Adaptive Distributed Key Generation for Discrete-Log Cryptosystems

Ruben Baecker[1], Paul Gerhart[2], Stanislaw Jarecki[3], Phillip Nazarian[3], Daniel Rausch[4], and Dominique Schröder[2,1]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
[2] TU Wien, Vienna, Austria
[3] University of California, Irvine, USA
[4] University of Stuttgart, Stuttgart, Germany

Abstract. Threshold signatures are widely deployed in decentralized asset custody and blockchain infrastructure to eliminate single points of failure. In these systems, an attacker can adaptively corrupt committee members based on public membership and protocol transcripts. Recent cryptanalysis shows this threat is concrete: schemes exposing per-party key commitments enable practical forgeries for committees of roughly 200 signers at the cost of just one minute of the global Bitcoin hashrate.

While a new generation of signing schemes defeats this attack by hiding individual key shares, every distributed key generation (DKG) protocol used in practice exposes this information, voiding the adaptive security of any scheme built on top. Trusted dealers avoid the leakage but reintroduce the single point of failure that threshold cryptography exists to eliminate. True adaptive security must therefore span both key generation and signing.

We present Janus, the first DKG suite to achieve this end-to-end guarantee. Janus serves as a drop-in replacement for a trusted dealer in threshold schemes, hiding all key shares while outputting a standard DLog public key compatible with existing verifiers, wallets, and smart contracts. Assuming secure erasures, we prove security against an adaptive adversary that corrupts members mid-protocol and controls a dishonest majority. We also show that Janus directly supports a variety of DLog-based primitives, including threshold Schnorr signatures and oblivious pseudorandom functions. Our suite includes Janus2R, which achieves an unbiased key in two rounds, and Janus1R, which completes within a single round by tolerating an additive key shift. When a node misbehaves, both variants generate a 259-byte universally verifiable proof for automated on-chain slashing, and both naturally allow committees to proactively refresh their shares without changing the public key. Our open-source Rust implementation completes key generation for a 16-node intercontinental committee in 310 ms and scales to 512 nodes.

Keywords: Distributed Key Generation, Adaptive Security, Universal Composability, Proactive Security

1 Introduction

Threshold cryptographic systems have become a central building block of modern distributed systems and blockchain infrastructure. They enable decentralized custody, distributed signing of high-value transactions, and threshold wallets without entrusting any single party with the full secret key. Such systems now protect substantial assets across real-world deployments reliant on discrete-logarithm signatures (such as Ed25519, sr25519, and secp256k1), ranging from institutional custody platforms like Fireblocks and Coinbase's Wallet-as-a-Service (WaaS) [48,51] to threshold bridging and staking protocols for Bitcoin (Taproot/BIP340). In these high-stakes environments, distributed committees must jointly manage signing keys across long lifetimes against sophisticated, reactive adversaries, where a single point of failure can lead to catastrophic financial loss or network-wide liveness failure.

Requirements of the Decentralized Setting Three requirements emerge from the nature of blockchain deployments. (i) Because blockchains are public and observable, adversaries can leverage the public protocol transcript to strategically target specific nodes. Security must therefore hold against *adaptive corruptions*, where the adversary's choice of targets evolves based on information gathered during execution. (ii) As corruptions accumulate over long durations, systems must provide *proactive security* through periodic key refresh. This process renews each party's secret share without altering the global public key pk, thereby avoiding the on-chain overhead and state migration risks associated with public-key rotation. (iii) The Distributed Key Generation (DKG) phase that bootstraps a threshold system takes place before any trust is established among participants. Consequently, a DKG cannot rely on a trusted majority, unlike consensus protocols that can pre-screen validators. Systems must therefore tolerate a *dishonest majority*. Without an honest majority, any malicious participant can stall the protocol, making *identifiable abort (IA)* a critical requirement: upon failure, honest parties must be able to identify at least one cheating participant and produce verifiable evidence of their misbehavior. This enables recovery of liveness through economic penalties such as slashing and the exclusion of adversarial nodes.

Threshold Schnorr signatures, as instantiated by EdDSA over Edwards25519 and Bitcoin's BIP340 over secp256k1, are a widely deployed and practically significant instance of the decentralized setting. NIST's First Call for Multi-Party Threshold Schemes [52] includes Schnorr-based primitives as an explicit target. In addition to the three requirements above, threshold Schnorr schemes must satisfy two further properties. (iv) To maintain *compatibility* with existing hardware security modules and on-chain verification logic (e.g., Bitcoin's BIP340), threshold signatures must verify under existing single-party Schnorr verification logic. Consequently, the public key must retain its *standard form* $\mathsf{pk} = g^{\mathsf{sk}}$. This requirement is also part of NIST's standardization call [52]. (v) As shown by Crites and Stewart [29], many threshold signing schemes that expose individual public key shares $\mathsf{pk}_i = g^{\mathsf{sk}_i}$ (including threshold variants of NIST-standardized ECDSA and EdDSA) must rely on the non-falsifiable *P-problem* to achieve adaptive security. This is not merely a theoretical concern. For an Ed25519-sized group ($p \approx 2^{252}$) with a large-scale blockchain committee of $n = 196$ at the full corruption threshold, Crites and Stewart [29] show that forging a signature requires work equivalent to a single minute of the global Bitcoin hashrate. This represents a readily tractable attack budget for a nation-state or well-funded industrial adversary. Therefore, threshold Schnorr schemes must be *key-share hiding*.

The DKG Gap While recent threshold Schnorr signature schemes satisfy all five target requirements [5,4,3,8], they rely on a trusted setup for key generation, reintroducing the single point of failure that threshold cryptography exists to eliminate. In practice, the key material must be generated using a Distributed Key Generation (DKG) protocol.

A DKG for this setting must also satisfy these requirements to avoid degrading the security of the signing scheme. Existing DKGs used in practice (including those underlying FROST [27] and related deployments [28]) expose key shares through a Feldman VSS-based verification mechanism, reintroducing the vulnerability exploited by the Crites-Stewart attack [29]. Among share-hiding DKGs, the state of the art is the five-round protocol of Canetti *et al.* [22], which is restricted to an honest majority. In summary:

> *No existing DKG simultaneously produces standard master public keys, achieves key-share hiding, and provides adaptive security with identifiable abort in the dishonest-majority setting.*

1.1 Our Contributions

We resolve this tension with the following contributions:

– **Janus: Low-Latency, Adaptively Secure DKG.** We present Janus, the first DKG to simultaneously satisfy all five target requirements. Janus comes in two variants:
  - **Janus$_{\mathsf{2R}}$:** A two-round protocol that ensures an unbiasable public key distribution, intended for high-security root-of-trust setups.
  - **Janus$_{\mathsf{1R}}$:** A *single-round* protocol that optimizes for latency at the cost of an adversarial shift in the resulting public key distribution.

		BC Rounds	SH & std. pk	Liveness	Dishon. Maj.	UC	Assumption	Erasure	ROM
Prior Work	CGJKR99 [22]	5	•	•			L	•	
	JL00 [43]	10		•			D		
	AF04 [1]	5		•		•†	D		
	FMT24 [35]	3*		•	•		D	•	•
Concurrent Work	EMR26 [34]	3	•	•		•	D		•
	EMR26 [34]	3	•	•		•	D		•
	EMR26 [34]	2	•	•	•	•	D		•
Our Work	$\mathsf{Janus}_{\mathsf{2R}}$	2	•	•	•	•	C	•	•
	$\mathsf{Janus}_{\mathsf{1R}}$	1	•	•‡	•	•	C	•	•

Table 1: Comparison of adaptively secure DLog-based DKG protocols in the synchronous setting. All protocols rely on a broadcast channel. SH denotes key-share-hiding. Liveness denotes guaranteed output delivery ($t < n/2$) or identifiable abort ($t < n$, where GOD is impossible). Security assumptions: L (DLog), C (CDH), and D (DDH). *One of the three rounds is a multicast round. †Abe and Fehr [1] prove security in the "single-inconsistent-player" UC model, which does not provide standard UC composability guarantees. ‡$\mathsf{Janus}_{\mathsf{1R}}$ requires one additional broadcast round to achieve identifiable abort.

Both variants naturally extend to proactive key refresh, allowing decentralized networks to recover from key leakage by renewing secret shares without changing the public key.

– **Formal Model.** We introduce a new ideal functionality in the Universal Composability (UC) framework (e.g., [19,50,17]). It is the first DKG functionality to output a standard public key $\mathsf{pk} = g^{\mathsf{sk}}$ alongside hiding key-share commitments, while supporting identifiable abort under adaptive corruptions. This establishes a formal foundation for systems where cryptographic misbehavior must be attributable to enforce economic penalties. Beyond our specific constructions, the functionality provides a reusable abstraction for analyzing the security of threshold protocols.
– **Implementation and Evaluation.** We provide an open-source Rust implementation of Janus and evaluate both variants through microbenchmarks and end-to-end latency estimates for WAN deployments. In its full UC-secure configuration, $\mathsf{Janus}_{\mathsf{1R}}$ completes key generation for a 16-node intercontinental committee in under 400 ms, and both protocols scale efficiently to 512 nodes. Verifiable misbehavior complaints require a single 259-byte message, verifying in 0.14 ms for $\mathsf{Janus}_{\mathsf{1R}}$ (independent of committee size) and at most 8.2 ms for $\mathsf{Janus}_{\mathsf{2R}}$ at $n = 512$. Finally, we quantify the concrete overhead of full UC security: straight-line extractable proofs dominate computation in $\mathsf{Janus}_{\mathsf{1R}}$ and can be replaced by proofs an order of magnitude faster in the algebraic group model or under an honest majority.

1.2 Related Work

DKG protocols for discrete-logarithm systems have been studied extensively to eliminate the single point of failure in trusted setups. Foundational designs, such as the Pedersen DKG [54], provided efficient secret sharing but allowed a rushing adversary to bias the public key and lacked liveness guarantees, which was resolved by Gennaro *et al.* [38] in a multi-round protocol. Below, we focus on adaptively secure synchronous DKGs (see [6] for a comprehensive survey).

Canetti *et al.* [22] proposed the first adaptively secure DKG (5 broadcast rounds, honest majority). Subsequent erasure-free protocols incur prohibitive communication (10 and 5 broadcast rounds in [43] and [1], respectively) and leak key-share commitments ([43,1,59]). More recently, GRandLine [7] achieves log-quadratic communication but outputs key shares as group elements, which limits its applicability, while Feng *et al.* [35] assume an any-trust model but lack key-share hiding. Crucially, none of these protocols achieve all five target requirements, and all rely on standalone or specialized models that lack universal composability guarantees.

The signing schemes that consume DKG keys underscore this gap. While RFC-standardized FROST [49,27] and Chu *et al.* [25] achieve adaptively secure signing, both rely on share-exposing DKGs, inheriting the *P-problem* vulnerability at full corruption. Similarly, robust protocols like ROAST [58] assume static corruptions, while recent adaptive constructions (such as Bacho *et al.* [3] for Schnorr and Canetti *et al.* [20] for ECDSA) still presuppose a trusted setup or share-exposing DKG. Janus provides the adaptively secure, share-hiding foundation these signing protocols require.

Concurrent and Independent Work Ek, Melissaris, and Roy [34] recently proposed adaptively UC-secure, key-share hiding DKG protocols without secure erasures or non-standard assumptions, including a three-round unbiased protocol with guaranteed output delivery assuming honest majority, and a two-round protocol with identifiable abort under a dishonest majority. While their work is a significant result in reducing assumptions, the Janus suite prioritizes systems-oriented performance. Specifically, $\mathsf{Janus}_{\mathsf{1R}}$ is the first single-round adaptively secure DKG by permitting an adversarial shift, an acceptable trade-off in latency-sensitive, large-scale decentralized networks. Furthermore, while the two-round scheme of [34] avoids erasures under DDH, $\mathsf{Janus}_{\mathsf{2R}}$ relies on erasures and in exchange requires only the weaker CDH assumption. Because our target deployments require proactive key refresh, which inherently requires erasures to delete old shares, relying on erasures imposes no additional constraint.

Bünz, Choi, and Komlo [15] proposed a single-round DKG with adversarial shift in the *static* corruption setting, supporting in-round identifiable aborts at the cost of expensive cryptographic primitives. In contrast, Janus achieves adaptive security while avoiding these overheads by adopting report-based per-party identifiable aborts.

2 A Distributed Key Generation Functionality

We first define $\mathcal{F}_{\mathsf{DKG}}^{t,n}$, an ideal DKG functionality for discrete-logarithm threshold systems, capturing adaptive security against reactive node compromise and cryptographic attribution (identifiable abort) for automated slashing. We build on the statically secure functionality of Katz [46] (Figure 1) and refine it step by step for our setting.

Katz's functionality [46] lets the adversary choose the secret shares of all corrupted parties, which are then interpolated into a random polynomial f of degree at most $t-1$. The master secret key is $\mathsf{sk} = f(0)$ with public key $\mathsf{pk} = g^{\mathsf{sk}}$, while each party $\mathcal{P}_i$ receives a secret share $\mathsf{sk}_i = f(i)$ and verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i}$. This ensures a uniformly distributed joint key while preserving adversarial control over corrupted shares. This functionality is state-of-the-art for statically secure DKG, but it does not meet the requirements of adaptively secure threshold systems in the dishonest majority setting. We identify the following mismatches:

1. **Share-Binding.** The partial verification keys $\mathsf{vk}_i = g^{f(i)}$ bind parties to their secret share. As discussed in Section 1, verification keys must be key-share hiding under adaptive corruptions to prevent the Crites and Stewart attack [29].
2. **Static Corruption.** Targeting static corruption, it naturally lacks a corruption interface to model adaptive corruption.
3. **Implicit Honest Majority.** The functionality enforces guaranteed output delivery for honest parties, implicitly assuming an honest majority. Since we target the dishonest majority setting, we cannot guarantee output delivery.
4. **Identifiable Abort.** To ensure liveness without an honest majority, our functionality must incorporate an identifiable abort mechanism to attribute misbehavior.

We address these limitations by developing a refined functionality for adaptive security in DLog-based threshold systems.

Key Share Hiding We replace standard verification keys of the form $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ with perfectly hiding Pedersen commitments to keep key shares hidden. Concretely, the functionality samples randomness $\omega_i \leftarrow\!\$\ \mathbb{Z}_p$ and sets

$$\mathsf{sk}_i := f(i), \qquad \mathsf{vk}_i := g^{\mathsf{sk}_i} \cdot h^{\omega_i}.$$

$\mathcal{F}_{st-\mathtt{DKG}}^{t,n}$

The functionality interacts with an ideal adversary $\mathcal{A}^*$ and an ordered set $\mathcal{P}$ of n parties. Let Corr with $|\text{Corr}| \leq t-1$ be the set of corrupted parties in $\mathcal{P}$. The functionality assumes group parameters $(\mathbb{G}, p, g)$.

1. Receive $\{\mathsf{sk}_i\}_{i\in\text{Corr}}$ from $\mathcal{A}^*$ and set $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ for all $i \in \text{Corr}$.
2. Pick a random $(t-1)$-degree polynomial f over $\mathbb{Z}_p$ s.t. $f(i) = \mathsf{sk}_i$ for all $i \in \text{Corr}$.
3. Set $\mathsf{sk} := f(0)$ and $\mathsf{pk} = g^{\mathsf{sk}}$.
4. For all parties $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$, set $\mathsf{sk}_i := f(i)$, and $\mathsf{vk}_i = g^{\mathsf{sk}_i}$.
5. Set $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, send $(\mathsf{sk}_i, \mathsf{pk}, \mathsf{vk})$ to each party.

Fig. 1: Ideal functionality for statically secure DKGs [46].

While individual shares remain hidden in vk_i, the public key retains its standard form $\mathsf{pk} = g^{f(0)}$, ensuring compatibility with downstream DLog-based threshold protocols [3,8].

Adaptive Corruption We extend the functionality with a corruption interface that enables the adversary to corrupt a party at any time. Corrupting a party $\mathcal{P}_i$ before its key share is generated allows the adversary to set its key-share, whereas subsequent corruption reveals its existing share $(\mathsf{sk}_i, \omega_i)$. The functionality restricts total corruptions to $t-1$, rejecting further corruption requests beyond the threshold.

Dishonest Majority and Identifiable Abort Because guaranteed output delivery is impossible under a dishonest majority, our functionality supports identifiable abort. This allows the adversary to abort protocol execution, but guarantees that any abort is attributed to at least one corrupted party. In practical deployments, identifying cheaters enables systems to restore liveness by excluding malicious nodes before rerunning, or by discouraging malicious behaviour through economic penalties (slashing).

Formally, the adversary controls completion via two mutually exclusive messages: sending $\mathsf{success}$ triggers output delivery of key shares to all honest parties, whereas sending $(\mathsf{abort}, \mathcal{P}_j)$ exposes a currently corrupted party $\mathcal{P}_j$ and causes honest parties to output (abort, j).

These changes result in the functionality $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ (Figure 2), capturing all our target properties: adaptive security, key-share hiding, dishonest-majority viability, and identifiable aborts.

However, $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ cannot be realized in a single round. Katz [46] showed that no one-round protocol can achieve a *statistically* unbiased public key distribution, even against a single corrupted party. To circumvent this barrier, we introduce a relaxed functionality $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$.

Reducing Round Complexity Through Shifts Our relaxed functionality $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$ allows an additive shift to the secret sharing. Internally, it samples a preliminary sharing polynomial f' over $\mathbb{Z}_p$, defining baseline shares $\mathsf{sk}'_i := f'(i)$ and key $\mathsf{sk}' := f'(0)$. Following Bünz, Choi, and Komlo [15], the adversary may specify a "polynomial shift" Δ (with degree $\leq t-1$), yielding the final key $\mathsf{sk} := \mathsf{sk}' + \Delta(0)$ and shares $\mathsf{sk}_i := \mathsf{sk}'_i + \Delta(i)$. Similarly, the adversary can specify an additive shift to the randomness of the key-share commitments.

Crucially, because honest shares are hidden within perfectly hiding commitments, the adversary learns only the preliminary public key $\mathsf{pk}' = g^{f'(0)}$, so the shift is *blind*. The adversary may choose Δ after seeing pk' and can thus steer $\mathsf{pk} = \mathsf{pk}' \cdot g^{\Delta(0)}$ towards any efficiently testable property, but it cannot cancel the honest contribution, since $\mathsf{sk} = f'(0) + \Delta(0)$ remains unknown to it. Intuitively, Δ formalizes the unavoidable rushing advantage in a single-round protocol: an adversary observing all honest messages before publishing its own message can adapt its contribution, but cannot bias honest shares.

Round Optimality Through Report-Based Identifiable Aborts Standard identifiable abort requires all honest parties to reach global agreement on an abort. In a single-round DKG, every party must therefore detect

$\mathcal{F}_{\mathsf{DKG}}^{t,n}$

The functionality interacts with an ideal adversary $\mathcal{A}^*$ and an ordered set $\mathcal{P}$ of n parties identified by numbers $\{1, \ldots, n\}$. Let Corr be the set of all corrupted parties in $\mathcal{P}$, initially empty.

Init: Upon first activation, the functionality runs a group generation algorithm $\mathsf{GroupGen}(1^\eta)$ that returns group parameters $(\mathbb{G}, p, g)$ and Pedersen commitment parameter h. All parties and the adversary can send a query to obtain these parameters.

Setup: Upon receiving init from party $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$:
Send (init, i) to $\mathcal{A}^*$.

Setup Finished: Upon receiving $(\mathsf{initcomplete}, \{\mathsf{sk}_i, \omega_i\}_{i \in \text{Corr}})$ from $\mathcal{A}^*$, where $\mathsf{sk}_i, \omega_i \in \mathbb{Z}_p$:

1. If setup has already finished, return control to $\mathcal{A}^*$.
2. If init has not been received for all $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$, return control to $\mathcal{A}^*$. Otherwise proceed.
3. Pick a random $(t-1)$-degree polynomial f over $\mathbb{Z}_p$ s.t. $f(i) = \mathsf{sk}_i$ for all $i \in \text{Corr}$, and set $\mathsf{sk} := f(0)$ and $\mathsf{pk} = g^{\mathsf{sk}}$.
4. For all parties $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$, set $\mathsf{sk}_i := f(i)$ and $\omega_i \leftarrow\!\!\$\, \mathbb{Z}_p$.
5. Set $\mathsf{vk}_i = g^{\mathsf{sk}_i} \cdot h^{\omega_i}$ for all $i \in \mathcal{P}$.
6. Set $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ and return $(\mathsf{pk}, \mathbf{vk})$ to $\mathcal{A}^*$.

Success Protocol: Upon receiving $\mathsf{success}$ from $\mathcal{A}^*$:
If setup has finished and $\mathcal{A}^*$ has not previously sent $\mathsf{success}$ or abort, output $((\mathsf{sk}_i, \omega_i), \mathsf{pk}, \mathsf{vk})$ to each $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$.[a]

Abort Protocol: Upon receiving (abort, j) from $\mathcal{A}^*$:
If $\mathcal{A}^*$ hasn't sent $\mathsf{success}$ and $\mathcal{P}_j \in \text{Corr}$, then output (abort, j) to all initialized and honest parties. Note that the adversary can trigger this several times in case multiple misbehaving parties were identified.

Corrupt Party: Upon receiving $(\mathsf{corrupt}, j)$ from $\mathcal{A}^*$:
If $j \in \text{Corr}$ or $|\text{Corr}| = t - 1$ return $\perp$. Else, set $\text{Corr} \leftarrow \text{Corr} \cup \{j\}$ and return $\top$ as well as, if setup has finished, $(\mathsf{sk}_j, \omega_j)$. As usual, from this point onwards $\mathcal{A}^*$ gets to control inputs and outputs of $\mathcal{P}_j$.

[a] Despite being a commonly used notation in the UC literature, "sending outputs to multiple parties" is technically not possible within UC. However, one can implement very close approximations of this operation using standard modeling techniques. For readability we thus keep the more intuitive and established notation of "sending multiple outputs" in the main body while recalling the implementation we consider in Section D.1 for formal completeness.

Fig. 2: Ideal functionality for Distributed Key Generation with (Pedersen) committed shares, identifiable abort, and adaptive corruptions.

protocol failure solely based on first-round messages. However, because parties encrypt secret shares point-to-point, a malicious party can distribute malformed ciphertexts that only the intended recipient can detect privately. Reaching global abort agreement within a single round thus requires publicly verifiable encryption with heavy cryptographic overhead [15]. To achieve round optimality without expensive primitives, we relax the abort mechanism.

Concretely, $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$ supports *report-based identifiable aborts* on a per-party basis rather than requiring protocol-wide agreement. Each party may either output a valid key share or abort independently of others. Whenever a party aborts, all honest parties (including those that already obtained a valid key share) receive an attributable report identifying both the aborting party $\mathcal{P}_i$ and the blamed corrupted party $\mathcal{P}_j$. Additionally, $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$ retains protocol-wide aborts for misbehavior that is publicly verifiable from broadcast messages.

$\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$

The functionality interacts with an ideal adversary $\mathcal{A}^*$ and an ordered set $\mathcal{P}$ of n parties identified by numbers $\{1,\ldots,n\}$. Let Corr be the set of all corrupted parties in $\mathcal{P}$, initially empty.

Init: Upon first activation, the functionality runs a group generation algorithm $\mathsf{GroupGen}(1^\eta)$ that returns group parameters $(\mathbb{G}, p, g)$ and Pedersen commitment parameter h. All parties and the adversary can send a query to obtain these parameters. Next, pick a random $(t-1)$-degree polynomial f' over $\mathbb{Z}_p$ and random exponents $\{\omega'_i\}_{i\in\mathcal{P}}$. Set $\mathsf{pk}' \leftarrow g^{f'(0)}$, and $\mathsf{vk}'_i = g^{f'(i)} \cdot h^{\omega'_i}$ for all $i \in \mathcal{P}$. The adversary $\mathcal{A}^*$ can send a query to learn $(\mathsf{pk}', \{\mathsf{vk}'_i\}_{i\in\mathcal{P}})$.

Setup: Upon receiving init from party $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$:
Send (init, i) to $\mathcal{A}^*$.

Setup Finished: Upon receiving $(\mathsf{initcomplete}, \Delta, \{\Omega_i\}_{i\in\mathcal{P}})$, where Δ is a $(t-1)$-degree polynomial over $\mathbb{Z}_p$ and $\Omega_i \in \mathbb{Z}_p$ are exponents:

1. If setup has already finished, return control to $\mathcal{A}^*$.
2. If init has not been received for all $\mathcal{P}_i \in \mathcal{P} \setminus \text{Corr}$, return control to $\mathcal{A}^*$. Otherwise proceed.
3. Set $f \leftarrow f' + \Delta$, and set $\mathsf{sk} := f(0)$ and $\mathsf{pk} = g^{\mathsf{sk}}$.
4. For all parties $\mathcal{P}_i \in \mathcal{P}$, set $\mathsf{sk}_i := f(i)$ and $\omega_i \leftarrow \omega'_i + \Omega_i$.
5. Set $\mathsf{vk}_i = g^{\mathsf{sk}_i} \cdot h^{\omega_i}$ for all $i \in \mathcal{P}$.
6. Set $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, return $(\mathsf{pk}, \mathbf{vk})$ to $\mathcal{A}^*$, and record ready.

Success Party: Upon receiving $(\mathsf{success}, i)$ from $\mathcal{A}^*$:
If $\mathcal{P}_i$ is honest, setup has finished and $\mathcal{A}^*$ has not previously triggered a $\mathsf{success}$ or abort output to $\mathcal{P}_i$, output $((\mathsf{sk}_i, \omega_i), \mathsf{pk}, \mathsf{vk})$ to $\mathcal{P}_i$.

Abort Party: Upon receiving (abort, i, j) from $\mathcal{A}^*$:
If $\mathcal{A}^*$ has not triggered a $\mathsf{success}$ output to $\mathcal{P}_i$, and $\mathcal{P}_j \in \text{Corr}$, then output $(\mathsf{reportAbort}, i, j)$ to all initialized and honest parties except $\mathcal{P}_i$. If $\mathcal{P}_i$ is honest, also output (abort, j) to $\mathcal{P}_i$. Note that the adversary can trigger this several times in case multiple misbehaving parties were identified.

Abort Protocol: Upon receiving (abort, j) from $\mathcal{A}^*$:
If $\mathcal{A}^*$ hasn't triggered a $\mathsf{success}$ output to any $\mathcal{P}_i$, and $\mathcal{P}_j \in \text{Corr}$, then output (abort, j) to all initialized and honest parties. Note that the adversary can trigger this several times in case multiple misbehaving parties were identified.

Corrupt Party: Upon receiving $(\mathsf{corrupt}, j)$ from $\mathcal{A}^*$:
If $j \in \text{Corr}$ or $|\text{Corr}| = t-1$ return $\perp$. Else, set $\text{Corr} \leftarrow \text{Corr} \cup \{j\}$ and return $(\top, f'(j), \omega'_j)$ as well as, if setup has finished, $(\mathsf{sk}_j, \omega_j)$. As usual, from this point onwards, $\mathcal{A}^*$ gets to control inputs and outputs of $\mathcal{P}_j$.

Fig. 3: Ideal functionality for *Shifted* Distributed Key Generation with (Pedersen) committed shares, identifiable abort, adaptive corruptions, where the adversary can introduce an additive shift to the secret key.

Allowing parties to proceed with valid outputs while others abort not only enables single-round execution, but also aligns naturally with practical threshold systems:

- **Robustness without restarts:** Downstream applications (e.g., threshold signing) require only t operational signers and can proceed despite isolated aborts without rerunning the DKG. While $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ does not guarantee that at least t parties succeed, downstream protocols can verify quorum availability during their very first communication round (either by collecting t valid partial signatures or by observing the number of abort reports). Moreover, under adaptive corruption and a dishonest majority, the adversary can break downstream liveness at any point regardless (e.g., by corrupting signers later); thus, enforcing an upfront completion quorum during DKG yields no additional resilience.
- **Immediate local output:** Honest parties can use their shares and the resulting public key as soon as local computation completes, without waiting for global consensus. Because the overall public key pk is

uniquely fixed by the first-round public broadcasts, all non-aborting parties obtain the identical public key, eliminating synchronization latency and enabling downstream protocols to begin using the key material immediately.
– **Decoupled attribution:** Late-arriving abort reports do not compromise the secrecy or correctness of previously generated key shares or of the overall public key. The reports only provide attribution data for external slashing mechanisms.

We formally specify $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ with shiftable keys and report-based identifiable aborts in Figure 3. While its guarantees are strictly weaker than $\mathcal{F}^{t,n}_{\mathsf{DKG}}$, we show in Section 7 that they suffice for common decentralized applications.

Looking ahead, we prove that $\mathsf{Janus}_{\mathsf{2R}}$ (Section 3.1) securely realizes $\mathcal{F}^{t,n}_{\mathsf{DKG}}$ and $\mathsf{Janus}_{\mathsf{1R}}$ (Section 3.2) securely realizes $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ in the UC framework.

3 Janus: Adaptively Secure DKG

In this section, we introduce the *Janus*[5] protocol suite, comprising $\mathsf{Janus}_{\mathsf{2R}}$, a two-round protocol achieving an unbiased key distribution, and $\mathsf{Janus}_{\mathsf{1R}}$, a single-round protocol that permits an adversarial shift in the key distribution. We first describe the common backbone shared by both variants, and then detail the protocol-specific mechanisms of $\mathsf{Janus}_{\mathsf{2R}}$ and $\mathsf{Janus}_{\mathsf{1R}}$ in Sections 3.1 and 3.2.

Following the standard paradigm that all discrete-logarithm DKGs share [55,38,22,43,1,35,6,34,15], Janus has each party execute an independent Verifiable Secret Sharing (VSS) instance over broadcast channel[6], aggregating all valid contributions into the joint secret sharing. Within this universal architecture, our core technical challenge lies in re-engineering the underlying components to solve three interdependent problems: (i) constructing a standard DLog public key while hiding individual key shares, (ii) supporting adaptive corruptions, and (iii) attributing misbehavior to a specific party. The two variants then diverge in when the individual key contributions are revealed and how the VSS commitments and verification keys are structured.

Pedersen VSS and Key Decoupling We base Janus on perfectly hiding Pedersen VSS to prevent the Crites-Stewart attack [29], which exploits unique key share commitments. Aggregating n parallel Pedersen VSS instances yields randomized verification key commitments $\mathsf{vk}_i = g^{\mathsf{sk}_i} h^{\omega_i}$ with fresh randomizers ω_i, leaking no information about the individual secret shares sk_i.

However, Pedersen commitments naturally produce a blinded overall public key $\mathsf{pk}' = g^{\mathsf{sk}} h^{\omega}$ rather than the standard public key $\mathsf{pk} = g^{\mathsf{sk}}$. We recover the standard public key by having each party additionally broadcast its unblinded partial key contribution $\mathsf{pk}_i = g^{f_i(0)}$ alongside a NIZK proof certifying consistency with its VSS commitment. The master key is then computed directly as $\mathsf{pk} = \prod_{i \in \mathcal{P}} \mathsf{pk}_i = g^{\mathsf{sk}}$. Crucially, publishing pk_i exposes only the constant term $f_i(0)$ of the VSS sharing polynomial in the exponent, revealing no information about individual share evaluations $f_i(j)$. The partial verification keys vk_i therefore remain strictly key-share hiding. The two Janus variants diverge in *when* pk_i is revealed, as detailed below.

Batched Hashed ElGamal We use a batched variant of Hashed ElGamal encryption for point-to-point share distribution. To encrypt shares for all n parties efficiently, the sender samples ephemeral randomness $r \leftarrow\!\$\, \mathbb{Z}_p$, sets the shared first component $c_0 := g^r$, and computes the recipient-specific ciphertexts $c_j := \mathsf{H}((\mathsf{pk}_j^E)^r) \oplus m_j$, where H is modeled as a random oracle. To ensure plaintext extractability, the sender $\mathcal{P}_i$ attaches a proof of knowledge of r such that $c_0 = g^r$, whose language we denote by L^{enc}, and Dec returns $\perp$ if this proof does not verify. For simplicity, we abstract this mechanism as Enc and Dec in our protocol descriptions (Figures 4 and 5). Here and in every other proof of Janus, the statement contains the index of the proving party, so that no party can present an honest party's proof, or an honest party's c_0, as its own.

[5] In Roman mythology, Janus is the god of beginnings and transitions.
[6] According to [6], [59] is the only protocol that does not rely on broadcast channels in the synchronous setting.

Adaptive Security via Selective Erasure and Equivocation To handle adaptive corruptions, the simulator must be able to equivocate the internal state of any party corrupted mid-protocol, making all past transcripts consistent with the newly revealed state. Because encryption keys pk_j^E are static PKI entries required for identifiable aborts (see below), the secret keys sk_j^E cannot be erased. Therefore, an adaptive adversary corrupting party $\mathcal{P}_j$ learns sk_j^E and can retroactively decrypt all past inbound ciphertexts.

We resolve this challenge through a balanced combination of selective erasures and random-oracle equivocability:

- **Sender-side erasures:** Immediately after broadcasting, each party securely erases its secret sharing polynomial $f_i(x)$, the blinding values, and its ephemeral encryption randomness r. Because all evaluations and blinding factors are already sent, this resolves the sender commitment problem, relieving the simulator from having to explain outbound messages.
- **Receiver-side equivocation:** For inbound messages, the simulator leverages the equivocability of Hashed ElGamal. Since the underlying Diffie-Hellman values $(\mathsf{pk}_j^E)^r$ remain unqueried by an adversary that does not know sk_j^E, the simulator can adaptively program the random oracle H upon a recipient corruption, opening past inbound ciphertexts to consistent values.

Consequently, the simulator can consistently explain the complete internal state and communication history of adaptively corrupted parties without ever having to equivocate the erased sender polynomials.

Identifiable Aborts via Verifiable Complaints Shares are encrypted under the recipient's public key. When a recipient $\mathcal{P}_i$ receives an inconsistent ciphertext, it publishes the Diffie-Hellman value $K_i := (g^r)^{\mathsf{sk}_i^E}$ with a DLog equality proof. Revealing K_i allows any party to evaluate $\mathsf{H}(K_i)$ in the clear, avoiding expensive zero-knowledge proofs over the random oracle. Combined with the broadcast transcript, any observer can reconstruct the plaintext share and publicly verify its inconsistency against the sender's VSS commitments, yielding a lightweight, universally verifiable complaint. In $\mathsf{Janus}_{\mathsf{1R}}$, a complaint broadcast is an additional message beyond the main round, but requires no interactive follow-up.

The Two Variants: Timing and Public Key Derivation The first difference between the two variants lies in when parties reveal their unblinded public key contributions $\mathsf{pk}_i = g^{f_i(0)}$. In $\mathsf{Janus}_{\mathsf{2R}}$, pk_i is withheld until Round 2. This prevents a rushing adversary from observing honest key contributions before fixing its own polynomial, guaranteeing an unbiased master key distribution at the cost of an additional communication round. In contrast, $\mathsf{Janus}_{\mathsf{1R}}$ broadcasts pk_i immediately in Round 1 alongside the VSS commitments. While this permits a blind additive shift to the key distribution, it completes within a single round.

The Two Variants: VSS Structure and Verification Key Derivation The second difference concerns how VSS commitments and individual verification keys are constructed. In $\mathsf{Janus}_{\mathsf{2R}}$, parties commit to the *coefficients* of their sharing polynomial as $c_{i,k} \leftarrow g^{a_{i,k}} \cdot v^{b_{i,k}}$. While it is possible to evaluate the polynomial in the exponent, resulting in $c_{i,j} \leftarrow g^{f_i(j)} \cdot v^{f'_i(j)}$, the aggregated randomness would be correlated between all parties. Therefore, each party commits to an independent blinding value ω_i during the first round and broadcasts its verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i} \cdot h^{\omega_i}$ during the second round, proving well-formedness with a NIZK.

In $\mathsf{Janus}_{\mathsf{1R}}$, parties commit directly to the *evaluations* of their sharing polynomial instead: $c_{i,j} \leftarrow g^{f_i(j)} h^{\omega_{i,j}}$, sampling each blinding factor $\omega_{i,j}$ independently per recipient. This allows verification keys to be aggregated non-interactively from first-round messages alone. Because these blinding factors share no polynomial relationship, the aggregated blinding $\omega_i = \sum_{j \in \mathcal{P}} \omega_{j,i}$ is uniformly random, ensuring the derived verification keys $\mathsf{vk}_i = g^{\mathsf{sk}_i} h^{\omega_i}$ remain perfectly hiding.

3.1 $\mathsf{Janus}_{\mathsf{2R}}$

Our two-round protocol, $\mathsf{Janus}_{\mathsf{2R}}$, UC-realizes $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ (Figure 2) under the CDH assumption (Definition 1) in the $(\mathcal{F}_{\mathrm{bc}}, \mathcal{F}_{\mathrm{ro}}, \mathcal{F}_{\mathrm{nizk}}, \mathcal{F}_{\mathrm{pki}})$-hybrid world assuming secure erasures, where the proof π_i^{pk} is a concrete zero-knowledge and simulation-extractable proof system rather than a call to $\mathcal{F}_{\mathrm{nizk}}$, because its statement must remain private until Round 2 (Section D.4) (see Theorem 1 in Section 4 for the formal statement).

$\Pi^{t,n}_{\mathsf{DKG}}$

Target security is UC realization of functionality $\mathcal{F}^{t,n}_{\mathsf{DKG}}$ with $t \leq n$.

We assume erasure, authenticated and reliable broadcast, and the batched hashed ElGamal encryption $(\mathsf{Enc}, \mathsf{Dec})$ of Section 3. $\mathsf{NIZK.PoK}$ denotes straight-line extractable NIZK PoK, while NIZK is a standard NIZK, without extraction. Proofs for L^{VSS}, L^{comeq}, L^{dec} and L^{enc} are modeled by $\mathcal{F}_{\text{nizk}}$, while the proof for L^{pk} is a concrete proof system (Section D.4), since its statement must stay private until Round 2. Every proof statement additionally contains the index i of the proving party $\mathcal{P}_i$, and verifiers take that index from the sender identity reported by $\mathcal{F}_{\text{bc}}$. We omit that from the notation below for readability.

CRS: Group parameters $(\mathbb{G}, p, g)$, external Pedersen commitment parameter $h \in \mathbb{G}$, internal Pedersen commitment parameter $v \in \mathbb{G}$.

PKI model: The functionality is tagged by participant set $\mathcal{P}$, which translates into encryption public key pk_i^E for each $\mathcal{P}_i \in \mathcal{P}$ ($\mathcal{P}_i$'s private key is denoted sk_i^E).

Round 1. Each $\mathcal{P}_i$ generates a random Pedersen VSS, a public key contribution pk_i, a NIZK PoK of the corresponding secret key contribution $f_i(0)$, and commits to the randomness used in vk_i :

1. Pick two random $(t-1)$-degree polynomials $f_i(x) = \sum_{k=0}^{t-1} a_{i,k}x^k$ and $f'_i(x) = \sum_{k=0}^{t-1} b_{i,k}x^k$, let $a_i = (a_{i,k})_{k=0}^{t-1}$ and $b_i = (b_{i,k})_{k=0}^{t-1}$.
2. Create VSS information, $\mathsf{PedVSS}_i \leftarrow \{c_{i,k} = g^{a_{i,k}} \cdot v^{b_{i,k}}\}_{k=0}^{t-1}$.
3. Pick $\omega_i, r_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and create commitment $D_i = g^{\omega_i} \cdot v^{r_i}$.
4. Set $\pi_i^{\mathsf{VSS}} \leftarrow \mathsf{NIZK.PoK}[L^{\mathsf{VSS}}_{g,v}]((a_i, b_i), \omega_i, r_i), (\mathsf{PedVSS}_i, D_i))$
5. Set $\mathsf{pk}_i \leftarrow g^{f_i(0)}$ and $\pi_i^{\mathsf{pk}} \leftarrow \mathsf{NIZK}[L^{\mathsf{pk}}_{g,v}]((a_{i,0}, b_{i,0}), (\mathsf{pk}_i, c_{i,0}))$.
6. $\forall\ j \neq i$ set $\phi_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j^E, (f_i(j), f'_i(j))$ for pk_j^E retrieved from PKI for $\mathcal{P}_j$.
7. Broadcast $(\mathsf{PedVSS}_i, D_i, \pi_i^{\mathsf{VSS}}, \{\phi_{i,j}\}_{j\neq i})$, and erase all ephemeral state except $(s_{i,i}, s'_{i,i}) \leftarrow (f_i(i), f'_i(i))$ and $(\mathsf{pk}_i, \pi_i^{\mathsf{pk}}, \omega_i, r_i)$.

Round 2. Each $\mathcal{P}_i$ broadcasts pk_i and re-commits to its key share:

1. If, for some j, π_j^{VSS} or the proof of knowledge attached to $\{\phi_{j,k}\}_{k\neq j}$ doesn't verify, then output (abort, j) for every such j and stop.
2. Parse $\{c_{j,k}\}_{k=0}^{t-1} \leftarrow \mathsf{PedVSS}_j$ and set $(s_{j,i}, s'_{j,i}) \leftarrow \mathsf{Dec}(\mathsf{sk}_i^E, \phi_{j,i})\ \forall\ j \in \mathcal{P} \setminus \{i\}$.
3. If $g^{s_{j,i}} \cdot v^{s'_{j,i}} \neq \prod_{k=0}^{t-1}(c_{j,k})^{(i^k)}$ for some j, then for every such j broadcast $(j, s_{j,i}, s'_{j,i}, \pi_{i,j}^{\mathsf{dec}})$ where $\pi_{i,j}^{\mathsf{dec}} \leftarrow \mathsf{NIZK}[L^{\mathsf{dec}}](\mathsf{sk}_i^E, (\mathsf{pk}_i^E, (s_{j,i}, s'_{j,i}), \phi_{j,i}))$, skip the remaining steps of Round 2, and proceed to Output.
4. Set $(s_i, s'_i) \leftarrow (\sum_{j\in\mathcal{P}} s_{j,i}, \sum_{j\in\mathcal{P}} s'_{j,i})$ and $C_i^* \leftarrow \prod_{j\in\mathcal{P}} \prod_{k=0}^{t-1}(c_{j,k})^{(i^k)}$
5. Set $\mathsf{vk}_i \leftarrow g^{s_i} \cdot h^{\omega_i}$ and $\pi_i^{\mathsf{comeq}} \leftarrow \mathsf{NIZK}[L^{\mathsf{comeq}}_{g,h}]((s_i, s'_i, \omega_i, r_i), (C_i^*, \mathsf{vk}_i, D_i))$.
6. Broadcast $(pk_i, \pi_i^{\mathsf{pk}}, vk_i, \pi_i^{\mathsf{comeq}})$.

Output. Each $\mathcal{P}_i$ processes all received messages:

1. If any $\mathcal{P}_{i'}$ broadcasts $(j, s_{j,i'}, s'_{j,i'}, \pi_{i',j}^{\mathsf{dec}})$ s.t. $\pi_{i',j}^{\mathsf{dec}}$ verifies and $g^{s_{j,i'}} \cdot v^{s'_{j,i'}} \neq \prod_{k=0}^{t-1}(C_{j,k})^{((i')^k)}$ then output (abort, j) for every such j and stop.
2. If, for some j, π_j^{pk} or π_j^{comeq} doesn't verify, then output (abort, j) for every such j and stop.
3. Output $(sk_i, \omega_i, \mathsf{pk}, \mathsf{vk})$ for $\mathsf{sk}_i = s_i$, $\mathsf{pk} = \prod_{j\in\mathcal{P}} \mathsf{pk}_j$, and $\mathsf{vk} = (\mathsf{vk}_j)_{j\in\mathcal{P}}$.

Fig. 4: $\mathsf{Janus}_{\mathsf{2R}}$: Two-Round Distributed Key Generation Protocol that realizes $\mathcal{F}^{t,n}_{\mathsf{DKG}}$

Round 1: Commit Each party samples two random degree-$(t-1)$ polynomials

$$f_i(x) = \sum_{k=0}^{t-1} a_{i,k} x^k \quad \text{and} \quad f'_i(x) = \sum_{k=0}^{t-1} b_{i,k} x^k,$$

where f_i carries the secret and f'_i serves as its blinding polynomial. The coefficients of the two polynomials are combined into Pedersen commitments $c_{i,k} \leftarrow g^{a_{i,k}} \cdot v^{b_{i,k}}$ using an independent generator v. In addition, each party samples its partial verification-key blinding randomness $\omega_i \leftarrow\!\$\ \mathbb{Z}_p$ and commits to ω_i via $D_i \leftarrow g^{\omega_i} \cdot v^{r_i}$ with auxiliary randomness $r_i \leftarrow\!\$\ \mathbb{Z}_p$. Party $\mathcal{P}_i$ then computes proofs of knowledge for the coefficients $(a_{i,k}, b_{i,k})_{k=0}^{t-1}$ and (ω_i, r_i), pre-computes its public key contribution $\mathsf{pk}_i = g^{f_i(0)}$ with a NIZK proof π_i^{pk} certifying consistency with $c_{i,0}$. Each party broadcasts the commitments, proofs of knowledge, and encrypted share evaluations $(f_i(j), f'_i(j))$ for all $j \in \mathcal{P}$. Immediately after broadcasting, $\mathcal{P}_i$ erases f_i, f'_i, and the encryption randomness r, retaining only its local share $(f_i(i), f'_i(i))$ and the pre-computed tuple $(\mathsf{pk}_i, \pi_i^{\mathsf{pk}}, \omega_i, r_i)$.

Round 2: Open and Aggregate After verifying all first-round proofs, each party $\mathcal{P}_i$ decrypts its inbound ciphertexts to obtain $(f_j(i), f'_j(i))$ for each $j \in \mathcal{P}$, and checks consistency against the implicit point commitments derived from the public VSS commitments $c_{j,k}$. Party $\mathcal{P}_i$ computes its secret-key share and the blinding randomness of the VSS commitments $s_i = \sum_{j\in\mathcal{P}} f_j(i), s'_i = \sum_{j\in\mathcal{P}} f'_j(i)$. It computes its partial verification key $\mathsf{vk}_i \leftarrow g^{s_i} \cdot h^{\omega_i}$ with respect to the standard Pedersen base h, where ω_i is the value previously committed to in D_i. Only now $\mathcal{P}_i$ broadcasts the unblinded public key contribution $\mathsf{pk}_i = g^{f_i(0)}$ with proof π_i^{pk} (certifying consistency with $c_{i,0}$), and its partial verification key vk_i with proof π_i^{comeq} certifying consistency between vk_i, D_i and the aggregated VSS commitments. All parties aggregate the individual contributions to obtain the master public key $\mathsf{pk} = \prod_{j\in\mathcal{P}} \mathsf{pk}_j = g^{\mathsf{sk}}$.

Identifiable Aborts Identifiable aborts handle two distinct classes of misbehavior: publicly observable and privately detectable deviations. First, if any party $\mathcal{P}_j$ broadcasts an invalid NIZK proof, all parties observe the failure directly from the public transcript and trigger an immediate global abort (abort, j) without any further interaction. Second, if a corrupted party sends malformed point-to-point ciphertexts in Round 1, the affected recipient $\mathcal{P}_i$ detects the inconsistency privately upon decryption. Party $\mathcal{P}_i$ resolves this by broadcasting a proof of correct decryption (revealing $(g^r)^{\mathsf{sk}_i^E}$ with a DLog equality proof) as its Round 2 message. Combined with the broadcast transcript, this enables all participants to publicly verify the decrypted share against the sender's global VSS commitments and attribute the abort to $\mathcal{P}_j$ without requiring additional communication rounds.

We formally define $\mathsf{Janus}_{\mathsf{2R}}$ in Figure 4.

3.2 $\mathsf{Janus}_{\mathsf{1R}}$

Our single-round protocol, $\mathsf{Janus}_{\mathsf{1R}}$, UC-realizes $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$ (Figure 3) under the CDH assumption (Definition 1) in the $(\mathcal{F}_{\mathrm{bc}}, \mathcal{F}_{\mathrm{ro}}, \mathcal{F}_{\mathrm{nizk}}, \mathcal{F}_{\mathrm{pki}})$-hybrid world assuming secure erasures (see Theorem 2 in Section 4 for the formal statement).

Round 1: Broadcast and Local Output Each party $\mathcal{P}_i$ samples a random degree-$(t-1)$ polynomial $f_i(x) = \sum_{k=0}^{t-1} a_{i,k} x^k$, and commits directly to its point evaluations:

$$c_{i,j} \leftarrow g^{f_i(j)} h^{\omega_{i,j}},$$

where each blinding factor $\omega_{i,j} \leftarrow\!\$\ \mathbb{Z}_p$ is sampled independently per recipient. In addition, each party computes its public key contribution $\mathsf{pk}_i = g^{f_i(0)}$ alongside a proof of knowledge for the coefficients $(a_{i,k})_{k=0}^{t-1}$ and the blinding factors $(\omega_{i,j})_{j\in\mathcal{P}}$, certifying consistency between pk_i and $f_i(0)$. Each party broadcasts the commitments, proofs of knowledge, and encrypted share evaluations $(f_i(j), \omega_{i,j})$ for all $j \in \mathcal{P}$. Immediately

$\Pi^{t,n}_{\mathsf{S\text{-}DKG}}$

Target security is UC realization of functionality $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ with any $t \leq n$. The protocol is 1-round only in *optimistic execution*.

We assume erasure, authenticated and reliable broadcast, and the batched hashed ElGamal encryption $(\mathsf{Enc}, \mathsf{Dec})$ of Section 3. $\mathsf{NIZK.PoK}$ denotes straight-line extractable NIZK, while NIZK is a non-extractable NIZK. Every proof statement contains the index i of the proving party $\mathcal{P}_i$, and verifiers take that index from the sender identity reported by $\mathcal{F}_{\mathrm{bc}}$. We omit that from the notation below.

CRS: Group parameters $(\mathbb{G}, p, g)$, Pedersen parameter $h \in \mathbb{G}$.

PKI model: the functionality is tagged by participant set $\mathcal{P}$, which translates into encryption public key pk_i^E for each $\mathcal{P}_i \in \mathcal{P}$ ($\mathcal{P}_i$'s private key is denoted sk_i^E).

Initiate Key Generation. Each P_i generates a random Pedersen VSS, a public key contribution pk_i, and a NIZK PoK of the well-formedness of all messages:

1. Pick $(t-1)$-degree polynomial $f_i(x) = \sum_{k=0}^{t-1} a_{i,k} x^k$.
2. Sample $\{\omega_{i,j}\}_{j\in\mathcal{P}}$ uniformly at random from $\mathbb{Z}_p^*$.
3. Create commitments, $\mathsf{ComVSS}_i \leftarrow \{c_{i,j}\}_{j\in\mathcal{P}}$ for $c_{i,j} \leftarrow g^{f_i(j)} \cdot h^{\omega_{i,j}}$.
4. Set $\mathsf{pk}_i \leftarrow g^{f_i(0)}$.
5. Set $\pi_i^* \leftarrow \mathsf{NIZK.PoK}_g((\{a_{i,k}\}_{k=0}^{t-1}, \{\omega_{i,j}\}_{j\in\mathcal{P}}), \mathsf{pk}_i, \mathsf{ComVSS}_i)$.
6. For all $j \neq i$ set $\phi_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j^E, (f_i(j), \omega_{i,j}))$.
7. Broadcast $(\mathsf{pk}_i, \mathsf{ComVSS}_i, \pi_i^*, \{\phi_{i,j}\}_{j\neq i})$.
8. Erase everything except $(s_{i,i}, s'_{i,i})$ with $s_{i,i} \leftarrow f_i(i)$, $s'_{i,i} \leftarrow \omega_{i,i}$.

Output Key Generation. Each party processes all received messages:

1. If, for some $\mathcal{P}_j$, π_j^* or the proof of knowledge attached to $\{\phi_{j,k}\}_{k\neq j}$ doesn't verify, then output (abort, j) for every such j and stop.
2. Parse $\{c_{j,k}\}_{k\in\mathcal{P}} \leftarrow \mathsf{ComVSS}_j$ and set $(s_{j,i}, s'_{j,i}) \leftarrow \mathsf{Dec}(\mathsf{sk}_i^E, \phi_{j,i})$ for all $j \neq i$.
3. If $g^{s_{j,i}} \cdot h^{s'_{j,i}} \neq c_{j,i}$ for some j, then for every such j broadcast $(j, s_{j,i}, s'_{j,i}, \pi_{i,j}^{\mathsf{dec}})$ where $\pi_{i,j}^{\mathsf{dec}} \leftarrow \mathsf{NIZK}[L^{\mathsf{dec}}](\mathsf{sk}_i^E, (\mathsf{pk}_i^E, (s_{j,i}, s'_{j,i}), \phi_{j,i}))$, output (abort, j), and stop.
4. Set $(s_i, s'_i) \leftarrow (\sum_{j\in\mathcal{P}} s_{j,i}, \sum_{j\in\mathcal{P}} s'_{j,i})$.
5. Set $\mathsf{vk}_k \leftarrow \prod_{j\in\mathcal{P}} c_{j,k}$ and $\mathsf{pk} = \prod_{j\in\mathcal{P}} \mathsf{pk}_j$.
6. Output $(\mathsf{sk}_i, \omega_i, \mathsf{pk}, \mathsf{vk})$ for $\mathsf{sk}_i = s_i$, $\omega_i = s'_i$ and $\mathsf{vk} = \{\mathsf{vk}_j\}_{j\in\mathcal{P}}$.

Report Handling. If any $\mathcal{P}_{i'}$ broadcasts $(j, s_{j,i'}, s'_{j,i'}, \pi_{i',j}^{\mathsf{dec}})$ s.t. $\pi_{i',j}^{\mathsf{dec}}$ verifies and $g^{s_{j,i'}} \cdot h^{s'_{j,i'}} \neq c_{j,i'}$ then output $(\mathsf{reportAbort}, i', j)$.

Fig. 5: $\mathsf{Janus}_{\mathsf{1R}}$: Single-Round Distributed Key Generation Protocol that realizes $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$.

after broadcasting, $\mathcal{P}_i$ erases $f_i(x)$, the blinding factors $\omega_{i,j}$, and the encryption randomness r, retaining only its local share $(f_i(i), \omega_{i,i})$. Upon receiving all n broadcasts and verifying each π_j^* and the proof of knowledge attached to $\mathcal{P}_j$'s ciphertexts, each party $\mathcal{P}_i$ computes the global public key $\mathsf{pk} = \prod_{j\in\mathcal{P}} \mathsf{pk}_j$ and derives all partial verification keys non-interactively as

$$\mathsf{vk}_{i'} = \prod_{j\in\mathcal{P}} c_{j,i'} = g^{\sum_{j\in\mathcal{P}} f_j(i')} h^{\sum_{j\in\mathcal{P}} \omega_{j,i'}} = g^{s_{i'}} h^{\omega_{i'}}.$$

Each party $\mathcal{P}_i$ decrypts its inbound ciphertexts to obtain $(f_j(i), \omega_{j,i})$ for each $j \in \mathcal{P}$ and checks consistency against the public VSS commitment $c_{j,i}$. Party $\mathcal{P}_i$ computes its secret-key share and the blinding randomness of its key share commitment $s_i = \sum_{j\in\mathcal{P}} f_j(i), s'_i = \sum_{j\in\mathcal{P}} \omega_{j,i}$.

Report-Based Identifiable Aborts Identifiable aborts handle two distinct classes of misbehavior:

– **Public misbehavior (Global Abort):** If any broadcast proof π_j^*, or the proof of knowledge attached to $\mathcal{P}_j$'s ciphertexts, is invalid, all parties detect the failure directly from the public transcript and immediately output a global abort (abort, j) without further interaction.
– **Private misbehavior (Local Abort):** If party $\mathcal{P}_i$ decrypts an inconsistent share, it locally outputs (abort, j) and broadcasts a proof of correct decryption (revealing $(g^r)^{\mathsf{sk}_i^E}$ with a DLog equality proof). Combined with the broadcast transcript, this forms a universally verifiable proof for external arbiters or smart contracts. Non-aborting honest parties complete the protocol locally while still identifying the reporting party $\mathcal{P}_i$ and the corrupted sender $\mathcal{P}_j$ (see Section 2 for more details on report-based identifiable abort guarantees).

We formally define $\mathsf{Janus}_{\mathsf{1R}}$ in Figure 5.

4 Security Analysis

In this section, we prove that our protocols securely realize the corresponding ideal functionalities in the UC framework under the CDH assumption. We write $\mathcal{P} \leq \mathcal{F}$ if protocol $\mathcal{P}$ UC-realizes functionality $\mathcal{F}$, that is, if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no polynomial-time environment $\mathcal{E}$ distinguishes an interaction with $\mathcal{P}$ and $\mathcal{A}$ from an interaction with $\mathcal{F}$ and $\mathcal{S}$ with more than negligible advantage. Corruptions are adaptive, and we treat each broadcast of an honest party together with the erasure that follows it as one atomic step, so a party is corrupted either before it broadcasts or after it has erased.

Definition 1 (Computational Diffie-Hellman (CDH)). *Let $\mathbb{G}$ be a cyclic group of prime order p generated by g. The* Computational Diffie-Hellman *(CDH) assumption holds in $\mathbb{G}$ if for every PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathsf{CDH}}(\mathcal{A}) := \mathsf{Pr}[\mathcal{A}(\mathsf{g}, \mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}}) = \mathsf{g}^{\mathsf{ab}} \mid \mathsf{a}, \mathsf{b} \leftarrow\!\$\, \mathbb{Z}_{\mathsf{p}}] \leq \mathsf{negl}(\lambda).$$

Theorem 1. *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\Pi_{\mathsf{DKG}}^{t,n}$ be our protocol and $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ be the ideal* DKG *functionality as defined in Figure 2. Let $n, t \in \mathbb{N}$ with $t \leq n$ be the number of parties, respectively the number of parties necessary to reconstruct the key generated during the DKG execution. Suppose the* CDH *assumption holds and let Π be a proof system for L^{pk} that is zero-knowledge (Definition 2) and simulation extractable (Definition 3), then*

$$\Pi_{\mathsf{DKG}}^{t,n} \leq \mathcal{F}_{\mathsf{DKG}}^{t,n}$$

in the $(\mathcal{F}_{bc}, \mathcal{F}_{\mathrm{ro}}, \mathcal{F}_{\mathrm{nizk}}, \mathcal{F}_{pki})$-hybrid world, assuming secure erasure, against an adversary that adaptively corrupts up to $t-1$ parties, which is a dishonest majority whenever $t-1 > n/2$. Here $\mathcal{F}_{\mathrm{nizk}}$ models the proofs for L^{VSS}, L^{comeq}, L^{dec} and L^{enc} only, while the proof for L^{pk} is the concrete system Π, for the reason given in Section D.4.

We demonstrate the UC security of the protocol by constructing a simulator $\mathcal{S}$ and showing through a sequence of hybrid games that no polynomial-time environment $\mathcal{E}$ can distinguish between the real execution and the ideal world.

The simulator $\mathcal{S}$. The simulator $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ and with the environment $\mathcal{E}$ via the network interface. Its strategy is summarized as follows:

– **Setup:** $\mathcal{S}$ retrieves the group parameters from $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ and samples the internal Pedersen parameter $v \leftarrow g^{\tau}$, retaining τ as a trapdoor.
– **Honest Party Init:** For honest parties, $\mathcal{S}$ follows the protocol, generating perfectly hiding Pedersen commitments, the Round 1 NIZK.PoKs, and each pk_i with its proof π_i^{pk}. However, it replaces honest-to-honest ciphertexts with uniformly random values, preparing them for later equivocation.

– **Finish Init:** $\mathcal{S}$ extracts the adversarial secret-sharing polynomials and blinding factors from the adversarial $\mathsf{NIZK.PoKs}$[7]. It submits these extracted values to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$, which returns the global public key pk and verification keys vk. To ensure consistency, $\mathcal{S}$ overwrites the public key contribution pk_{i^*} of the least-indexed honest party $\mathcal{P}_{i^*}$ to align with the global pk, and obtains the corresponding $\pi_{i^*}^{\mathsf{pk}}$ from the zero-knowledge simulator $\mathcal{S}_{\mathsf{nizk}}$ of Π rather than from $\mathcal{F}_{\mathrm{nizk}}$[8] Only that one contribution and that one proof are changed, and every other honest party keeps what Honest Party Init produced.
– **Adaptive Corruption:** Upon the corruption of an honest party, $\mathcal{S}$ receives the true secret share and blinding factor from $\mathcal{F}_{\mathsf{DKG}}^{t,n}$. Using the secure erasure property, the random oracle programmability, and the trapdoor τ, $\mathcal{S}$ shifts the internal state so that the previously broadcast (dummy) commitments consistently open to the true values retrieved from the functionality.

Sequence of games. We structure the indistinguishability proof using a sequence of hybrid games, starting from the real-world execution and ending in the ideal-world simulation.

Game 0 (Real World) This is the real-world execution of the protocol $\Pi_{\mathsf{DKG}}^{t,n}$ with the environment $\mathcal{E}$. All honest parties hold real inputs and compute their messages exactly as prescribed by the protocol.

Game 1 (Simulated Public-Key Proofs) We now stop proving L^{pk} honestly for one party. Write $\mathcal{P}_{i^*}$ for the least-indexed party still uncorrupted at the end of Round 1, whose contribution the ideal execution will later overwrite. Its proof $\pi_{i^*}^{\mathsf{pk}}$ is obtained from the zero-knowledge simulator $\mathcal{S}_{\mathsf{nizk}}$ of Π rather than from $\mathcal{P}$, while the statement $(\mathsf{pk}_{i^*}, c_{i^*,0})$ and the witness $(a_{i^*,0}, b_{i^*,0})$ are still the real ones. Every other honest party proves as the protocol prescribes, so a single invocation of $\mathcal{S}_{\mathsf{nizk}}$ separates this game from Game 0. Random-oracle queries are routed through $\mathcal{S}_{\mathsf{nizk},1}$, which is consistent with $\mathcal{F}_{\mathrm{ro}}$ being local and programmable. Any environment that distinguishes Game 1 from Game 0 yields an adversary against Definition 2 of the same advantage, so $\mathsf{Adv}_{\Pi}^{\mathsf{zk}}$ bounds this hop. For the Fiat–Shamir instantiation of Section D.4 the hop is in fact perfect, up to the probability q_{ro}/p that $\mathcal{S}_{\mathsf{nizk}}$ fails to program a point $\mathcal{A}$ has already queried. This game is well-defined only because $\pi_{i^*}^{\mathsf{pk}}$ never reaches $\mathcal{A}$ before $\mathcal{P}_{i^*}$'s Round 2 broadcast, and had the proof been routed through $\mathcal{F}_{\mathrm{nizk}}$, the statement would already have been disclosed in Round 1.

Game 2 (Extraction) Game 2 is identical to Game 1, except that at the end of the execution it extracts a witness from every accepting proof of a party that was corrupted when it sent its Round 1 message, and aborts if one of these extractions fails. For π_j^{VSS} and π_j^{comeq}, the witness is read off the internal table of $\mathcal{F}_{\mathrm{nizk}}$, which holds one for every such proof, so this part of the change is perfect. For π_j^{pk}, the game runs an extractor $\mathcal{E}_{\mathsf{nizk}}$ on the final state of $\mathcal{S}_{\mathsf{nizk}}$, the statement $(\mathsf{pk}_j, c_{j,0})$ and the proof π_j^{pk}. Definition 3 fixes no extraction mechanism, and even a rewinding extractor is admissible here, since $\mathcal{E}_{\mathsf{nizk}}$ is called by the reduction, which is a stand-alone algorithm running the entire experiment and may rewind $\mathcal{E}$ and $\mathcal{A}$. The simulator $\mathcal{S}$ of the ideal execution may not rewind, and it never invokes $\mathcal{E}_{\mathsf{nizk}}$. Which exact extraction mechanism is therefore a property of the instantiation. We show that an execution in which this extraction fails yields a reduction $\mathcal{B}$ that breaks Definition 3, and we let $\mathcal{E}_{\mathsf{nizk}}$ be the extractor that Definition 3 guarantees for $\mathcal{B}$. The reduction $\mathcal{B}$ plays Game 1 and follows the game description throughout, forwarding the random-oracle queries and the proving query for $\mathcal{P}_{i^*}$, which Game 1 already answers with $\mathcal{S}_{\mathsf{nizk}}$, to its oracles $\mathcal{S}_{\mathsf{nizk},1}$ and $\mathcal{S}'_{\mathsf{nizk},2}$. Furthermore, at the end of the execution, it picks one of the at most $t-1$ parties above uniformly at random and outputs

[7] For valid proofs, $\mathcal{S}$ can lookup the correct witness from the internal table of $\mathcal{F}_{\mathrm{nizk}}$, defined in Section D.4. $\mathcal{S}$ has access to that table, because $\mathcal{F}_{\mathrm{nizk}}$ is simulated by $\mathcal{S}$. Since every statement contains the index of the proving party, $\mathcal{S}$ adds witness-free entries only for statements of parties that are honest when they broadcast, so the table holds a witness for every accepting proof of a party that was corrupted when it broadcast, and a copied honest proof does not verify for any other party.

[8] The remaining simulated proofs, for L^{VSS}, L^{comeq} and L^{dec}, are handled differently. There, $\mathcal{S}$ adds a statement/proof pair without a witness to the internal table of $\mathcal{F}_{\mathrm{nizk}}$, which it may do because $\mathcal{F}_{\mathrm{nizk}}$ is simulated by $\mathcal{S}$ and because the statements in question are true. This route is unavailable for L^{pk}, since $\mathcal{F}_{\mathrm{nizk}}$ would publish the statement one round too early.

that party's statement and proof π_j^{pk}. The extractor $\mathcal{E}_{\mathsf{nizk}}$ receives only the state of $\mathcal{S}_{\mathsf{nizk}}$, the statement and the proof, so its output on each of these proofs is independent of the choice of $\mathcal{B}$, and $\mathcal{B}$ outputs a proof on which $\mathcal{E}_{\mathsf{nizk}}$ fails with probability at least $1/(t-1)$ whenever Game 2 aborts. The output of $\mathcal{B}$ is never in $\mathcal{Q}$, because $\mathcal{Q}$ contains only the proof of $\mathcal{P}_{i^*}$, which was honest at the end of Round 1, and every statement contains the index of its prover. Hence, the environment's advantage in distinguishing Game 2 from Game 1 is at most $(t-1) \cdot \mathsf{Adv}_{\Pi}^{\mathsf{sim\text{-}ext}}(\mathcal{B})$.

Game 3 (Pedersen Binding) Game 3 is identical to Game 2, except that it aborts if a party $\mathcal{P}_j$ that was corrupted when it sent its Round 1 message has opened one of its commitments under (g, v) in two different ways. The reference opening is the witness extracted from π_j^{VSS}, namely the coefficient vectors $a_j = (a_{j,k})_{k=0}^{t-1}$ and $b_j = (b_{j,k})_{k=0}^{t-1}$, which open the t commitments in PedVSS_j and determine the polynomials f_j and f'_j, and (ω_j, r_j) for D_j. The game compares it with (i) the opening of $c_{j,0}$ extracted from π_j^{pk} in Game 2, (ii) the openings of D_j and of C_j^* in the witness of π_j^{comeq}, where the reference opening of C_j^* is $(\sum_{k\in\mathcal{P}} f_k(j), \sum_{k\in\mathcal{P}} f'_k(j))$, and (iii) every share $(s_{j,i}, s'_{j,i})$ that an honest party $\mathcal{P}_i$ accepts as an opening of $\prod_{k=0}^{t-1}(c_{j,k})^{(i^k)}$, whose reference opening is $(f_j(i), f'_j(i))$. We show that an execution in which Game 3 aborts yields a reduction $\mathcal{B}'$ that computes $\log_g(v)$. The reduction $\mathcal{B}'$ plays Game 2 and follows the game description throughout, deviating in exactly one place. It sets the internal Pedersen parameter v to its dLog challenge Y, which is distributed like an honestly sampled v. This is possible because no step of Game 2 uses $\log_g(v)$. When the abort occurs, $\mathcal{B}'$ holds two distinct openings $(x, y) \neq (x', y')$ of one commitment, so $g^x Y^y = g^{x'} Y^{y'}$, which forces $y \neq y'$ and reveals $\log_g(Y) = (x - x')/(y' - y)$. Hence, the environment's advantage in distinguishing Game 3 from Game 2 is at most $\mathsf{Adv}_{\mathcal{B}'}^{\mathsf{dLog}}$.

Game 4 (Ciphertext Consistency) In the next transition, we rule out inconsistencies with honest-to-honest ciphertexts. The simulator aborts if $\mathcal{E}$ queries $\mathcal{F}_{\mathrm{ro}}$ with a valid decryption value $u = (\mathsf{pk}_i^E)^r$ for a ciphertext sent by a party that was honest at the time of encryption to a party that is still honest. We show by reduction that if the abort happens, we can solve the CDH problem. $\mathcal{S}$ guesses which honest party $\mathcal{P}^*$ is the receiver of the ciphertext for which the abort happens, which is correct with probability at least $\frac{1}{\mathsf{n}}$. We embed a CDH challenge $(A = g^a, B = g^b)$ such that the public key of the guessed honest party is set to A and the ephemeral randomness in the ciphertext is set to B. Specifically, all remaining honest parties sample a random exponent ψ_i and use B^{ψ_i} as their ephemeral randomness.

The ciphertext sent from an honest party $\mathcal{P}_i$, except $\mathcal{P}^*$, to a malicious party $\mathcal{P}_j$ is computed as $\phi_{i,j} \leftarrow (B^{\psi_i}, \mathsf{H}(B^{\psi_i \cdot \mathsf{sk}_j^E}) \oplus msg)$ with sk_j^E taken from $\mathcal{F}_{\mathrm{pki}}$ as defined in Section D.5. Decrypting ciphertexts sent by a malicious party $\mathcal{P}_j$ to $\mathcal{P}^*$ is done by extracting $\mathcal{P}_j$'s ephemeral randomness r from the proof that is part of the ciphertext, read off the internal table of $\mathcal{F}_{\mathrm{nizk}}$ for L^{enc}, and computing the input to $\mathcal{F}_{\mathrm{ro}}$ as $(\mathsf{pk}_{\mathcal{P}^*}^E)^r$. The same technique is used when generating a complaint against $\mathcal{P}_j$ if it sent an incorrect ciphertext. The proofs for L^{enc} of honest senders, whose randomness B^{ψ_i} has an unknown discrete logarithm, are added to that table without a witness, which is possible because $\mathcal{F}_{\mathrm{nizk}}$ is simulated and the statements are true.

In addition, each ciphertext component c_j sent between two honest parties is replaced by a uniformly random string of the same length, honest recipients obtain the corresponding plaintext m_j directly instead of querying $\mathcal{F}_{\mathrm{ro}}$, and $\mathcal{F}_{\mathrm{ro}}$ is programmed at $(\mathsf{pk}_j^E)^r$ to $c_j \oplus m_j$ when the recipient $\mathcal{P}_j$ is corrupted. Unless the abort above occurs, $\mathcal{F}_{\mathrm{ro}}$ has not been queried at $(\mathsf{pk}_j^E)^r$ before this programming, so in the previous game $c_j = \mathcal{F}_{\mathrm{ro}}((\mathsf{pk}_j^E)^r) \oplus m_j$ is already uniformly random and independent of the remaining view, and the replacement does not change the distribution.

Extracting the query that includes the solution to the CDH challenge requires $\mathcal{S}$ to guess a query to $\mathcal{F}_{\mathrm{ro}}$, which succeeds with probability at least $\frac{1}{q_{\mathrm{ro}}}$, where q_{ro} is the number of queries to $\mathcal{F}_{\mathrm{ro}}$. The environment's advantage in distinguishing Game 4 from Game 3 is bounded by the advantage of solving CDH, which is negligible.

Game 5 (Ideal Execution) In this final game, the simulator $\mathcal{S}$ interacts fully with the ideal functionality $\mathcal{F}_{\mathsf{DKG}}^{t,n}$, replacing the real-world execution. $\mathcal{S}$ extracts the adversarial shares, submits them to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$, and

receives the global public key pk, which it matches by overwriting $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk}/\prod_{j\neq i^*}\mathsf{pk}_j$ for the least-indexed honest party $\mathcal{P}_{i^*}$. This overwrite changes the statement that $\mathcal{S}_{\mathsf{nizk}}$ is asked to prove, but not its distribution. In Game 4 the value $\mathsf{pk}_{i^*} = g^{a_{i^*,0}}$ is uniform in $\mathbb{G}$ and independent of $\{\mathsf{pk}_j\}_{j\neq i^*}$, because $a_{i^*,0}$ is a uniform coefficient of an honest polynomial. In Game 5, $\mathcal{F}^{t,n}_{\mathsf{DKG}}$ draws f uniformly among the degree-$(t-1)$ polynomials that agree with the $|\mathrm{Corr}| \leq t-1$ extracted shares, so $\mathsf{sk} = f(0)$ and hence pk are uniform and independent of those shares. The derived pk_{i^*} is again uniform and independent of $\{\mathsf{pk}_j\}_{j\neq i^*}$. The two joint distributions thus coincide exactly. Switching the proving wrapper from $\mathcal{S}_{\mathsf{nizk},2}$ to $\mathcal{S}'_{\mathsf{nizk},2}$ is likewise free, since the two differ only in a relation check that always passes here.

A core component of this hop is ensuring that the simulator can provide a consistent internal state for any honest party corrupted mid-execution. When a corruption occurs after protocol initialization, $\mathcal{S}$ retrieves the true secret material (s_i^*, ω_i^*) from $\mathcal{F}^{t,n}_{\mathsf{DKG}}$. $\mathcal{S}$ computes a shift $\Delta \leftarrow s_i^* - \sum_{j\in\mathcal{P}} f_j(i)$ and leverages the trapdoor τ to adjust the blinding factors dynamically, providing the adversary with the equivocated state $(f_i(i)+\Delta, f'_i(i)-\Delta/\tau)$. To complete this state equivocation, $\mathcal{S}$ retroactively programs $\mathcal{F}_{\mathrm{ro}}$ so that the originally broadcast dummy ciphertexts addressed to the newly corrupted party decrypt exactly to shares consistent with the adjusted state.

To conclude, the programming of the random oracle does not introduce any inconsistencies, as Game 4 already established that the probability that $\mathcal{S}$ cannot program $\mathcal{F}_{\mathrm{ro}}$, because $\mathcal{F}_{\mathrm{ro}}$ has already been queried before on that value, is bound by the probability of solving the CDH problem. Furthermore, replacing real proofs with simulated ones in $\mathcal{F}_{\mathrm{nizk}}$ is perfectly indistinguishable due to $\mathcal{S}$'s ability to simulate $\mathcal{F}_{\mathrm{nizk}}$ internally, and the proof for L^{pk} of $\mathcal{P}_{i^*}$ was already replaced in Game 1. Because the Pedersen commitments are perfectly hiding and $\mathcal{S}$'s state equivocation provides a perfectly consistent internal view to the adversary, Game 5 is distributed identically to Game 4 as long as none of the aborts of Games 2, 3 and 4 occurs, so this hop adds no term to the bound, and Game 5 represents the final ideal-world execution.

The overall distinguishing advantage of the environment $\mathcal{E}$ between the real-world execution of $\Pi^{t,n}_{\mathsf{DKG}}$ and the ideal-world interaction with $\mathcal{F}^{t,n}_{\mathsf{DKG}}$ and $\mathcal{S}$ is bounded by the probability of breaking dLog or CDH together with the two advantage terms of Π. Specifically:

$$\mathsf{Adv}^{\mathrm{dist}}_{\mathcal{E}} \leq \mathsf{Adv}^{\mathsf{dLog}}_{\mathcal{B}'} + (t-1)\cdot \mathsf{Adv}^{\mathsf{sim\text{-}ext}}_{\Pi}(\mathcal{B}) + \mathsf{n}\cdot q_{\mathrm{ro}}\cdot \mathsf{Adv}^{\mathsf{CDH}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{zk}}_{\Pi}(\mathcal{A}) + \mathrm{negl}(\lambda)$$

Under the CDH assumption and the two properties of Π, these advantages are negligible. Therefore, $\Pi^{t,n}_{\mathsf{DKG}}$ securely realizes $\mathcal{F}^{t,n}_{\mathsf{DKG}}$ in the UC framework. Instantiating Π with Fiat–Shamir makes $\mathsf{Adv}^{\mathsf{sim\text{-}ext}}_{\Pi}$ a forking term and hence quadratic in the advantage of the underlying dLog solver. Fischlin's transform instead keeps the reduction straight-line, at the bandwidth cost reported in Section 5.

Theorem 2. *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\Pi^{t,n}_{\mathsf{S\text{-}DKG}}$ be our protocol and $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ be the ideal DKG functionality as defined in Figure 3. Let $n, t \in \mathbb{N}$ with $t \leq n$ be the number of parties, respectively the number of parties necessary to reconstruct the key generated during the DKG execution. Suppose the CDH assumption holds, then*

$$\Pi^{t,n}_{\mathsf{S\text{-}DKG}} \leq \mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$$

in the $(\mathcal{F}_{bc}, \mathcal{F}_{\mathrm{ro}}, \mathcal{F}_{\mathrm{nizk}}, \mathcal{F}_{pki})$-hybrid world, assuming secure erasure, against an adversary that adaptively corrupts up to $t-1$ parties, which is a dishonest majority whenever $t-1 > n/2$.

We demonstrate the UC security of the $\mathsf{Janus_{1R}}$ protocol by constructing a simulator $\mathcal{S}$ and showing through a sequence of hybrid games that no polynomial-time environment $\mathcal{E}$ can distinguish between the real execution and the ideal world.

The simulator $\mathcal{S}$. To handle the "rushing" nature of the adversary in a single-round protocol, the simulator $\mathcal{S}$ leverages the shift capabilities of $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$. Its strategy is defined as follows:

- **Setup:** $\mathcal{S}$ retrieves the target global public key and verification keys $(\mathsf{pk}', \{\mathsf{vk}'_i\}_{i\in\mathcal{P}})$ from $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$.
- **First Honest Party Init:** $\mathcal{S}$ embeds these target values into the broadcast of the first honest party $\mathcal{P}_{i^*}$. It sets the public key contribution to $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk}'$ and the VSS commitments to $c_{i^*,j} \leftarrow \mathsf{vk}'_j$. It simulates the required NIZK $\pi^*_{i^*}$. Honest-to-honest ciphertexts are substituted with uniformly random values for later equivocation.

Target security is UC realization of functionality $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ with any $t \leq n$.

We assume erasure, authenticated and reliable broadcast, and the batched hashed ElGamal encryption $(\mathsf{Enc}, \mathsf{Dec})$ of Section 3. $\mathsf{NIZK.PoK}$ denotes straight-line extractable NIZK, while NIZK is a non-extractable NIZK. Proofs for L^{VSS}, L^{comeq}, L^{dec} and L^{enc} are simulated inside $\mathcal{F}_{\mathrm{nizk}}$, while proofs for L^{pk} are produced by the zero-knowledge simulator $\mathcal{S}_{\mathsf{nizk}}$ of the concrete proof system (Definition 2), which programs $\mathcal{F}_{\mathrm{ro}}$.

CRS: Retrieve group parameters $(\mathbb{G}, p, g)$ and external Pedersen parameter $h \in \mathbb{G}$ from $\mathcal{F}_{\mathsf{DKG}}^{t,n}$. Sample $\tau \leftarrow_{\$} \mathbb{Z}_p$ and define internal Pedersen parameter $v \leftarrow g^{\tau}$.

PKI model: The functionality is tagged by participant set $\mathcal{P}$, which translates into encryption public key pk_i^E for each $\mathcal{P}_i \in \mathcal{P}$ ($\mathcal{P}_i$'s private key is denoted sk_i^E).

Honest Party Init. Upon receiving (init, i) for an honest party $\mathcal{P}_i \notin$ Corr:

1. Pick two random $(t-1)$-degree polynomials $f_i(x) = \sum_{k=0}^{t-1} a_{i,k} x^k$ and $f_i'(x) = \sum_{k=0}^{t-1} b_{i,k} x^k$, let $a_i = (a_{i,k})_{k=0}^{t-1}$ and $b_i = (b_{i,k})_{k=0}^{t-1}$.
2. Create VSS information, $\mathsf{PedVSS}_i \leftarrow \{c_{i,k} = g^{a_{i,k}} \cdot v^{b_{i,k}}\}_{k=0}^{t-1}$.
3. Pick $\omega_i, r_i \leftarrow_{\$} \mathbb{Z}_p$ and create commitment $D_i = g^{\omega_i} \cdot v^{r_i}$.
4. Set $\pi_i^{\mathsf{VSS}} \leftarrow \mathsf{NIZK.PoK}[L_{g,v}^{\mathsf{VSS}}]((a_i, b_i), \omega_i, r_i), (\mathsf{PedVSS}_i, D_i))$
5. Set $\mathsf{pk}_i \leftarrow g^{f_i(0)}$ and $\pi_i^{\mathsf{pk}} \leftarrow \mathsf{NIZK}[L_{g,v}^{\mathsf{pk}}]((a_{i,0}, b_{i,0}), (\mathsf{pk}_i, c_{i,0}))$, exactly as the protocol prescribes.
6. For all $j \in$ Corr set $\phi_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j^E, (f_i(j), f_i'(j))$. For all $j \in (\mathcal{P} \setminus \text{Corr}) \setminus \{i\}$ simulate $\phi_{i,j}$ such that it can be equivocated later.
7. Broadcast $(\mathsf{PedVSS}_i, D_i, \pi_i^{\mathsf{VSS}}, \{\phi_{i,j}\}_{j \neq i})$.

Finish Init. Upon receiving the last round 1 broadcast from a corrupt party, or after sending the last round 1 broadcast as an honest party (whichever comes last), proceed with the following steps:

1. If, for some $i \in$ Corr, π_i^{VSS} or the proof of knowledge attached to $\{\phi_{i,k}\}_{k \neq i}$ doesn't verify, then send (abort, i) to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ for every such i and stop.
2. For all $i \in$ Corr whose Round 1 broadcast was sent while $\mathcal{P}_i$ was corrupted, extract $(a_i, b_i, \omega_i, r_i)$ from π_i^{VSS}, define polynomials $f_i(x) = \sum_{k=0}^{t-1} a_{i,k} x^k$ and $f_i'(x) = \sum_{k=0}^{t-1} b_{i,k} x^k$, and define $\mathsf{pk}_i \leftarrow g^{f_i(0)}$. For all other $i \in$ Corr, use the values generated in Honest Party Init.
3. For all $i \in$ Corr, $s_i \leftarrow \sum_{j \in \mathcal{P}} f_j(i)$.
4. Send $(\mathsf{initcomplete}, \{s_i, \omega_i\}_{i \in \text{Corr}})$ to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ and receive $(\mathsf{pk}, \mathsf{vk}_1, ..., \mathsf{vk}_n)$ in response.
5. For the least-indexed uncorrupted party $i^* \in \mathcal{P} \setminus$ Corr, overwrite $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk} / \prod_{j \in \mathcal{P} \setminus \{i^*\}} \mathsf{pk}_j$ and replace $\pi_{i^*}^{\mathsf{pk}}$ by $\mathcal{S}'_{\mathsf{nizk},2}((\mathsf{pk}_{i^*}, c_{i^*,0}))$. The contributions and proofs of all other honest parties stay as generated in Honest Party Init.
6. For all $i \in \mathcal{P} \setminus$ Corr:
   (a) For all $j \in$ Corr, trapdoor-decrypt $(s_{j,i}, s'_{j,i}) \leftarrow \phi_{j,i}$.
   (b) If, for some $j \in$ Corr, $(s_{j,i}, s'_{j,i}) \neq (f_j(i), f'_j(i))$ then for every such j broadcast $(j, s_{j,i}, s'_{j,i}, \pi_{i,j}^{\mathsf{dec}})$ where $\pi_{i,j}^{\mathsf{dec}}$ is simulated.
   (c) Otherwise, broadcast $(\mathsf{pk}_i, \pi_i^{\mathsf{pk}}, \mathsf{vk}_i, \pi_i^{\mathsf{comeq}})$ where π_i^{comeq} is simulated.

Output Generation. If there is no abort in round 2, upon receiving the last round 2 broadcast from a corrupt party (if any), proceed with the following steps:

1. If some party $\mathcal{P}_{i'}$, honest or corrupted, broadcast $(j, s_{j,i'}, s'_{j,i'}, \pi_{i',j}^{\mathsf{dec}})$ such that $\pi_{i',j}^{\mathsf{dec}}$ verifies and $g^{s_{j,i'}} \cdot v^{s'_{j,i'}} \neq \prod_{k=0}^{t-1} (c_{j,k})^{((i')^k)}$, then send (abort, j) to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ for every such j.
2. Otherwise, if, for some $i \in$ Corr, π_i^{pk} or π_i^{comeq} doesn't verify then send (abort, i) to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ for every such i.
3. Otherwise, send `success` to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$.

Adaptive Corruption. On corruption of party i, send $(\mathsf{corrupt}, i)$ to $\mathcal{F}_{\mathsf{DKG}}^{t,n}$, and program $\mathcal{F}_{\mathrm{ro}}$ such that every ciphertext $\phi_{j,i}$ that was simulated in Honest Party Init of a party $\mathcal{P}_j$ decrypts to $(f_j(i), f'_j(i))$. If init is not yet finished, then send $(f_i(i), f_i'(i), \mathsf{pk}_i, \pi_i^{\mathsf{pk}}, \omega_i, r_i)$ to $\mathcal{A}^*$, where π_i^{pk} is the proof already generated in Honest Party Init. $\mathcal{P}_i$ is then excluded from the choice of i^* below. If init is finished, then receive (s_i^*, ω_i^*) from $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ and proceed with the following steps:

1. $\Delta \leftarrow s_i^* - \sum_{j \in \mathcal{P}} f_j(i)$.
2. Send $(f_i(i) + \Delta, f_i'(i) - \Delta/\tau, \mathsf{pk}_i, \pi_i^{\mathsf{pk}}, \omega_i^*, r_i - (\omega_i^* - \omega_i)/\tau)$ to $\mathcal{A}^*$.

Fig. 6: Simulator for the Two-Round Distributed Key Generation Protocol.

Target security is UC realization of functionality $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$with any $t \leq n$.

We assume erasure, authenticated and reliable broadcast, and the batched hashed ElGamal encryption $(\mathsf{Enc}, \mathsf{Dec})$ of Section 3. $\mathsf{NIZK.PoK}$ denotes straight-line extractable NIZK, while NIZK is a non-extractable NIZK.

CRS: Group parameters $(\mathbb{G}, p, g)$, Pedersen parameter $h \in \mathbb{G}$.

PKI model: the functionality is tagged by participant set $\mathcal{P}$, which translates into encryption public key pk_i^E for each $\mathcal{P}_i \in \mathcal{P}$ ($\mathcal{P}_i$'s private key is denoted sk_i^E).

Setup. Upon the first invocation, request $(\mathsf{pk}', \{\mathsf{vk}_i'\}_{i\in\mathcal{P}})$ from $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$, and store those information.

First Honest Party Init. Upon receiving $(\mathsf{init}, \mathcal{P}_i)$ for the first honest party to broadcast, denoted $\mathcal{P}_{i^*}$:

1. Set $\mathsf{ComVSS}_i \leftarrow \{c_{i,j}\}_{j\in\mathcal{P}}$ with $c_{i,j} \leftarrow \mathsf{vk}_j'$ for all $j \in \mathcal{P}$.
2. Set $\mathsf{pk}_i \leftarrow \mathsf{pk}'$.
3. Simulate π_i^*.
4. For all $j \neq i \in$ Corr set $\phi_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j^E, (f'(j), \omega_j'))$. For all $j \neq i \in \mathcal{P} \setminus$ Corr simulate $\phi_{i,j}$ such that it can be equivocated later.
5. Broadcast $(\mathsf{pk}_i, \mathsf{ComVSS}_i, \pi_i^*, \{\phi_{i,j}\}_{j\neq i})$.

Remaining Honest Party Init. Upon receiving $(\mathsf{init}, \mathcal{P}_i)$ for the remaining honest party:

1. Pick $(t-1)$-degree polynomial $f_i(x) = \sum_{k=0}^{t-1} a_{i,k}x^k$.
2. Sample $\{\omega_{i,j}\}_{j\in\mathcal{P}}$ uniformly at random from $\mathbb{Z}_p^*$.
3. Create commitments, $\mathsf{ComVSS}_i \leftarrow \{c_{i,j}\}_{j\in\mathcal{P}}$ for $c_{i,j} \leftarrow g^{f_i(j)} \cdot h^{\omega_{i,j}}$.
4. Set $\mathsf{pk}_i \leftarrow g^{f_i(0)}$.
5. Set $\pi_i^* \leftarrow \mathsf{NIZK.PoK}_g((\{a_{i,k}\}_{k=0}^{t-1}, \{\omega_{i,j}\}_{j\in\mathcal{P}}), \mathsf{pk}_i, \mathsf{ComVSS}_i)$.
6. For all $j \neq i$ set $\phi_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j^E, (f_i(j), \omega_{i,j})$.
7. Broadcast $(\mathsf{pk}_i, \mathsf{ComVSS}_i, \pi_i^*, \{\phi_{i,j}\}_{j\neq i})$.

Finish Init. Upon receiving the last broadcast from a corrupt party, or after sending the last broadcast as an honest party (whichever comes last), proceed with the following steps:

1. If, for some $i \in$ Corr, π_i^* or the proof of knowledge attached to $\{\phi_{i,k}\}_{k\neq i}$ doesn't verify, then send (abort, i) to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$ for every such i and stop.
2. Extract $(\{a_{i,k}\}_{k=0}^{t-1}, \{\omega_{i,j}\}_{j\in\mathcal{P}})$ from π_i^* for all $i \in$ Corr whose broadcast was sent while $\mathcal{P}_i$ was corrupted. For every other party except the first honest party $\mathcal{P}_{i^*}$, use the values generated in Remaining Honest Party Init.
3. Construct Δ as the sum of all polynomials except for the one sent by the first honest party.
4. Construct $\{\Omega_i\}_{i\in\mathcal{P}}$ as the sum of $\omega_{j,i}$ for all $j \in \mathcal{P}$ except for the first honest party.
5. Send $(\mathsf{initcomplete}, \Delta, \{\Omega_i\}_{i\in\mathcal{P}})$ to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$.

Output Generation. After Finish Init, for every party $\mathcal{P}_i$ that is still honest:

1. Set $(s_{j,i}, s'_{j,i}) \leftarrow \mathsf{Dec}(\mathsf{sk}_i^E, \phi_{j,i})$ for every $j \in$ Corr whose broadcast was sent while $\mathcal{P}_j$ was corrupted.
2. If $g^{s_{j,i}} \cdot h^{s'_{j,i}} \neq c_{j,i}$ for some such j, then for every such j broadcast $(j, s_{j,i}, s'_{j,i}, \pi_{i,j}^{\mathsf{dec}})$, where $\pi_{i,j}^{\mathsf{dec}}$ is simulated, and send (abort, i, j) to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$.
3. Otherwise, send $(\mathsf{success}, i)$ to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$.

Report Handling. If a corrupted party $\mathcal{P}_{i'}$ broadcasts $(j, s_{j,i'}, s'_{j,i'}, \pi_{i',j}^{\mathsf{dec}})$ s.t. $\pi_{i',j}^{\mathsf{dec}}$ verifies and $g^{s_{j,i'}} \cdot h^{s'_{j,i'}} \neq c_{j,i'}$, then send (abort, i', j) to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$.

Adaptive Corruption. On corruption of party i, send $(\mathsf{corrupt}, i)$ to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$ and receive $(\top, f'(i), \omega_i')$, as well as (s_i^*, ω_i^*) if setup has finished, from $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$, and proceed with the following steps:

1. If $i \neq i^*$, program $\mathcal{F}_{\mathrm{ro}}$ such that the ciphertext $\phi_{i^*,i}$ sent by the first honest party $\mathcal{P}_{i^*}$ decrypts to $(f'(i), \omega_i')$.
2. Send to $\mathcal{A}^*$ the retained share $(s_{i,i}, s'_{i,i})$, which is $(f_i(i), \omega_{i,i})$ from Remaining Honest Party Init if $i \neq i^*$ and $(f'(i^*), \omega_{i^*}')$ if $i = i^*$, together with (s_i^*, ω_i^*) if $\mathcal{P}_i$ has already output.

Fig. 7: Simulator for the Single-Round Distributed Key Generation Protocol.

– **Remaining Honest Party Init:** $\mathcal{S}$ follows the protocol honestly for all other uncorrupted parties, sampling random polynomials and commitments, computing their NIZKs, and encrypting shares for other parties.
– **Finish Init:** Once all broadcasts are complete, $\mathcal{S}$ extracts the adversarial sharing polynomial f_i and blinding values $\{\omega_{i,j}\}_{j\in\mathcal{P}}$ from the adversarial $\mathsf{NIZK.PoK}$s. $\mathcal{S}$ then aggregates the polynomials and blinding values of all parties, except for the first honest party $\mathcal{P}_{i^*}$, to compute the shift polynomial Δ and the blinding shifts $\{\Omega_i\}_{i\in\mathcal{P}}$. It submits these shifts to $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ to finalize the setup.
– **Adaptive Corruption:** Upon corruption of an honest party, $\mathcal{S}$ retrieves the key share and the blinding value from $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$. $\mathcal{S}$ programs $\mathcal{F}_{\text{ro}}$ such that the initially broadcast dummy ciphertexts decrypt exactly to the original, unshifted key share and blinding value.

Sequence of games. We structure the indistinguishability proof using a sequence of hybrid games, starting from the real-world execution and ending in the ideal-world simulation.

Game 0 (Real World) This is the real-world execution of the protocol $\Pi^{t,n}_{\mathsf{S\text{-}DKG}}$ with the environment $\mathcal{E}$. All honest parties hold real inputs and compute their messages exactly as prescribed by the protocol.

Game 1 (Ciphertext Consistency) In the next transition, we rule out inconsistencies with honest-to-honest ciphertexts. The simulator aborts if $\mathcal{E}$ queries $\mathcal{F}_{\text{ro}}$ with a valid decryption value $u = (\mathsf{pk}_i^E)^r$ for a ciphertext sent by a party that was honest at the time of encryption to a party that is still honest. We show by reduction that if the abort happens, we can solve the CDH problem. $\mathcal{S}$ guesses which honest party $\mathcal{P}^*$ is the receiver of the ciphertext for which the abort happens, which is correct with probability at least $\frac{1}{\mathsf{n}}$. We embed a CDH challenge $(A = g^a, B = g^b)$ such that the public key of the guessed honest party is set to A and the ephemeral randomness in the ciphertext is set to B. Specifically, all remaining honest parties sample a random exponent ψ_i and use B^{ψ_i} as their ephemeral randomness.

The ciphertext sent from an honest party $\mathcal{P}_i$, except $\mathcal{P}^*$, to a malicious party $\mathcal{P}_j$ is computed as $\phi_{i,j} \leftarrow (B^{\psi_i}, \mathsf{H}(B^{\psi_i \cdot \mathsf{sk}_j^E}) \oplus msg)$ with sk_j^E taken from $\mathcal{F}_{\text{pki}}$ as defined in Section D.5. Decrypting ciphertexts sent by a malicious party $\mathcal{P}_j$ to $\mathcal{P}^*$ is done by extracting $\mathcal{P}_j$'s ephemeral randomness r from the proof that is part of the ciphertext, read off the internal table of $\mathcal{F}_{\text{nizk}}$ for L^{enc}, and computing the input to $\mathcal{F}_{\text{ro}}$ as $(\mathsf{pk}_{\mathcal{P}^*}^E)^r$. The same technique is used when generating a complaint against $\mathcal{P}_j$ if it sent an incorrect ciphertext. The proofs for L^{enc} of honest senders, whose randomness B^{ψ_i} has an unknown discrete logarithm, are added to that table without a witness, which is possible because $\mathcal{F}_{\text{nizk}}$ is simulated and the statements are true.

Write $\mathcal{P}_{i^*}$ for the first honest party to broadcast, whose broadcast the ideal execution will later replace by one that embeds pk' and $\{\mathsf{vk}'_j\}_{j\in\mathcal{P}}$ from $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$. Each ciphertext component c_j sent by $\mathcal{P}_{i^*}$ to another honest party is replaced by a uniformly random string of the same length, honest recipients obtain the corresponding plaintext m_j directly instead of querying $\mathcal{F}_{\text{ro}}$, and $\mathcal{F}_{\text{ro}}$ is programmed at $(\mathsf{pk}_j^E)^r$ to $c_j \oplus m_j$ when the recipient $\mathcal{P}_j$ is corrupted. Unless the abort above occurs, $\mathcal{F}_{\text{ro}}$ has not been queried at $(\mathsf{pk}_j^E)^r$ before this programming, so in the previous game $c_j = \mathcal{F}_{\text{ro}}((\mathsf{pk}_j^E)^r) \oplus m_j$ is already uniformly random and independent of the remaining view, and the replacement does not change the distribution.

Extracting the query that includes the solution to the CDH challenge requires $\mathcal{S}$ to guess a query to $\mathcal{F}_{\text{ro}}$, which succeeds with probability at least $\frac{1}{q_{\text{ro}}}$, where q_{ro} is the number of queries to $\mathcal{F}_{\text{ro}}$. The environment's advantage in distinguishing Game 1 from Game 0 is bounded by the advantage of solving CDH, which is negligible.

Game 2 (Pedersen Binding) Game 2 is identical to Game 1, except that it aborts if an honest party $\mathcal{P}_i$ accepts a share $(s_{j,i}, s'_{j,i})$ with $g^{s_{j,i}} h^{s'_{j,i}} = c_{j,i}$ from a party $\mathcal{P}_j$ that was corrupted when it broadcast, but $(s_{j,i}, s'_{j,i}) \neq (f_j(i), \omega_{j,i})$ for the values extracted from π_j^*. This abort is needed because the ideal functionality derives $\mathcal{P}_i$'s share from the extracted polynomials, while $\mathcal{P}_i$ in the real execution sums the decrypted shares. We show that an execution in which Game 2 aborts yields a reduction that computes $\log_g(h)$. The reduction plays Game 1 with h set to its dLog challenge and deviates from it nowhere else, which is possible because no step of Game 1 uses $\log_g(h)$. When the abort occurs, $c_{j,i}$ has the two distinct openings $(f_j(i), \omega_{j,i})$ and

$(s_{j,i}, s'_{j,i})$, which forces $s'_{j,i} \neq \omega_{j,i}$ and reveals $\log_g(h) = (f_j(i) - s_{j,i})/(s'_{j,i} - \omega_{j,i})$. Hence, the environment's advantage in distinguishing Game 2 from Game 1 is at most $\mathsf{Adv}^{\mathsf{dLog}}_{\mathcal{A}}$.

Game 3 (Ideal Execution & Key Shifting) In this final game, the simulator $\mathcal{S}$ interacts fully with the ideal functionality $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$, replacing the real-world execution. $\mathcal{S}$ injects the values retrieved from $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$, extracts adversarial shares from the NIZK.PoKs, and equivocates honest party states. Because the environment expects the final global public key to be the aggregate of all individual contributions, $\mathcal{S}$ computes the sum of all other contributions (both adversarial and the remaining honest ones generated by $\mathcal{S}$) to form the shifts Δ and $\{\Omega_i\}$. By submitting these shifts to $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$, the ideal functionality outputs a final key pk and verification keys vk that perfectly match the algebraic structure of the aggregated broadcast messages. Since the real protocol sums uniformly random polynomials, and this game shifts a uniformly random functionality-provided polynomial by a known offset, the resulting distributions of the public keys and commitments are perfectly identical.

A core component of this hop is ensuring that the simulator can provide a consistent internal state for any honest party corrupted mid-execution. Because honest parties erase all ephemeral randomness and state (retaining only their final shares $(s_{i,i}, s'_{i,i})$) after the first round, $\mathcal{S}$'s task simplifies to dynamically equivocating the received ciphertexts. When a corruption occurs, $\mathcal{S}$ retrieves the targeted secret values from $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ and adjusts for the previously submitted shifts to deduce the exact expected local state. To complete this state equivocation, $\mathcal{S}$ retroactively programs the random oracle $\mathcal{F}_{\text{ro}}$ evaluating the ElGamal hashes. Because the dummy ciphertexts were originally generated with fresh randomness, $\mathcal{S}$ can seamlessly program the $\mathcal{F}_{\text{ro}}$ outputs so that the dummy ciphertexts, specifically those sent by the first honest party, decrypt exactly to shares consistent with the secrets provided by the functionality.

To conclude, the programming of the random oracle does not introduce any inconsistencies, as Game 1 already established that the probability that $\mathcal{S}$ cannot program $\mathcal{F}_{\text{ro}}$, because $\mathcal{F}_{\text{ro}}$ has already been queried before on that value, is bound by the probability of solving the CDH problem. Furthermore, replacing real proofs with simulated ones in $\mathcal{F}_{\text{nizk}}$ is perfectly indistinguishable due to $\mathcal{S}$'s ability to simulate $\mathcal{F}_{\text{nizk}}$ internally. Because the state equivocation provides a perfectly consistent internal view to the adversary, Game 3 is distributed identically to Game 2 as long as neither the abort of Game 1 nor that of Game 2 occurs, so this hop adds no term to the bound, and Game 3 represents the final ideal-world execution.

The overall distinguishing advantage of the environment $\mathcal{E}$ between the real-world execution of $\Pi^{t,n}_{\mathsf{S\text{-}DKG}}$ and the ideal-world interaction with $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ and $\mathcal{S}$ is bounded by the probability of breaking dLog or CDH. Specifically:

$$\mathsf{Adv}^{\text{dist}}_{\mathcal{E}} \leq \mathsf{Adv}^{\mathsf{dLog}}_{\mathcal{A}} + \mathsf{n} \cdot q_{\text{ro}} \cdot \mathsf{Adv}^{\mathsf{CDH}}_{\mathcal{A}} + \text{negl}(\lambda)$$

Under the CDH assumption, these advantages are negligible. Therefore, $\Pi^{t,n}_{\mathsf{S\text{-}DKG}}$ securely realizes $\mathcal{F}^{t,n}_{\mathsf{S\text{-}DKG}}$ in the UC framework.

5 Prototype and Benchmark

We support our work with an implementation and performance evaluation. Our prototype is available at [39].Beyond demonstrating that both Janus protocols are practical even for large deployments, we structure our performance evaluation around four questions:

Variant trade-offs: How do $\mathsf{Janus}_{\mathsf{1R}}$ and $\mathsf{Janus}_{\mathsf{2R}}$ scale with committee size?
System bottlenecks: What are the primary bottlenecks in large-scale executions ($n \geq 256$)?
Identifiable abort: What overhead does identifiable abort introduce on the happy path and during complaint verification?
Straight-line extractability cost: What computational and bandwidth savings are achieved by replacing Fischlin's transform either by working in the AGM, settling for simulation-based security, or in the honest majority setting?

Experimental setup We ran all benchmarks on a dedicated AWS `c8i.4xlarge` instance equipped with an Intel Xeon 6975P CPU (8 physical cores, 16 hardware threads) and 30 GB of RAM. We implement Janus in Rust (v1.97.1), leveraging `curve25519-dalek` for elliptic-curve operations, `sha2` for SHA-512 hashing, `merlin` for transcripts, and `zeroize` for secure erasures. Adaptive public-key encryption is instantiated using hashed ElGamal. Throughout this section, we measure local computation time, communication overhead, and latency for the settings $t = n \in \{16, 32, 64, 128, 256, 512\}$ for both $\mathsf{Janus}_{\mathsf{1R}}$ and $\mathsf{Janus}_{\mathsf{2R}}$.

We instantiate all zero-knowledge proofs as Chaum-Pedersen proofs [18]. For non-extractable proofs, we use the Fiat-Shamir transform [36] to make the proofs non-interactive. For extractable proofs, we benchmark multiple instantiations. First, we compile them using the Fischlin transform [37], which provides straight-line extractability. Following the optimizations of [24], we evaluate two parameter sets targeting 128-bit security:

- *Fischlin-small* ($\rho = 16$ repetitions, $b = 8$ challenge bits), which keeps proof size smaller at the expense of higher prover overhead ($\approx 16\times$ larger than Fiat–Shamir); and
- *Fischlin-large* ($\rho = 43$ repetitions, $b = 3$ challenge bits), which is designed to reduce prover computation at the cost of larger proof size ($\approx 43\times$ larger than Fiat–Shamir).

Finally, to quantify the overhead of the Fischlin transform, we evaluate the proofs of knowledge with the Fiat-Shamir transform [36] (referred to as *Fiat-Shamir*).

Variant trade-offs We benchmark the local computation times of both $\mathsf{Janus}_{\mathsf{1R}}$ and $\mathsf{Janus}_{\mathsf{2R}}$, deferring detailed measurements to Section C (Tables 4 and 5). As expected, verifying the other parties' PoKs dominates the runtime. Under *Fischlin-small* proofs, $\mathsf{Janus}_{\mathsf{1R}}$ spends 792 ms in its Output phase (which includes PoK verification) at $(t, n) = (64, 64)$ and 62.1 s at $(512, 512)$ (Table 4). By comparison, $\mathsf{Janus}_{\mathsf{2R}}$ spends 262 ms and 14.3 s in its Finalize phase for the respective committee configurations (Table 5). We also measured that the Fischlin-large configuration is less favorable than Fischlin-small, as it increases the verification time and proof size, which scale linearly with n. For this reason, we use Fischlin-small as the default for the online-extractable setting.

Regardless of the proof system used, the performance gap between $\mathsf{Janus}_{\mathsf{1R}}$ and $\mathsf{Janus}_{\mathsf{2R}}$ stems from a structural difference: $\mathsf{Janus}_{\mathsf{1R}}$ proves one evaluation per recipient, so the first message of its PoK carries $n+1$ group elements, per repetition under Fischlin, whereas $\mathsf{Janus}_{\mathsf{2R}}$ proves knowledge of the $t + 1$ committed coefficients and its first message carries exactly one group element. This asymmetry drives the proof size, 48.0 KB for $\mathsf{Janus}_{\mathsf{1R}}$ against 32.1 KB for $\mathsf{Janus}_{\mathsf{2R}}$ at $(512, 512)$ under Fiat-Shamir, the proving cost, and the verification cost. $\mathsf{Janus}_{\mathsf{2R}}$'s cost per proof is essentially independent of the committee size, so verifying all n proofs is quadratic in n; $\mathsf{Janus}_{\mathsf{1R}}$'s cost per proof itself grows with the committee, since the verifier evaluates the folded response polynomial of degree t at n points, so the total grows faster than quadratically, about $n^{2.4}$ under Fiat-Shamir.

To isolate the effects of different thresholds on both schemes, we evaluate a parameter sweep fixing the committee size at $n = 256$ while varying the threshold t independently (Figure 8). This sweep also quantifies the overhead of the Fischlin transform across both variants: it incurs a 4.3–7.3× slowdown for $\mathsf{Janus}_{\mathsf{1R}}$, but only 1.4–1.6× for $\mathsf{Janus}_{\mathsf{2R}}$. This difference occurs because PoK verification dominates the Output phase of $\mathsf{Janus}_{\mathsf{1R}}$, whereas $\mathsf{Janus}_{\mathsf{2R}}$ spends the majority of its Finalize phase evaluating the committed polynomial in the exponent, which is unaffected by the choice of NIZK transform (cf. Figure 10). Since both $\mathsf{Janus}_{\mathsf{1R}}$ and $\mathsf{Janus}_{\mathsf{2R}}$ have the highest computation time in the $t = n$ setting, we continue our benchmarks with this setting, which gives us the most conservative estimates. We expect $\mathsf{Janus}_{\mathsf{1R}}$ only to be slightly more efficient for smaller t, and $\mathsf{Janus}_{\mathsf{2R}}$ considerably more efficient.

In addition to local computation, we also measure the latency of our protocols in three settings: within one region (1 ms RTT, 10 Gbit/s bandwidth), across regions (25 ms, 1 Gbit/s), and intercontinental (150 ms, 1 Gbit/s). We model end-to-end latency across r network rounds as $\max(\mathsf{compute}, \mathsf{transfer}) + r \cdot \mathrm{RTT}$, where compute is the measured single-core computation time and transfer is the network transmission delay. We use the maximum of both rather than the sum since each party can verify all messages when they arrive instead of waiting for the complete round. The compute we report is the single-core path, so our latencies are the conservative reading. Attributing the network from measured byte counts and a stated link follows the

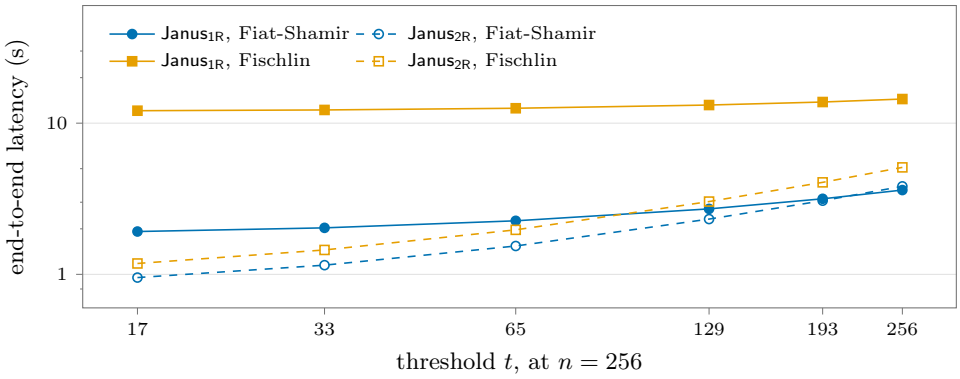

Fig. 8: End-to-end latency against the threshold t at a fixed committee $n = 256$, on an intercontinental link (150 ms RTT, 1 Gbit/s), single-core.

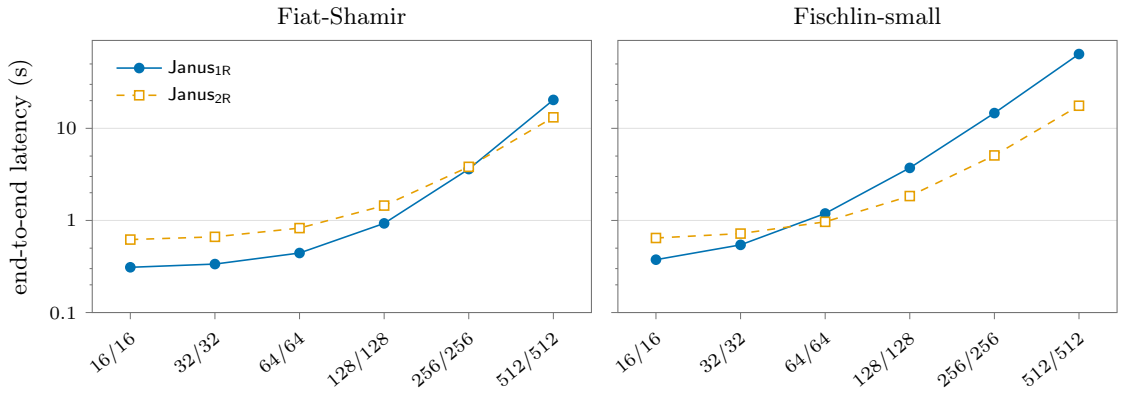

Fig. 9: End-to-end latency per party at $t = n$, intercontinental link (150 ms RTT, 1 Gbit/s), single-core, one vertical scale.

practice of [56]. Additionally, we charge each broadcast round two network rounds, which is what a broadcast costs both in the optimistic case and with a trusted dealer (cf. Section 7), so $\mathsf{Janus}_{\mathsf{1R}}$ has two network rounds and $\mathsf{Janus}_{\mathsf{2R}}$ four. We depict the results in Figure 9.

As expected, $\mathsf{Janus}_{\mathsf{1R}}$ is favorable for smaller committee sizes where eliminating a broadcast outweighs its higher computational overhead, whereas $\mathsf{Janus}_{\mathsf{2R}}$ dominates in larger committees where local computation surpasses network delay. Under an intercontinental link, $\mathsf{Janus}_{\mathsf{1R}}$ achieves an end-to-end latency of 310 ms compared to 620 ms for $\mathsf{Janus}_{\mathsf{2R}}$ at $(t, n) = (16, 16)$, where the second broadcast of $\mathsf{Janus}_{\mathsf{2R}}$ accounts for the entire latency difference. Conversely, at $(512, 512)$, $\mathsf{Janus}_{\mathsf{2R}}$ completes in 13.2 s versus 20.4 s for $\mathsf{Janus}_{\mathsf{1R}}$, as network delay becomes negligible.

The crossover point occurs at $(512, 512)$ under Fiat-Shamir proofs and already at $(64, 64)$ under *Fischlin-small* proofs (Figure 9). This divergence occurs because $\mathsf{Janus}_{\mathsf{2R}}$ trades an additional broadcast for reduced local computation, and hence, the crossover occurs precisely where the computational saving exceeds the 300 ms RTT penalty of that broadcast. Under Fiat-Shamir proofs, $\mathsf{Janus}_{\mathsf{2R}}$'s computational saving is only 90 ms at $(256, 256)$ and surpasses the penalty at $(512, 512)$ (7.5 s saved). Under Fischlin proofs, the proof-size expansion shifts this saving to 524 ms already at $(64, 64)$. In single-region deployments, computation dominates total latency across all configurations: $\mathsf{Janus}_{\mathsf{1R}}$ leads only up to $(128, 128)$ under Fiat-Shamir proofs and is strictly outperformed by $\mathsf{Janus}_{\mathsf{2R}}$ across all settings under Fischlin proofs.

System bottlenecks To better understand system bottlenecks for larger committee sizes, we analyze $\mathsf{Janus}_{\mathsf{1R}}$ in the $(512, 512)$ setting under *Fischlin-small* proofs, as this is the configuration in which a party receives the most data. We first observe that each party receives 408 MB of data and needs 62.1 s to verify it. On a 1 Gbit/s link, transfer costs 3.42 s, and computation is the limiting factor by a factor of 18×. To better understand how computation is split amongst our protocols, we measure each component of both Janus versions at $(16, 16)$ and $(512, 512)$ and show them in Table 6 and Figure 10.

Of the 823 ms of output-phase work we account for, 512 ms verify the n well-formedness proofs and 296 ms decode the received bytes into group elements, since decompressing every compressed Ristretto point costs an inverse square root. All remaining checks together account for 15 ms, and both dominant items are quadratic in n, so this split persists at the larger settings.

As decoding messages is a dominant factor, we also implemented a shortened proof. The observation we used there is that the verifier can rebuild the first-round commitments from the challenge and the responses instead of receiving them, which shortens the proof by 33%. However, our measurements show that encoding, decoding and verifying together become 2.2 to 5.5 times slower, and the trade only pays below roughly 4 Mbit/s. We therefore decided to keep the longer encoding, but ship the alternative with our artifact.

Since verification and decoding can both be parallelized, we also measured the protocols with parallelization over all cores which we report pinned to a single core throughout. Distributing $\mathsf{Janus}_{\mathsf{1R}}$'s output phase over all cores reduces those 62.1 s to 5.58 s, a speedup of 11.1×, which leaves computation ahead of the transfer by less than a factor of two, so beyond this point further cores no longer buy latency and the link becomes the constraint. $\mathsf{Janus}_{\mathsf{2R}}$'s finalize phase behaves the same way and gains 6.0 to 10.8× (Tables 4 and 5), both above the eight physical cores because the machine runs sixteen threads. Only $\mathsf{Janus}_{\mathsf{2R}}$'s output phase stalls near 3×, since it checks two short proofs per dealer and then aggregates, but it stays near 100 ms even at the largest setting. During our measurements, memory was never the constraining factor, since holding every party of the $(512, 512)$ setting at once peaks at 1.7 GB.

Identifiable abort We also measured the costs of the identifiable abort protocol, and depict the numbers in Table 2, plotted in Figure 12. A complaint is a single message of 259 bytes in both variants, carrying the two party indices, the disputed opening, a group element, a proof of correct decryption, and a signature. Constructing it costs 0.09 ms and does not depend on the committee size. Verifying it costs 0.14 ms in $\mathsf{Janus}_{\mathsf{1R}}$, again independently of the committee size, since the verifier checks a single point commitment; in $\mathsf{Janus}_{\mathsf{2R}}$ the verifier evaluates the committed polynomial in the exponent, so the cost grows with the threshold, from 0.39 ms at $t = 16$ to 8.2 ms at $t = 512$. This gap again comes from the commitment structure: $\mathsf{Janus}_{\mathsf{1R}}$ broadcasts a commitment per recipient and can check a disputed share by looking it up, whereas $\mathsf{Janus}_{\mathsf{2R}}$ broadcasts only t coefficient commitments and has to rebuild the one in question. The worst case for IA is if the dealer misbehaves and all $n - 1$ other parties object. In this case the IA cost grows linearly in $\mathsf{Janus}_{\mathsf{1R}}$ and quadratically in $\mathsf{Janus}_{\mathsf{2R}}$: at $(512, 512)$ it costs 69 ms against a run of 20.4 s, and 4.17 s against a run of 13.2 s.

	$\mathsf{Janus}_{\mathsf{1R}}$		$\mathsf{Janus}_{\mathsf{2R}}$	
(t, n)	one	all	one	all
$(16, 16)$	0.14	2.0	0.39	5.8
$(32, 32)$	0.14	4.2	0.64	19.8
$(64, 64)$	0.14	8.5	1.1	72.0
$(128, 128)$	0.14	17.2	2.1	272.4
$(256, 256)$	0.14	34.6	4.1	1 060
$(512, 512)$	0.14	69.3	8.2	4 170

Table 2: Complaint verification time in ms, for one complaint and for all $n - 1$.

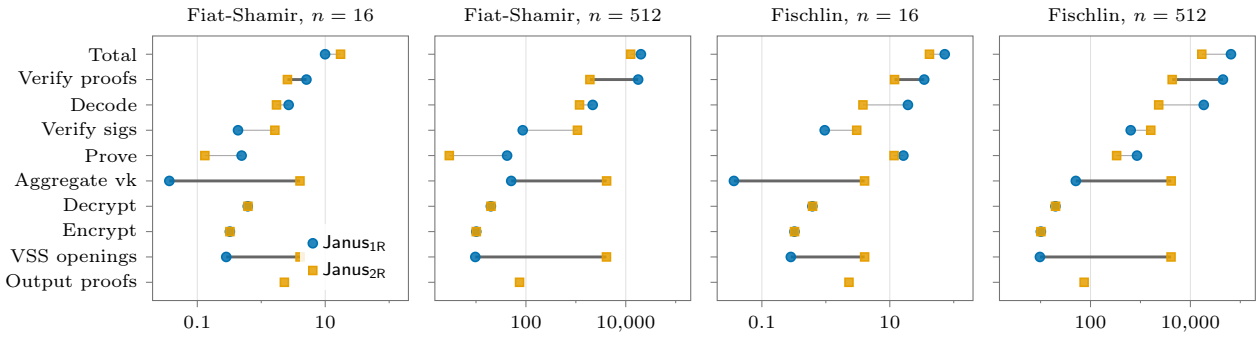

Fig. 10: Per-operation cost per party in ms at $t = n$. We mark the three rows that produce the cross-over of Figure 9 with a heavier connector. Each panel on its own logarithmic axis.

Straight-line extractability cost We finally analyze how much additional performance we can achieve either by accepting stronger assumptions or a weaker security notion. The intuition behind this is that if we either run our security proof in the AGM or reduce our model to simulation-based security rather than UC, we can replace the Fischlin transformation with Fiat-Shamir for our PoKs.[9]

We observe that this improves performance in three different ways. The proof itself shrinks by exactly ρ, 16.0× at $(128, 128)$ and again at $(512, 512)$, and within Fischlin the large parameters exceed the small ones by 2.69×, as the ratio 43/16 predicts. The broadcast message shrinks by only 8.4×, from 818 KB to 97 KB sent and from 408 MB to 48 MB received per party at $(512, 512)$, since commitments and ciphertexts are unaffected by the transform. Computation shrinks least, by 3.1× in the output phase of $\mathsf{Janus}_{\mathsf{1R}}$ at $(512, 512)$, since the ρ repetitions already collapse into a single batched verification equation.

6 Extensions

Many discrete-logarithm threshold primitives require auxiliary correlated randomness beyond a standard secret key sharing (cf. Table 3). In this section, we show how the core Janus architecture naturally extends to support four key capabilities: (i) establishing pairwise blinding seeds for masking-based adaptive security, (ii) generating shares of zero for share blinding, (iii) executing proactive key-share refreshing across dynamic epochs, and (iv) achieving semi-robustness against malicious actors or network faults.

Pairwise Blinding Seeds Several threshold protocols achieving adaptive security via masking using symmetric seeds $k_{i,j} = k_{j,i}$ with a public commitment $K_{i,j} = \mathsf{H}_k(k_{i,j})$ between every pair of parties, where the commitments $\{K_{i,j}\}_{j \neq i}$ are part of party $\mathcal{P}_i$'s verification key [10,45,9,3,23,8]. We equip Janus with this auxiliary key material at *zero communication overhead* via non-interactive Diffie-Hellman key agreement: for each pair $(\mathcal{P}_i, \mathcal{P}_j)$ with static encryption keys $(\mathsf{pk}_i^E, \mathsf{pk}_j^E)$, both parties locally compute

$$k_{i,j} \leftarrow \mathsf{H}_k((\mathsf{pk}_j^E)^{\mathsf{sk}_i^E}) = \mathsf{H}_k((\mathsf{pk}_i^E)^{\mathsf{sk}_j^E}) = k_{j,i}.$$

Because these values are derived purely locally from registered public keys, they introduce no additional communication rounds. This mechanism also integrates seamlessly with our simulation and identifiable abort strategies: (i) between honest parties, the underlying Diffie-Hellman secret is pseudorandom under CDH, (ii) for corrupted parties, the simulator directly computes $k_{i,j}$ from the extracted adversarial secret keys, and (iii) any dispute over pairwise seed consistency can reuse the ElGamal ciphertext complaint mechanism used in Janus.

[9] This also works if the parties have an honest majority. In this case we only need proofs of well-formedness and no proofs of knowledge.

		Pedersen Commitments	Pairwise Seeds	Zero Sharings
Threshold Signatures	DR24 [31]	•		•
	KRT24 [45]		•	
	BDLR25a [5]	•		•
	BDLR25b [4]	•		•
	C25 [23]		•	
	JN25 [44]	•		•
	BGLRSY25 [8]	•	•	
	GCRS26 [40]	•		•
	BCLTZ26 [3]		•	
Oblivious PRFs	BFHLY20 [10]		•	
	BGRS25 [9]		•	
	JN25 [44]	•		•

Table 3: Overview of DLog-based threshold cryptosystems compatible with Janus. "Pedersen commitments" indicates that the partial verification keys of Janus are required by the cryptosystem.

Zero-Sharings In addition to the sharing of the main secret key sk, several recent DLog-based threshold cryptosystems [31,44,5,4,40] utilize one or more auxiliary Shamir secret sharings of zero to achieve adaptive security.[10] In these schemes, each party's secret key is typically of the form $\mathsf{sk}_i = (x_i, y_i)$, where x_i is a share of sk and y_i is a share of 0. The global public key remains standard $\mathsf{pk} = g^{\mathsf{sk}}$, while the partial verification keys take the form of Pedersen commitments $\mathsf{vk}_i = g^{x_i} h^{y_i}$.

Both Janus variants naturally extend to provide zero-sharings without altering their round complexities or underlying cryptographic assumptions. For $\mathsf{Janus}_{\mathsf{2R}}$, the first round procedure is repeated in parallel for each needed zero sharing, with proofs that the new polynomials $f_i^0(x)$ evaluate to zero at $x = 0$. In the second round, each party forms their vk_i in the requisite form by combining the secret keys produced by the parallel runs. As already the case in $\mathsf{Janus}_{\mathsf{2R}}$, each party must prove that its published vk_i is consistent with the first-round messages. For $\mathsf{Janus}_{\mathsf{1R}}$, the independent Pedersen randomness $\omega_{i,j}$ is replaced with correlated randomness that is derived from a zero-sharing polynomial. Consistent with single-round lower bounds, this realizes a shifted functionality where the rushing adversary can inject an additive shift into the sharing while maintaining that the reconstruction of the additional sharing results in zero.

Proactive Key Refresh Key refresh [13] provides *proactive security* by replacing existing key shares with fresh, independent shares of the same secret without altering the public key. Both Janus variants support key refresh by rerunning the protocol with sharing polynomials whose constant term is fixed to zero and adding the resulting shares to the existing ones. At each refresh, every party also erases its decryption key sk_i^E and registers a fresh encryption key pair, so that a party corrupted in a later epoch cannot decrypt the ciphertexts it received in earlier epochs. Because $\sum_i f_i(0) = 0$, the master secret sk and public key $\mathsf{pk} = g^{\mathsf{sk}}$ remain unchanged, while all shares and blinding factors are re-randomized independently of their prior values. Periodic refreshes across epochs effectively reset the adversary's corruption budget, tolerating up to $t - 1$ compromised parties in each epoch.

We note that the standard proactive refresh technique described above restores secrecy against past state leakage, but does not provide *robust share recovery* for parties whose internal state is actively erased or modified while compromised. Achieving self-healing proactive security against active adaptive adversaries, who may maliciously destroy or overwrite stored key shares, requires dedicated share-recovery subprotocols or out-of-band backups. Formally capturing these state-restoration semantics would substantially complicate

[10] We present the case of a single zero-sharing, but the construction extends directly to multiple zero-sharings.

the ideal functionality while remaining orthogonal to the core contributions of our paper, and we therefore leave robust adaptive refresh with share recovery to future work.

Semi-Robustness In its base form, both Janus versions abort globally if any party's first-round proof fails. Semi-robustness offers a weaker but more practical guarantee: a party $\mathcal{P}_j$ whose first-round proof π_j^{VSS} does not verify is excluded from key generation, and the remaining parties proceed with the reduced set $\mathcal{P} \setminus \{\mathcal{P}_j\}$. Since first-round messages are broadcast and independently verifiable by all parties, exclusion can be decided without additional communication.

Restoring robustness after other failures (such as an invalid share or an inconsistent verification key in $\mathsf{Janus}_{\mathsf{2R}}$) cannot be handled as gracefully, since resolving them would require an additional round of interaction. The semi-robust variant therefore applies only to first-round exclusions. We treat this extension informally: it realizes a variant of $\mathcal{F}_{\mathsf{DKG}}^{t,n}$ in which the adversary may reduce the participant set before key generation begins, provided the remaining honest parties still exceed the threshold t, but we leave a formal functionality to future work.

7 Discussion

We discuss design choices of Janus, suitable instantiations of the broadcast channel, and applicability of $\mathsf{Janus}_{\mathsf{1R}}$ to FROST [8].

Point- vs. Coefficient-Commitments The choice between point- and coefficient-based Pedersen commitments involves a tradeoff between communication cost and round complexity. Coefficient-commitments require only t group elements per party per round (one per polynomial coefficient), while point-commitments require n (one per recipient evaluation). For $n \gg t$, coefficient-commitments are significantly more communication-efficient.

Point-commitments are essential to $\mathsf{Janus}_{\mathsf{1R}}$'s single-round design: since each $c_{i,j}$ directly encodes $\mathcal{P}_i$'s contribution to $\mathcal{P}_j$'s share, verification keys can be aggregated non-interactively without further communication. Using coefficient-commitments in $\mathsf{Janus}_{\mathsf{1R}}$ would break this property: parties would need an additional round to open their evaluations and publish their verification keys, essentially recovering the two-round structure of $\mathsf{Janus}_{\mathsf{2R}}$. Conversely, using point-commitments in $\mathsf{Janus}_{\mathsf{2R}}$ would allow dropping the explicit verification-key opening step in Round 2, at the cost of higher Round 1 communication. We discuss the concrete performance impact of this tradeoff in Section 5.

Realizing the Broadcast Channel. Our protocols are stated in the $\mathcal{F}_{\mathrm{bc}}$-hybrid world. Building a deterministic broadcast channel from point-to-point channels requires t rounds [33], and broadcast with identifiable abort needs $\Omega(n/(n-t))$ rounds [26]. Since these general results are impractical for large signer sets and high threshold t, yet a broadcast channel is a common requirement, deployments rely on one of the following alternatives.

The first option realizes $\mathcal{F}_{\mathrm{bc}}$ via an underlying broadcast layer that operates under its own corruption threshold, decoupled from the $t-1$ corruptions among the n participants tolerated by the DKG. Given signatures and a PKI, honest-majority Byzantine agreement provides such a broadcast layer in expected constant rounds [47]. For instance, NEAR's MPC network [53] follows this design, which fixes the fault tolerance of its reliable broadcast layer to $n/3$ as an invariant infrastructure parameter while allowing the cryptographic threshold t to be configured independently per scheme.

The second option routes communication through a coordinator (a relay server or initiator node), as in SSV's DKG initiator [60], where parties echo hashes of received messages in subsequent rounds to detect transcript divergence [20,21]. ChillDKG [57] tightens this into transcript certification, requiring participants to collect signatures on the session transcript from all n parties. Consequently, the coordinator is trusted solely for liveness: it can force an abort, and attributing silent parties requires it to report deliveries honestly, but the coordinator cannot compromise safety or forge messages.

The third option instantiates $\mathcal{F}_{\mathrm{bc}}$ via an underlying blockchain, where each broadcast round corresponds to one block confirmation. For example, Aptos employs this design in its DKG, where validators reach

consensus on a single aggregated PVSS transcript [30]. Similarly, Ferveo implements this pattern on Tendermint: validators submit PVSS instances via vote extensions to a designated DKG block, and the subsequent proposer verifies and aggregates a 2/3-stake supermajority into the next block [2,11].

Applying Janus to FROST

Threshold signature schemes like FROST [25,28] and FaFROST [8,3] are designed to operate seamlessly with distributed key generation. Our two-round protocol $\mathsf{Janus}_{\mathsf{2R}}$ realizes $\mathcal{F}_{\mathsf{DKG}}^{t,n}$, producing ideally distributed keys that serve as a drop-in replacement in any DLog-based threshold cryptosystem that supports our key structure [25,28,8,3]. In this section, we discuss how our round-optimal protocol $\mathsf{Janus}_{\mathsf{1R}}$, which realizes the shifted functionality $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$, integrates into downstream schemes despite the adversary's ability to induce an additive key shift. We also detail how to apply the proactive key refresh extension from Section 6, using FaFROST [8,3] as a concrete example throughout.

Unforgeability under Additive Key Shifts We first show that the adversarial shift tolerated by $\mathsf{Janus}_{\mathsf{1R}}$ is compatible with the unforgeability proof of FaFROST [8,3] under the Algebraic One-More Discrete Logarithm (AOMDL) assumption. The key observation is that the AOMDL-based reduction treats the public key as an opaque group element: it embeds an AOMDL challenge $X = g^{x_0}$ into the public key, simulates signing queries using the AOMDL discrete-logarithm helper oracle, and extracts the secret key from a successful forgery.

In our setting, the reduction embeds the challenge by setting the global public key to $\mathsf{pk} \leftarrow X \cdot g^{\delta} = g^{x_0+\delta}$, where δ is the shift submitted by the adversary to $\mathcal{F}_{\mathsf{S\text{-}DKG}}^{t,n}$. Simulation of signing queries proceeds as before using the DLog oracle. When the adversary provides a forgery, the extraction procedure in [8] recovers the discrete logarithm $\mathsf{sk} = x_0 + \delta$.[11] Because the reduction explicitly knows δ from the DKG functionality, it trivially solves the challenge by computing $x_0 = \mathsf{sk} - \delta$. We conjecture that the same argument holds for most threshold cryptosystems whose security reduces to AOMDL, and hence follow a similar proof strategy.

Compatibility with Proactive Key Refresh The proactive key refresh mechanism discussed in Section 6 is compatible with the unforgeability reduction of FaFROST [8,3] (and likely extends to most AOMDL-based schemes) without introducing additional assumptions. Because proactive refresh updates individual secret shares while preserving the global secret key, signing oracle simulations proceed identically across refresh epochs. Consequently, extracting the discrete logarithm from a forgery in any epoch yields the fixed key $\mathsf{sk} = x_0 + \delta$, allowing the reduction to recover the challenge $x_0 = \mathsf{sk} - \delta$ regardless of the number of elapsed refresh cycles.

8 Conclusion

We presented Janus, the first distributed key generation suite that provides end-to-end adaptive security against a dishonest majority while maintaining drop-in compatibility with standard discrete-logarithm verifiers and smart contracts. By concealing individual key shares while outputting an unblinded master public key, Janus eliminates trusted setups and closes the share-leakage vulnerability exploited by the Crites-Stewart attack in Feldman VSS-based deployments.

Our two-round protocol $\mathsf{Janus}_{\mathsf{2R}}$ realizes a DKG functionality with ideally distributed keys, while our round-optimal protocol $\mathsf{Janus}_{\mathsf{1R}}$ achieves single-round setup subject to an additive key shift that still integrates seamlessly into downstream AOMDL-based schemes like FaFROST. Coupled with compact 259-byte dispute proofs for automated slashing, proactive share refresh, and sub-400 ms execution scaling to 512 nodes, Janus provides a practical, provably secure foundation for decentralized custody and blockchain infrastructure.

[11] Extraction failure in [8] depends solely on collision events that are independent of δ and hold for any fixed public key.

Acknowledgements

We thank Tux Pacific for pointing out that modeling π^{pk} through $\mathcal{F}_{\text{nizk}}$ in a previous version of this work revealed honest key contributions one round too early. This work was partially supported by Deutsche Forschungsgemeinschaft as part of the Research and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393541319/GRK2475/1-2019) and through grant 442893093. This work is also partially supported by SBA Research (SBA-K1 NGC), which is a COMET Center within the COMET – Competence Centers for Excellent Technologies Programme and funded by BMIMI, BMWET, and the federal state of Vienna. The COMET Programme is managed by FFG. Paul Gerhart's research has been supported by the Google PhD Fellowship in Privacy, Safety, and Security.

References

1. Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334. Springer, Berlin, Heidelberg, August 2004.
2. Anoma. Ferveo documentation: Distributed key generation. https://github.com/anoma/ferveo/blob/main/book/src/dkg.md. Accessed: 2026-08-18.
3. Renas Bacho, Yanbo Chen, Julian Loss, Stefano Tessaro, and Chenzhi Zhu. Adaptively secure partially non-interactive threshold schnorr signatures in the AGM. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part I*, volume 16541 of *LNCS*, pages 33–62. Springer, Cham, May 2026.
4. Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Adaptively secure three-round threshold Schnorr signatures from DDH. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 390–422. Springer, Cham, August 2025.
5. Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Glacius: Threshold schnorr signatures from DDH with full adaptive security. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part II*, volume 15602 of *LNCS*, pages 304–334. Springer, Cham, May 2025.
6. Renas Bacho and Alireza Kavousi. SoK: Dlog-based distributed key generation. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 614–632. IEEE Computer Society Press, May 2025.
7. Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. GRandLine: Adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 941–955. ACM Press, October 2024.
8. Ruben Baecker, Paul Gerhart, Davide Li Calsi, Luigi Russo, Dominique Schröder, and Arkady Yerukhimovich. Fully adaptive FROST in the algebraic group model from falsifiable assumptions. Cryptology ePrint Archive, Paper 2025/1950, 2025.
9. Ruben Baecker, Paul Gerhart, Daniel Rausch, and Dominique Schröder. A fully-adaptive threshold partially-oblivious PRF. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 569–597. Springer, Cham, August 2025.
10. C. Baum, T. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai. Pesto: Proactively secure distributed single sign-on, or how to trust a hacked server. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 587–606, Los Alamitos, CA, USA, sep 2020. IEEE Computer Society.
11. Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Report 2022/898, 2022.
12. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.
13. Dan Boneh, Aditi Partap, and Lior Rotem. Proactive refresh for accountable threshold signatures. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 140–159. Springer, Cham, March 2024.
14. Xavier Boyen, Stanislaw Jarecki, Phillip Nazarian, Jiayu Xu, and Tianyu Zheng. Two-factor authentication can harden servers against offline password search. Cryptology ePrint Archive, Paper 2026/317, 2026.
15. Benedikt Bünz, Kevin Choi, and Chelsea Komlo. Golden: Lightweight non-interactive distributed key generation. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology – CRYPTO 2026*, pages 232–262, Cham, 2026. Springer Nature Switzerland.

16. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Cham, April / May 2018.
17. Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iUC: Flexible universal composability made simple. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 191–221. Springer, Cham, December 2019.
18. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Department of Computer Science, ETH Zurich, March 1997. Available at ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/.
19. Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.
20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.
21. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. Cryptology ePrint Archive, Report 2021/060, 2021.
22. Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Berlin, Heidelberg, August 1999.
23. Yanbo Chen. Round-efficient adaptively secure threshold signatures with rewinding. *CiC*, 2(2):16, 2025.
24. Yi-Hsiu Chen and Yehuda Lindell. Optimizing and implementing fischlin's transform for UC-secure zero-knowledge. Cryptology ePrint Archive, Report 2024/526, 2024.
25. Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023.
26. Ran Cohen, Daniel Collins, Pouyan Forghani, Juan Garay, and Vassilis Zikas. On the round complexity of dishonest-majority MPC. Cryptology ePrint Archive, Paper 2026/1277, 2026.
27. Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures. RFC 9591, June 2024.
28. Elizabeth C. Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro, and Chenzhi Zhu. On the adaptive security of FROST. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 480–511. Springer, Cham, August 2025.
29. Elizabeth C. Crites and Alistair Stewart. A plausible attack on the adaptive security of threshold Schnorr signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 457–479. Springer, Cham, August 2025.
30. Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. Distributed randomness using weighted VUFs. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 314–344. Springer, Cham, May 2025.
31. Sourav Das and Ling Ren. Adaptively secure BLS threshold signatures from DDH and co-CDH. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 251–284. Springer, Cham, August 2024.
32. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Berlin, Heidelberg, August 2001.
33. D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.
34. Hanna Ek, Kelsey Melissaris, and Lawrence Roy. Adaptively secure, universally composable distributed generation of discrete-logarithm based keys from standard assumptions. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology – CRYPTO 2026*, pages 263–295, Cham, 2026. Springer Nature Switzerland.
35. Hanwen Feng, Tiancheng Mai, and Qiang Tang. Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2636–2650. ACM Press, October 2024.
36. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.
37. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005.

38. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Berlin, Heidelberg, May 1999.
39. Paul Gerhart. Janus prototype, 2026. GitHub repository.
40. Paul Gerhart, Davide Li Calsi, Luigi Russo, and Dominique Schröder. Fully-adaptive two-round threshold schnorr signatures from DDH. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part II*, volume 16542 of *LNCS*, pages 251–283. Springer, Cham, May 2026.
41. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017.
42. Dennis Hofheinz and Victor Shoup. GNUC: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, July 2015.
43. Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 221–242. Springer, Berlin, Heidelberg, May 2000.
44. Stanislaw Jarecki and Phillip Nazarian. Adaptively secure threshold blind BLS signatures and threshold oblivious PRF. In Goichiro Hanaoka and Bo-Yin Yang, editors, *ASIACRYPT 2025, Part VI*, volume 16250 of *LNCS*, pages 601–634. Springer, Singapore, December 2025.
45. Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024.
46. Jonathan Katz. Round-optimal, fully secure distributed key generation. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 285–316. Springer, Cham, August 2024.
47. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Berlin, Heidelberg, August 2006.
48. Darya Kaviani. Fireblocks mpc wallet.
49. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.
50. Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: A simple and expressive model for universal composability. *Journal of Cryptology*, 33(4):1461–1584, October 2020.
51. Yehuda Lindell. Digital asset management with mpc (whitepaper).
52. National Institute of Standards and Technology. Nist first call for multi-party threshold schemes. Accessed: 2026-04-25.
53. NEAR Protocol. NEAR MPC: Distributed key generation. https://github.com/near/mpc/blob/main/crates/threshold-signatures/docs/dkg.md. Accessed: 2026-08-18.
54. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Berlin, Heidelberg, April 1991.
55. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992.
56. Deevashwer Rathee, Emma Dauterman, Allison Li, and Raluca Ada Popa. Chorus: Secret recovery with ephemeral client committees. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 2307–2326, 2026.
57. Tim Ruffing, Jonas Nick, Illia Melnyk, Mariia Zhvanko, and Sivaram Dhakshinamoorthy. ChillDKG: Distributed key generation for FROST. Bitcoin Improvement Proposal draft, https://github.com/BlockstreamResearch/bip-frost-dkg. Accessed: 2026-08-18.
58. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous Schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.
59. Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. *CiC*, 1(2):19, 2024.
60. SSV Labs. ssv-dkg: Distributed key generation tool for the SSV network. https://github.com/ssvlabs/ssv-dkg. Accessed: 2026-08-16.

A Open Science

We provide all artifacts necessary to evaluate the core contributions of this work. In particular, we release our full implementation of the adaptive DKG protocols, including both the one-round and two-round constructions, together with all proof system instantiations used in our evaluation. The codebase includes:

- a complete Rust implementation of the protocols,
- all cryptographic components (e.g., Pedersen VSS, Chaum–Pedersen proofs, Fischlin transform),
- benchmarking code used to generate the reported results,
- configuration files for all evaluated parameter settings,
- scripts to reproduce all benchmarks, and
- documentation explaining the structure of the code and how to run experiments.

All benchmark data reported in the paper are included in the repository.

https://github.com/pGerhart/Janus

The repository is self-contained and does not require access to external datasets or proprietary software. All experiments can be reproduced on standard hardware using the provided scripts. We do not rely on any artifacts that cannot be shared. All components required to validate our methodology and results are included in the repository.

B Ethical Considerations

This paper proposes Janus, a suite of adaptively secure and universally composable Distributed Key Generation (DKG) protocols. We evaluate Janus through formal cryptographic proofs in the UC framework and performance measurements of a prototype implementation. Following the stakeholder-based analysis, we discuss the ethical implications of this research.

Stakeholders The primary stakeholders are operators of decentralized infrastructure, including validator sets for Proof-of-Stake networks and decentralized custody providers. End users who rely on threshold-based systems for asset protection are the beneficiaries of the security improvements proposed herein. Secondary stakeholders include the broader cryptographic and blockchain communities, as well as developers of threshold primitives.

Potential Benefits and Societal Impact The primary benefit of this work is the mitigation of systemic risks in decentralized finance. As demonstrated by recent cryptanalysis, existing threshold Schnorr deployments that rely on unique key commitments are vulnerable to adaptive forgery attacks. In high-stakes environments, such vulnerabilities could lead to a catastrophic loss of funds or network-wide liveness failure.

By providing a key-share-hiding DKG that supports identifiable abort, Janus allows for the cryptographically attributable penalization (e.g., "slashing") of malicious actors without compromising adaptive security. Furthermore, the native support for proactive key refresh prevents the long-term accumulation of secret shares by an adversary, extending the secure lifetime of decentralized keys.

Potential Harms and Dual Use Our work is fundamentally defensive and constructive. While the techniques described could theoretically be studied by an adversary to understand the limits of threshold security, we do not introduce new attack vectors or exploit previously unknown vulnerabilities in deployed systems. Unlike "attack-oriented" research, our contribution provides the mathematical and systemic tools necessary to harden existing infrastructure against known adaptive threats. We assess the risk of dual-use harm to be low and outweighed by the critical need for secure setup protocols in the blockchain industry.

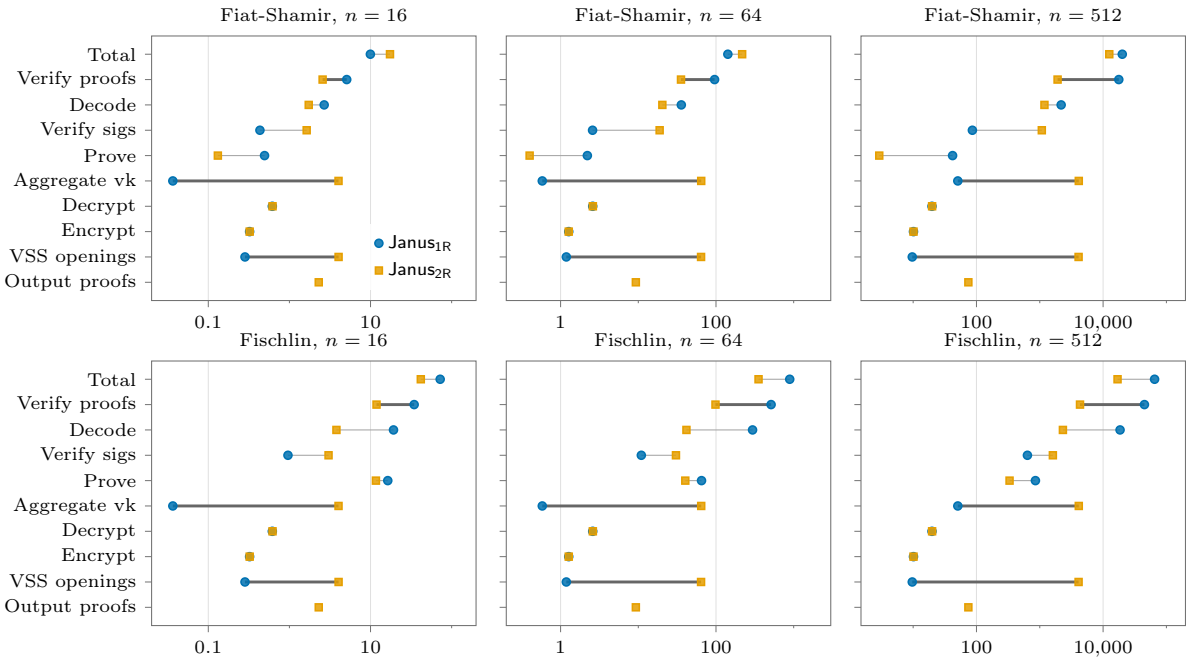

Fig. 11: The complete version of Figure 10 with $n \in \{16, 64, 512\}$,

Research Methodology and Data Handling The evaluation of Janus was conducted using mathematical modeling in the UC framework and local benchmarking of a prototype implementation. No human subjects were involved, and no data from real-world users or production blockchain environments were collected or analyzed. All experimental measurements were performed in controlled test environments using synthetic parameters. Consequently, this research does not involve personal data, deception, or unauthorized interaction with live services.

Implementation Risks and Secure Erasure We acknowledge that Janus relies on the secure erasure of internal state to achieve its security guarantees. We ethically disclose that implementations must ensure the physical zeroization of historical secret shares to prevent an adversary from accumulating state snapshots across epochs. We encourage practitioners to utilize Hardware Security Modules (HSMs) or Trusted Execution Environments (TEEs) to meet these architectural requirements.

Decision to Publish We believe that publishing these protocols serves the public interest. The current "gap" between theoretical adaptive security and practical DKG performance creates a dangerous fragility in decentralized systems. By providing a low-latency, adaptively secure, and interoperable DKG ($\mathsf{pk} = g^{\mathsf{sk}}$), we advance the state of the art in defensive cryptography and align with NIST's standardization goals for robust multi-party schemes.

C Measured Computation Cost

Additional numbers behind Section 5 can be seen in Tables 4 to 6 and Figures 11 and 12. All settings are $t = n$, the most expensive threshold the protocols admit.

(t, n)	Fiat-Shamir (Sim)	Fischlin (UC, small)	Fischlin (UC, large)
Initiate			
$(16, 16)$	1.3	18.3	22.5
$(32, 32)$	2.7	35.4	46.3
$(64, 64)$	5.7	73.3	100.5
$(128, 128)$	13.1	157.6	238.0
$(256, 256)$	33.0	369.3	–
$(512, 512)$	93.7	993.5	–
Output, one core			
$(16, 16)$	8.7	54.2	130.3
$(32, 32)$	32.3	203.8	494.1
$(64, 64)$	134.9	795.0	1 930
$(128, 128)$	610.1	3 220	7 740
$(256, 256)$	3 260	13 790	–
$(512, 512)$	19 930	62 060	–
Output, all cores			
$(16, 16)$	1.8	6.3	13.9
$(32, 32)$	5.1	22.2	49.9
$(64, 64)$	16.4	80.6	187.6
$(128, 128)$	63.1	306.7	735.2
$(256, 256)$	300.3	1 280	–
$(512, 512)$	1 740	5 580	–

Table 4: Per-party computation time (ms) for $\mathsf{Janus}_{\mathsf{1R}}$, one core against all cores. *Fischlin-large* up to $n = 128$.

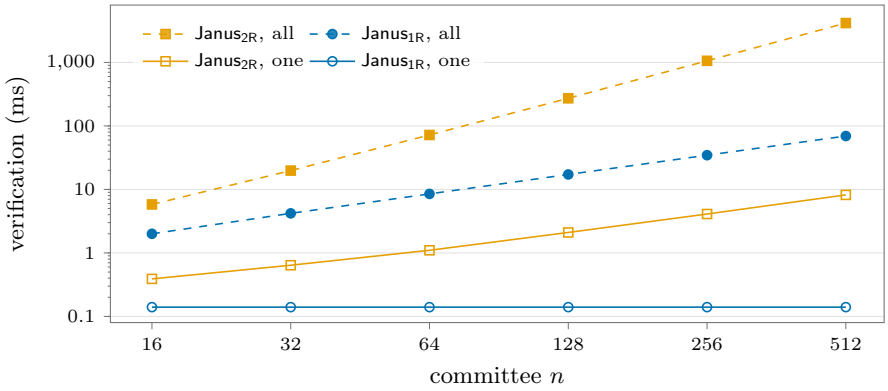

Fig. 12: Complaint verification cost, log-log axes.

D Additional Technical Details

D.1 Sending Outputs to Multiple Parties

Most models for universal composability, such as [17,19,50,42] including Canetti's UC model in particular, are inherently single-threaded: There is only ever a single machine instance I active at a time which is allowed to send a message/input/output m to just one other machine instance I'. This then deactivates I such that I cannot send any other messages and instead activates I' so it can process m (see, e.g., the definition of the external-write instruction in Canetti's UC model [19, Section 3.1.2]). It is hence impossible for an ideal

(t, n)	Fiat-Shamir (Sim)	Fischlin (UC, small)	Fischlin (UC, large)
Initiate			
$(16, 16)$	1.0	12.6	3.6
$(32, 32)$	2.0	23.1	5.9
$(64, 64)$	4.2	45.8	10.8
$(128, 128)$	9.9	91.5	21.8
$(256, 256)$	26.5	192.3	–
$(512, 512)$	80.4	373.9	–
Finalize, one core			
$(16, 16)$	13.2	24.0	41.1
$(32, 32)$	48.5	75.6	116.2
$(64, 64)$	186.5	261.9	367.6
$(128, 128)$	730.0	968.2	1 280
$(256, 256)$	2 840	3 660	–
$(512, 512)$	11 240	14 260	–
Finalize, all cores			
$(16, 16)$	2.2	3.2	4.7
$(32, 32)$	6.3	8.8	12.4
$(64, 64)$	20.2	27.4	37.3
$(128, 128)$	72.9	95.0	123.6
$(256, 256)$	271.4	347.2	–
$(512, 512)$	1 040	1 330	–
Output, one core			
$(16, 16)$	3.2	3.2	3.2
$(32, 32)$	6.3	6.3	6.3
$(64, 64)$	12.6	12.6	12.6
$(128, 128)$	25.3	25.2	25.2
$(256, 256)$	50.5	50.5	–
$(512, 512)$	101.0	101.0	–
Output, all cores			
$(16, 16)$	1.1	1.1	1.1
$(32, 32)$	2.4	2.4	2.3
$(64, 64)$	4.5	4.5	4.6
$(128, 128)$	8.6	8.6	8.5
$(256, 256)$	16.7	16.7	–
$(512, 512)$	32.8	32.9	–

Table 5: Per-party computation time (ms) for $\mathsf{Janus}_{\mathsf{2R}}$, one core against all cores. *Fischlin-large* up to $n = 128$.

functionality to send outputs to multiple different parties simultaneously, as each party corresponds to a separate machine instance that cannot be activated at the same time.

There are several different standard modeling techniques to implement close approximate versions of parallel outputs within a single-threaded computational framework. Here we consider an implementation using a polling approach that was also employed in the ideal synchronous channel functionality $\mathcal{F}_{\text{SYN}}$ given by Canetti in [19, Figure 14] to make outputs available to multiple parties at the same time. Specifically, whenever we write "send outputs to a set of parties" in an ideal functionality, e.g., for aborts in our ideal DKG functionalities, formally, the functionality only writes those outputs into party-specific output buffers kept in the functionality's local state before handing control over to the adversary. That is, the functionality does not actually *send* the output; it only stores it. The functionality instead implicitly provides an interface for parties

Operation	Janus_{1R}	Janus_{2R}
Initiate		
Prove well-formedness	65.14	40.27
Encrypt shares	1.27	1.27
Verify received round		
Verify one proof	8.00	1.52
Verify all n proofs	512.02	98.50
Decode messages	295.58	41.66
Verify channel signatures	10.94	30.54
Decrypt shares	2.59	2.60
Check VSS openings	1.19	64.26
Aggregate verification keys	0.58	64.55
Output		
Verify public-key contributions	–	3.66
Verify key-equality proofs	–	5.64

Table 6: Component-wise cost in ms, *Fischlin-small*, $(t, n) = (64, 64)$. Middle block: Janus_{1R}'s Output, Janus_{2R}'s Finalize.

to poll the current content of their respective output buffers, say, by sending an input `getAbortOutputs` for the case of abort outputs. Upon receiving such a polling query from some party, the functionality immediately returns the current content of that party's output buffer and clears that buffer.

Observe that this implementation accomplishes the original goal of the operation "send outputs to a set of parties" as the output still becomes available to be read for all parties in the set at the same time. For example, if an abort occurs in one of our functionalities, then every honest party will learn the identity of a malicious party as soon as they decide to read their output buffer.

D.2 Broadcast Channel

For our protocols, we assume a synchronous authenticated broadcast channel shared by protocol parties $\mathcal{P}_i$, $1 \leq i \leq n$. That is, the channel guarantees that all messages that are sent become available immediately at the same time to all parties, alongside the identity of the sender. Formally, such a broadcast channel can be modeled via a simple ideal functionality $\mathcal{F}_{\text{bc}}$ that has only one operation:

– Upon receiving input m from any party $\mathcal{P}_i$:
Output (m, i) towards all parties $\mathcal{P}_j$ and the adversary $\mathcal{A}$.

D.3 Random Oracle

We consider an ideal random oracle functionality $\mathcal{F}_{\text{ro}}$ which works in the standard way. That is, the functionality internally keeps a dynamically populated list L that tracks input-output pairs (x, h). It provides a single operation:

– Upon receiving x from any party $\mathcal{P}_i$ or the adversary $\mathcal{A}$:
If there is a pair (x, h) in L, return h. Otherwise, choose a fresh h uniformly at random from the output domain, add (x, h) to L, and return h.

We note that the random oracle functionality $\mathcal{F}_{\text{ro}}$ is treated as local in our proofs (cf. [16]). The simulator is hence responsible for simulating the random oracle and can, in particular, program its outputs as long as they remain indistinguishable from truly random ones.

D.4 Non-Interactive Zero Knowledge Proofs

Our protocols use five proof languages, and we do not model all of them in the same way. For L^{VSS}, L^{comeq}, L^{dec} and L^{enc} we model NIZKs via an ideal functionality $\mathcal{F}_{\mathrm{nizk}}$, which is a standard type of functionality commonly used in the literature. For completeness in Figure 13 we provide a definition of $\mathcal{F}_{\mathrm{nizk}}$ taken from [14].

The functionality is parameterized by an efficient relation $\mathcal{R}$ over statement-witness pairs.

- On $(\texttt{Prove}, ssid, \mathsf{stmt}, \mathsf{wit})$ from party $\mathcal{P}$: If $\mathcal{R}(\mathsf{stmt}, \mathsf{wit}) = 1$ then send $(\texttt{Prove}, \mathcal{P}, ssid, \mathsf{stmt})$ to $\mathcal{A}^*$, await response $(\texttt{Proof}, ssid, \pi)$ from $\mathcal{A}^*$, save record $(\mathsf{stmt}, \mathsf{wit}, \pi)$, and send $(\texttt{Proof}, ssid, \pi)$ to $\mathcal{P}$.
- On $(\texttt{Verify}, ssid, \mathsf{stmt}, \pi)$ from party $\mathcal{V}$: If there exists no record $(\mathsf{stmt}, \mathsf{wit}, \pi)$ then send $(\texttt{Verify}, \mathcal{V}, ssid, \mathsf{stmt}, \pi)$ to $\mathcal{A}^*$, await response $(\texttt{Witness}, ssid, \mathsf{wit})$, and save record $(\mathsf{stmt}, \mathsf{wit}, \pi)$. In either case, send $(\texttt{Verification}, ssid, \mathcal{R}(\mathsf{stmt}, \mathsf{wit}))$ to $\mathcal{V}$.

Fig. 13: Non-interactive zero-knowledge proof functionality $\mathcal{F}_{\mathrm{nizk}}$.

The remaining language, L^{pk}, cannot be modeled this way. On a proving request, $\mathcal{F}_{\mathrm{nizk}}$ forwards the statement to the ideal adversary before it returns a proof, so a statement becomes visible to the adversary when it is proved rather than when it is published. $\mathsf{Janus}_{\mathsf{2R}}$ depends on the gap between those two moments. Party $\mathcal{P}_i$ computes pk_i and π_i^{pk} already in Round 1 so that it may erase $(a_{i,0}, b_{i,0})$ before Round 2, yet broadcasts neither value until Round 2, so that a rushing adversary cannot choose its own contribution after seeing the honest ones. Proving L^{pk} through $\mathcal{F}_{\mathrm{nizk}}$ would thus disclose every honest pk_i in Round 1 and give up exactly the guarantee that the second round buys. We keep a concrete proof system for L^{pk} instead, and state our requirements on it in the game-based style. The proof is then computed locally, and no information about pk_i leaves $\mathcal{P}_i$ before the Round 2 broadcast.

Syntax. Fix an NP-relation $\mathcal{R}$ and its language $\mathsf{L} = \{\mathsf{stmt} \mid \exists \mathsf{wit}\colon (\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}\}$. A non-interactive proof system for $\mathcal{R}$ in the random-oracle model is a triple $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$. Here $\mathsf{Setup}(1^\lambda, \mathcal{R})$ outputs a common reference string crs, $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit})$ outputs a proof π, and $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$ outputs a bit. For L^{pk} the string crs consists of the group parameters $(\mathbb{G}, p, g)$, the Pedersen bases g and v, and the random oracle H. We let H be the same $\mathcal{F}_{\mathrm{ro}}$ that the protocol already uses for hashed ElGamal, under a distinct domain-separation prefix, which keeps the two query spaces disjoint. Disjointness is important, since $\mathcal{S}_{\mathsf{nizk}}$ programs challenge points in Game 1 while the CDH embedding of Game 4 programs key-derivation points, and neither may overwrite the other. Our prototype separates every hash use by its own domain string, so the challenge derivation of π^{pk} and the key derivation of the hashed ElGamal cannot collide. Completeness requires that $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \mathcal{P}(\mathsf{crs}, \mathsf{stmt}, \mathsf{wit})) = 1$ for every $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$.

Following [32,41], we define both security notions relative to a stateful simulator $\mathcal{S}_{\mathsf{nizk}}$ that operates in two modes. In the first mode, $\mathcal{S}_{\mathsf{nizk}}$ answers a random-oracle query and may program the answer, while in the second mode, it produces a proof for a statement without a witness. We write $\mathcal{S}_{\mathsf{nizk},1}(\cdot)$ for the induced random-oracle wrapper, $\mathcal{S}_{\mathsf{nizk},2}(\mathsf{stmt}, \mathsf{wit})$ for the proving wrapper that returns a simulated proof whenever $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$ and $\perp$ otherwise, and $\mathcal{S}'_{\mathsf{nizk},2}(\mathsf{stmt})$ for the proving wrapper that simulates on any statement. All three wrappers share the state of $\mathcal{S}_{\mathsf{nizk}}$, and $\mathcal{Q}$ denotes the set of statement-proof pairs that the proving wrapper has returned.

Definition 2 (Zero-knowledge). *A proof system Π is zero-knowledge for $\mathcal{R}$ if there exists a simulator $\mathcal{S}_{\mathsf{nizk}}$ such that for every PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{zk}}_{\Pi}(\mathcal{A}) := \left|\Pr[\mathcal{A}^{\mathsf{H},\mathcal{P}}(\mathsf{crs}) = 1] - \Pr[\mathcal{A}^{\mathcal{S}_{\mathsf{nizk},1},\mathcal{S}_{\mathsf{nizk},2}}(\mathsf{crs}) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$ in both experiments.

Definition 3 (Simulation extractability). *A proof system Π is simulation extractable for $\mathcal{R}$ with respect to $\mathcal{S}_{\mathsf{nizk}}$ if for every PPT adversary $\mathcal{A}$ there is a PPT extractor $\mathcal{E}_{\mathsf{nizk}}$ such that*

$$\mathsf{Adv}^{\mathsf{sim\text{-}ext}}_{\Pi}(\mathcal{A}) := \mathsf{Pr}[\begin{matrix}\mathcal{V}^{\mathcal{S}_{\mathsf{nizk},1}}(\mathsf{crs},\mathsf{stmt},\pi)=1 \wedge \\ (\mathsf{stmt},\pi)\notin\mathcal{Q} \ \wedge \ (\mathsf{stmt},\mathsf{wit})\notin\mathcal{R}\end{matrix}] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda,\mathcal{R})$, $(\mathsf{stmt},\pi) \leftarrow \mathcal{A}^{\mathcal{S}_{\mathsf{nizk},1},\mathcal{S}'_{\mathsf{nizk},2}}(\mathsf{crs})$ *and* $\mathsf{wit} \leftarrow \mathcal{E}_{\mathsf{nizk}}(\mathsf{st},\mathsf{stmt},\pi)$ *for the final state* st *of* $\mathcal{S}_{\mathsf{nizk}}$.

Simulation extractability implies simulation soundness, and for L^{pk} the weaker notion would not suffice. Writing $L^{\mathsf{pk}}_{g,v} = \{((\mathsf{pk},c),(a,b)) \mid \mathsf{pk} = g^a \ \wedge \ c = g^a v^b\}$, every pair $(\mathsf{pk},c) \in \mathbb{G}^2$ lies in the language, because v generates $\mathbb{G}$ and hence $c \cdot \mathsf{pk}^{-1}$ is a power of v. A simulation-soundness guarantee, which speaks only about statements outside the language, is thus vacuous here, and the entire content of π^{pk} is knowledge of an opening. Since $c_{i,0}$ does not constrain pk_i, no membership statement about the pair carries information.

Instantiation. We instantiate L^{pk} with the two-equation Σ-protocol that proves knowledge of (a,b) satisfying $\mathsf{pk} = g^a$ and $c = g^a v^b$ with a single first message and a shared response for a. We make it non-interactive over $\mathcal{F}_{\mathrm{ro}}$ with the statement, including the session transcript and the index of the proving party, bound into the challenge. The one-equation variant that proves knowledge of $\log_v(c \cdot \mathsf{pk}^{-1})$ is insufficient, since a malicious party may shift its contribution to $\mathsf{pk} \cdot v^{-\delta}$ for an arbitrary δ of its choosing and still produce an accepting proof. Both the Fiat–Shamir and the Fischlin transform serve for that step, and the choice between them is a trade-off rather than a requirement of the proof. Fiat–Shamir [36] keeps π^{pk} at its Σ-protocol size and satisfies Definition 2 perfectly in the programmable random-oracle model, but it reaches Definition 3 through the forking lemma [12], at the cost of the usual quadratic loss. Fischlin's transform [37] yields a straight-line extractor and avoids that loss, at the increased proof sizes evaluated in Section 5. Since π^{pk} is a single proof per party per execution, both are practical. For our prototype we use Fiat–Shamir.

D.5 Public Key Infrastructure

For our proof, we assume a PKI with strong key binding, where all parties prove knowledge of their secret keys, for example, via a NIZK during key registration. Formally, we model this setup assumption as an ideal PKI functionality that allows a party to register a public key only if it can also provide the corresponding secret key. Anyone can retrieve a party's public key.

Upon first activation, obtain the ElGamal group parameters, including group $\mathbb{G}$ and generator g, from the shared CRS. Then:

- On $(\texttt{Register},\mathsf{pk},\mathsf{sk})$ from party $\mathcal{P}_i$: Check that $\mathsf{pk} \in \mathbb{G}$, $\mathsf{pk} = g^{\mathsf{sk}}$, and no public key has been registered yet for $\mathcal{P}_i$. Record pk for $\mathcal{P}_i$ and return $(\texttt{Register},\texttt{success})$. Otherwise return $(\texttt{Register},\texttt{Error})$.
- On $(\texttt{Retrieve},i)$ from a party $\mathcal{P}_j$ or the adversary: If a public key pk has been registered for party $\mathcal{P}_i$, return $(\texttt{Retrieve},\mathsf{pk})$. Otherwise return $(\texttt{Retrieve},\texttt{Error})$.

Fig. 14: DLog-based Public Key Infrastructure functionality $\mathcal{F}_{\mathrm{pki}}$.

In our protocol, all parties generate and register their public encryption keys upon initialization before any other operation. If the adversary corrupts a party at this stage, it is free to choose the public key that this corrupted party registers (in particular, this public key might not follow the same distribution as an honestly generated key and can depend on the public keys of honest parties who have already registered theirs), as long as it also provides the corresponding secret key. This perfectly captures strong key binding.

Strong key binding We use a PKI with strong key binding, rather than a simpler PKI with only weak key binding (where malicious parties can register arbitrary public keys without proving knowledge of a secret key), to allow for an optimized batch-variant of Hashed ElGamal. Instead of sending standard ciphertexts $(g^{r_1}, \mathsf{H}((\mathsf{pk}_1^E)^{r_1}) \oplus m_1), \ldots, (g^{r_n}, \mathsf{H}((\mathsf{pk}_n^E)^{r_n}) \oplus m_n)$, we use a single ephemeral randomness r that is reused for all n ciphertexts: $(g^r, \mathsf{H}((\mathsf{pk}_1^E)^r) \oplus m_1, \ldots, \mathsf{H}((\mathsf{pk}_n^E)^r) \oplus m_n)$. This reduces the message size by $n-1$ group elements per party.

However, this introduces an edge case in our reduction. In one game-hop, we replace the public encryption key of an honest party $\mathcal{P}^*$ with a CDH challenge A, and the ephemeral randomness g^r for a ciphertext sent by party $\mathcal{P}_i$ with the second component of the challenge B. Since B is reused for all ciphertexts, $\mathcal{S}$ must construct the ciphertext sent from $\mathcal{P}_i$ to a malicious party $\mathcal{P}_j$ as $\mathsf{H}(B^{\mathsf{sk}_j^E}) \oplus m_j$. It can only do this if it knows sk_j^E, because it does not know the discrete logarithm of $B = g^b$ to compute $(\mathsf{pk}_j^E)^b$ normally.

Security for PKIs with weak key binding Our security proof and protocol permit two variations where a PKI with weak key binding suffices, each offering a different trade-off.

First, if we assume the stronger gapDH assumption rather than the standard CDH assumption, our protocol is provably secure even with weak key binding. In this variant, $\mathcal{S}$ sends a uniformly random ciphertext from party $\mathcal{P}_i$ to party $\mathcal{P}_j$ and observes queries to $\mathcal{F}_{\text{ro}}$. For each query X, $\mathcal{S}$ uses the DDH oracle provided by the gapDH assumption to detect the correct decryption query by checking if (pk_j^E, B, X) is a valid DDH tuple. Once it detects the correct query, it programs $\mathcal{F}_{\text{ro}}$ on that input to allow for correct decryption.

Second, if parties use fresh randomness for each ciphertext, as done in standard Hashed ElGamal, the security of this slightly less efficient protocol variant can be reduced to CDH even with a PKI that provides only weak key binding. In this case, $\mathcal{S}$ only embeds the CDH challenge into a single ciphertext directed at $\mathcal{P}^*$ and samples known ephemeral randomness for all remaining ciphertexts, avoiding the need to know the malicious parties' secret keys.