\in[n]}$.

If Equation 16 holds, then this means that at least one honest party $k \in \mathsf{honest}$ received a share $w_{j,k}$ from a corrupted party $j \in \mathsf{corrupt}$ such that $\mathsf{AV}[\mathsf{H}_1].\mathsf{Verify}(k, w_{j,k}, D_j) = 1$ but where Equation 17 holds.

$$\begin{aligned} &\mathsf{AV}[\mathsf{H}_1].\mathsf{Recover}(t, \{(\ell, w_{j,\ell})\}_{\ell \in \mathsf{honest}}) \neq \\ &\mathsf{AV}[\mathsf{H}_1].\mathsf{Recover}(t, \{(\ell, w_{j,\ell})\}_{\ell \in \mathsf{honest}, \ell \neq k}) \end{aligned} \tag{17}$$

Let $\mathsf{M}_1 = \{(\ell, w_{j,\ell})\}_{\ell\in\mathsf{honest}}$ and $\mathsf{M}_2 = \{(\ell, w_{j,\ell})\}_{\ell\in\mathsf{honest},\ell\neq k}$. When this event has occurred, $\mathcal{B}_2$ submits $(t, \mathsf{M}_1, \mathsf{M}_2, D_j)$ as its output to the AgVSS uniqueness game. When $\mathcal{A}$ can distinguish Game 2 from Game 1, then $\mathcal{B}_2$ wins the AgVSS uniqueness game.

Difference between Game 1 and Game 2. Let W_2 be the event that $\mathcal{A}$ wins in Game 2. Then,

$$\left|\Pr[W_2] - \Pr[W_1]\right| \leq \mathsf{Adv}^{\mathrm{unq}}_{\mathsf{AV}[\mathsf{H}_1],\mathcal{B}_2}(\lambda) \tag{18}$$

Finishing the Proof. Recall that in the weak robustness game, the adversary has four opportunities to win, assuming all honest parties have not aborted the protocol. First, if any honest party has a disjoint view of pk, second, if any honest party has a disjoint view of qual, and third, if any honest party has a disjoint view of aux. Finally, the adversary wins if the sk that is recovered from the honest parties' shares is invalid.

Because we assume a broadcast channel, if any honest party aborts in either $\mathsf{PerformRound}_1$ or $\mathsf{PerformRound}_2$, then all honest parties will abort in $\mathsf{PerformRound}_2$. And so if any honest party proceeds to $\mathsf{Finalize}$, then all honest parties will proceed to $\mathsf{Finalize}$, and all honest parties will set their status to accept. While honest parties might abort (meaning that strong robustness cannot be fulfilled), honest parties will consistently either end in an aborted or accepting status, meaning that the weak robustness game can be fulfilled. And so the only opportunity for the adversary to win against the generic construction in the weak robustness game is to disrupt consistency of $\mathsf{pk}, \mathsf{qual}, \mathsf{aux}$, or to make $\mathsf{D}[\mathsf{H}].\mathsf{Recover}$ output fail.

Again, because we assume a broadcast channel, all parties will maintain the same view of $\{(\beta_j, B_j, \mathsf{pk}_j, D_j)\}$. And so all parties will determine the set qual in $\mathsf{Finalize}$ using the same information. Hence, the adversary has no advantage in forcing a disjoint qual.

From Game 1, we know that all honest parties derive the same view of $\{\psi_j\}$ such that $\mathsf{NK.SharedKey}(\beta_j, \mathsf{pk}_j) \rightarrow \psi_j$. Because the tweak value v is derived from H_2 using the values $\{\psi_j\}_{j\in[n]}$, then all parties will obtain the same v. Because pk is derived using v and the set $\{D_j\}_{j\in[n]}$, then the adversary has no advantage in forcing a disjoint pk. Similarly, the adversary has no advantage in forcing a disjoint $\mathsf{aux} = \{\mathsf{pk}_i\}_{i\in\mathsf{qual}}$, which are also derived from v and the set $\{D_j\}_{j\in[n]}$.

Finally, from Game 2, we know that $\mathsf{D}[\mathsf{H}].\mathsf{Recover}$ will output a sk that is valid with respect to pk.

The adversary in Game 2 then has no advantage in winning the DKG weak robustness game. Concretely, combining the advantages in Equations 15 and 18 gives Equation 7. This completes the proof. □

D.2 Zero-Knowledge

Proof. We now complete the proof of Theorem 2 via a sequence of games.

Game 0. This is the DKG zero-knowledge game when $b = 0$ as in Figure 9, applied to the construction in Figure 11.

Let W_0 be the event that $\mathcal{A}$ outputs 1 in Game 0. Then,

$$\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{D},\mathcal{A},\mathsf{SimRound}_i,\mathsf{SimFinalize}}(\lambda) = \Pr[W_0]$$

Let $\mathcal{C}$ be the algorithm that simulates the DKG zero-knowledge game to $\mathcal{A}$. We now describe in further detail how $\mathcal{C}$ simulates the game.

Setup. To begin, $\mathcal{C}$ initializes $Q_1 \leftarrow \emptyset$ to simulate H_1 and $Q_2 \leftarrow \emptyset$ to simulate H_2, in addition to initializing each honest party as shown in Figure 9. $\mathcal{C}$ handles $\mathcal{A}$'s random oracle queries by lazy sampling, as follows:

H_1 : When the adversary queries H_1 on inputs (O_i, aux_i), the environment checks if it is in Q_1, and if so, returns the corresponding v_i. Otherwise, the environment randomly samples v_i from its respective domain, sets $Q_1[(O_i, \mathsf{aux}_i)] = v_i$, and then returns v_i.

H_2 : When the adversary queries H_2 on inputs $\{\psi_1, \ldots, \psi_n\}$, the environment checks if it is in Q_2, and if so, returns the corresponding aux_i. Otherwise, the environment randomly samples aux_i from its respective domain, sets $Q_2[\{\psi_1, \ldots, \psi_n\}] = \mathsf{aux}_i$, and then returns aux_i.

$\mathcal{C}$ performs all signing oracle queries honestly.

Game 1. The only difference between Game 0 and Game 1 is that in Game 1, $\mathcal{C}$ performs $\mathsf{SimRound}_0, \ldots, \mathsf{SimRound}_{\mathsf{numrounds}-1}, \mathsf{Si}$ for participant $\tau \xleftarrow{\$} \mathsf{honest}$ with respect to the challenge $\hat{\Delta}$. More specifically, when performing $\mathsf{SimRound}_0$, $\mathcal{C}$ employs $(\{w^*_{\tau,j}\}_{j\in\mathsf{corrupt}}, D^*_\tau) \leftarrow \mathsf{AV}[\mathsf{H}_1].\mathsf{SimShare}(\lambda, n, t, \tau, \hat{\Delta})$ for an honest participant τ sampled from the set of honest participants at random, such that Equation 19 holds:

$$\mathsf{S.GetPub}(0, D^*_\tau) = \hat{\Delta} \tag{19}$$

$\mathcal{C}$ performs $\mathsf{SimRound}_1, \mathsf{SimRound}_2, \mathsf{SimFinalize}$ as in the real protocol. However, one consequence of $\mathcal{C}$ simulating secret sharing with respect to $\hat{\Delta}$ is that $\mathcal{C}$ cannot correctly derive each honest players' sk_i, for all $i \in \mathsf{honest}$ when performing $\mathsf{SimFinalize}$. This however does not impact the ability of $\mathcal{C}$ to perfectly simulate all honest players, because $\mathcal{C}$ is never required to perform operations with respect to $\hat{\Delta}$ directly.

If the AgVSS is secret, then Game 0 and Game 1 are indistinguishable to $\mathcal{A}$.

Reduction to AgVSS (Non-Aggregated) Secrecy. We will construct a reduction $\mathcal{B}_1$ which is an adversary against the AgVSS (non-aggregated) secrecy game as defined in Definition 6, such that $\mathcal{B}_1$ simulates either Game 0 or Game 1 to $\mathcal{A}$. When the AgVSS hidden bit $b = 0$, $\mathcal{B}_1$ ends up simulating Game 0; when the AgVSS hidden bit $b = 1$, $\mathcal{B}_1$ ends up simulating Game 1.

$\mathcal{B}_1$ is constructed similar to Game 0, with the following exception. $\mathcal{B}_1$ simply performs $\mathsf{PerformRound}_1$ for each honest non-simulated party $i \in \mathsf{honest}, i \neq \tau$. However, for the simulated honest party τ, instead of following the protocol honestly, $\mathcal{B}_1$ instead employs the challenge received as input $(\{(j, w_j)\}_{j\in\mathsf{corrupt}}, D)$ from the AgVSS secrecy experiment. In particular, it sets $D_\tau \leftarrow D$, and $(\{(j, w_{\tau,j})\}_{j\in\mathsf{corrupt}} \leftarrow (\{(j, w_j)\}_{j\in\mathsf{corrupt}}$. It then follows the remainder of the protocol honestly. While it will not be able to derive the honest player's secret keys in $\mathsf{PerformRound}_3$, this will not have an impact on its ability to correctly simulate.

The simulation of Game 0 by $\mathcal{B}_1$ is identical to when $b = 0$ in the AgVSS secrecy game. Hence, the only distinction between Game 0 and Game 1 is when $b = 1$ in the AgVSS secrecy game.

When $\mathcal{A}$ outputs b' as its guess, then $\mathcal{B}_1$ outputs b'. Hence, when $\mathcal{A}$ distinguishes Game 0 from Game 1, $\mathcal{B}_1$ wins the AgVSS secrecy game.

Difference between Game 0 and Game 1. If the AgVSS is secret, then the difference between Game 0 and Game 1 is negligible. Let W_1 be the event that $\mathcal{A}$ wins in Game 1. Then

$$\left|\Pr[W_1] - \Pr[W_0]\right| \leq \mathsf{Adv}^{\mathsf{sec}}_{\mathsf{AV},\mathcal{B}_1}(\lambda) \tag{20}$$

$\mathsf{PerformRound}_2(\mathsf{state}_i, \mathsf{inp}, \mathsf{inb})$

1 : **for** $j \in [n], j \neq i$ **do**
2 : **if** $\mathsf{AV[H].Verify}(i, w_{ji}, D_j) \neq 1$
3 : **or if** $\mathsf{AV[H].GetPub}(0, D_j) \neq \mathsf{pk}_j$
4 : $\mathsf{state}_i.\mathsf{status} = \mathsf{abort}$
5 : **return** $(\mathsf{state}_i, \perp, \mathsf{fail})$
6 : $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup \{(i, w_{ji})\}_{j \in [n]}$
7 : $\boxed{\mathsf{M}_k = \{(j, w_{kj})\}_{k \in \mathsf{corrupt}, j \in \mathsf{honest}}}$
8 : $\boxed{\alpha_k \leftarrow \mathsf{AV.Recover[H]}(t, \mathsf{M}_k, D_k)}$
9 : $\boxed{\textbf{return } \perp \textbf{ if } \alpha_k = \mathsf{fail}}$
10 : $\boxed{\psi_k \leftarrow \mathsf{NK.SharedKey[H]}(\alpha_k, B_k), \forall\ i \in \mathsf{corrupt}}$
11 : $\boxed{O = \{D_j\}_{j \in [n]}}$
12 : $\boxed{\mathsf{aux} \leftarrow \{\psi_j\}_{j \in [n]}}$
13 : **return** $(\mathsf{state}_i,\ \perp,\ \mathsf{accept})$

Fig. 14. Game 2: Code movement, highlighting lines moved from $\mathsf{PerformRound}_3$ to $\mathsf{PerformRound}_2$

Game 2. The difference between Game 1 and Game 2 is code movement from Finalize to $\mathsf{PerformRound}_1$, as shown in Figure 14. Rather than learning corrupted players' blinding factors as input to $\mathsf{PerformRound}_3$, $\mathcal{C}$ instead derives blinding factors for all corrupted parties in $\mathsf{PerformRound}_2$, (importantly, before it reveals the honest players' blinding factors to $\mathcal{A}$) using the shares from all honest players. If $\mathcal{C}$ is unable to recover these blinding factors, then it returns $\perp$.

However, because $\mathcal{C}$ simulates at minimum t honest players (honest majority model), it can perform AV.Recover without the involvement of corrupted parties.

Difference between Game 1 and Game 2. Let W_2 be the event that $\mathcal{A}$ wins in Game 2. Then,

$$\left|\Pr[W_2]\right| = \left|\Pr[W_1]\right| \tag{21}$$

Game 3. The only difference between Game 3 and Game 2 is that if the adversary queries the random oracle H_1 with respect to any of the honest parties' blinding values $\{\psi_i\}_{i \in \mathsf{honest}}$ before the adversary is provided with the witness values $\{\beta_i\}_{i \in \mathsf{honest}}$ as input to $\mathsf{PerformRound}_3$, then the game aborts.

If the NIKE is session-key unrecoverable, the additional advantage to $\mathcal{A}$ is negligible.

Reduction to NIKE Unrecoverability. We will construct a reduction $\mathcal{B}_3$ which is an adversary against the NIKE unrecoverability game, such that $\mathcal{B}_3$ simulates Game 3 to $\mathcal{A}$.

$\mathcal{B}_3$ begins by receiving $(\mathsf{pk}_1, \mathsf{pk}_2)$ in the NIKE unrecoverability game. It then simulates Game 3 to $\mathcal{A}$ in exactly the same manner as in Game 2, with the following exception. $\mathcal{B}_3$ simply performs $\mathsf{PerformRound}_0$ honestly for each honest party $i \in \mathsf{honest}, i \neq \tau$. However, for honest party τ, $\mathcal{B}_4$ instead simulates the protocol via SimRound; it sets $\mathsf{pk}_\tau \leftarrow \mathsf{pk}_1$ and $B_\tau \leftarrow \mathsf{pk}_2$ $\mathcal{B}_3$ then simulates secret sharing via SimShare with respect to pk_τ. From Game 1, we know that doing so is indistinguishable to $\mathcal{A}$. $\mathcal{B}_3$ then follows the remainder of $\mathsf{PerformRound}_1$ as for all the other honest parties.

When simulating $\mathcal{O}_2^{\mathsf{PerformRound}_r}$, $\mathcal{B}_3$ acts honestly for all honest parties $i \in \mathsf{honest}, i \neq \tau$. However, for participant τ, $\mathcal{B}_4$ does the following. Because it does not know the corresponding β_c to B_c, it instead must guess. It looks at all of the queries made to H_1 that $\mathcal{A}$ has made up to that point, and randomly selects one of the inputs. Let this randomly selected input be $\{\psi_j\}^*_{j \in [n]}, \mathsf{aux}^*)$.

$\mathcal{B}_3$ then picks the simulated party's element $\psi_\tau \leftarrow \{\psi_j\}^*_{j\in[n]}$, and outputs $\psi' = \psi_\tau$ as its guess for the NIKE unrecoverability game. If $\mathcal{A}$ queries H_1 with ψ_τ before $\mathcal{B}_3$ has completed $\mathcal{O}_2^{\mathsf{PerformRound}_r}$, then $\mathcal{B}_3$ wins the NIKE game with probability $\frac{1}{q}$.

Difference between Game 2 and Game 3. Let W_3 be the event that $\mathcal{A}$ wins in Game 3. The additional ability for $\mathcal{A}$ to win Game 3 is then bounded by its ability to win the NIKE unrecoverability game. Hence,

$$\left|\Pr[W_3] - \Pr[W_2]\right| \leq \mathsf{Adv}^{\mathrm{unpred}}_{\mathsf{NK},\mathcal{B}_3}(\lambda) \cdot \frac{1}{q} \tag{22}$$

Game 4. The only difference between Game 3 and Game 4 is that in Game 4, $\mathcal{C}$ simulates secret sharing for participant $\tau \xleftarrow{\$} \mathsf{honest}$ with respect to the challenge $\hat{\Delta}$. From Game 1, we know that $\mathcal{C}$ can do so in such a way that is indistinguishable to $\mathcal{A}$. However, in Game 4, it additionally does so in such a way that Equation 23 holds after performing $\mathsf{Finalize}$.

$$C[\mathsf{H}_1].\mathsf{GetPub}(0, C) = \hat{\Delta} \tag{23}$$

More specifically, in $\mathcal{O}_0^{\mathsf{PerformRound}_r}$, $\mathcal{C}$ performs

$$(\{(j, w_{\tau j})\}_{j\in\mathsf{corrupt}}, D_\tau) \xleftarrow{\$} \mathsf{AV}[\mathsf{H}_1].\mathsf{SimShare}(\lambda, \mathsf{corrupt}, \hat{\Delta})\,.$$

It then follows the remainder of $\mathsf{PerformRound}_0$ as in Game 3.

Then, in $\mathcal{O}_3^{\mathsf{PerformRound}_r}$, it derives v' as $v' \xleftarrow{\$} \mathsf{AV}[\mathsf{H}_1].\mathsf{SimTweak}(O, \mathsf{state}_\tau)$. It then programs $\mathsf{H}_2(O, \mathsf{aux})$ with v'. From Game 3, we know that $\mathcal{C}$ can derive aux in $\mathcal{O}_3^{\mathsf{PerformRound}_r}$, but $\mathcal{A}$ has only negligible probability of doing so.

We will see that, if the AgVSS is securely aggregatable, then the additional advantage to the adversary in Game 4 is negligible.

Reduction to AgVSS Aggregated Secrecy. We will construct a reduction $\mathcal{B}_4$ which is an adversary against the AVSS aggregated secrecy game, such that $\mathcal{B}_4$ simulates either Game 3 or Game 4 to $\mathcal{A}$. When the hidden bit b in the AVSS aggregated secrecy game is $b = 0$, $\mathcal{B}_4$ ends up simulating Game 3; when $b = 1$, $\mathcal{B}_4$ ends up simulating Game 4.

$\mathcal{B}_4$ is constructed similar to Game 3, except as follows. When $\mathcal{A}$ queries $\mathcal{O}_1^{\mathsf{PerformRound}_r}$, for each honest player $k \in \mathsf{honest}$, $\mathcal{B}_4$ queries the AVSS oracle $\mathcal{O}^{\mathsf{GetShare}}()$, receiving $(\{w_j\}_{j\in\mathsf{honest}}, D^*)$ in return. $\mathcal{B}_4$ then uses these outputs to simulate participant τ for $\mathcal{O}_1^{\mathsf{PerformRound}_r}$. $\mathcal{B}_4$ follows the protocol honestly for all other players, and submits all $(\{w_{kj}\}_{j\in\mathsf{honest}}, D_k), k \in [n] \setminus \tau$ to the AVSS oracle $\mathcal{O}^{\mathsf{RecvShare}}$.

Then, $\mathcal{B}_4$ receives aux', and then queries the AgVSS aggregated secrecy oracle $\mathcal{O}^{\mathsf{GetTweak}}$ on inputs (O, aux'), receiving v' in return. $\mathcal{B}_4$ then programs its own $\mathsf{H}_1(O, \mathsf{aux}) = v'$. From Game 3, $\mathcal{A}$ can query for its output with negligible probability before $\mathcal{B}_4$ programs it. $\mathcal{B}_4$ then follows the remainder of Game 3 as before.

When $\mathcal{A}$ outputs b', then $\mathcal{B}_4$ outputs b'. Hence, when $\mathcal{A}$ distinguishes Game 3 from Game 4, $\mathcal{B}_4$ wins the AgVSS aggregated secrecy game.

Difference between Game 3 and Game 4. Let W_4 be the event that $\mathcal{A}$ wins in Game 4. The difference between W_3 and W_4 is bounded by the advantage of $\mathcal{A}$ in the AgVSS aggregated security game.

$$\left|\Pr[W_4] - \Pr[W_3]\right| \leq \mathsf{Adv}^{\mathrm{asec}}_{\mathsf{AV},\mathcal{B}_4,\mathsf{SimShare}}(\lambda) \tag{24}$$

Finishing the Proof. The resulting public key in Game 4 is identical to the challenge $\hat{\Delta}$, and hence is identical to the DKG zero-knowledge game when $b = 1$. The adversary then has at most negligible advantage in winning the DKG zero-knowledge game. Concretely, combining the advantage of the adversary in (20)-(24) proves Equation 9. This concludes the proof. □