Round-Optimal Robust Distributed Key Generation

Jonathan Katz[*]

Abstract

Protocols for distributed (threshold) key generation (DKG) in the discrete-logarithm setting have received a tremendous amount of attention in the past few years. Several synchronous DKG protocols have been proposed, but most such protocols are either not fully secure (in the sense of simulatability) or are not *robust* in that they allow even a single malicious party to prevent successful generation of a key.

In this paper we explore the round complexity of (robust) DKG in the honest-majority setting where robust DKG is feasible. On the negative side, we show the impossibility of one-round (robust) DKG protocols regardless of any prior setup the parties have. On the positive side, we show various two-round—and hence, round-optimal—protocols for robust DKG offering tradeoffs in terms of their efficiency, necessary setup, and required assumptions.

1 Introduction

In a $(t+1)$-out-of-n *threshold cryptosystem*, a secret key is shared among n parties such that any collection of $t+1$ honest parties can jointly perform some cryptographic operation, while an adversary compromising up to t parties cannot. The past few years have seen a significant interest in threshold signing, in particular, motivated by its application to the protection of cryptocurrency wallets as well as other applications such as threshold access control, random beacons, and distributed-protocol design. Research on threshold signatures has developed threshold protocols for the ECDSA, Schnorr, and BLS signature schemes, as well as protocols for distributed key generation (DKG) [39, 8, 20, 19, 30, 33, 2, 26, 7, 12, 24, 41, 3, 1, 14, 25, 36, 34] in the discrete-logarithm setting that underlies those schemes. Threshold protocols based on this work are being used extensively by companies such as Fireblocks, Dfns, and Coinbase (among others), and there has also been interest in standardizing such protocols [11, 6].

DKG protocols have been studied in both the synchronous [39, 8, 20, 19, 26, 7, 12, 24, 41, 3, 36, 34] and asynchronous [30, 33, 2, 1, 14, 25] settings. Although the asynchronous model may be more appropriate for large-scale protocols with globally distributed parties, the synchronous model is what is assumed in practice for small-scale protocols running in a local network (as is the case for the companies mentioned above). We consider the synchronous setting in this paper.

An important property that is often overlooked in the context of (synchronous[1]) threshold protocols is *robustness*—aka *guaranteed output delivery*—namely, the requirement that a protocol should produce correct output (e.g., a valid signature) whenever it is executed, even in the presence

[*]Dfns Labs. `jkatz@dfns.co, jkatz2@gmail.com`. This work was not part of my University of Maryland duties.

[1]Existing asynchronous threshold protocols generally do guarantee robustness, and of course such protocols are also secure when run in a synchronous network. However, asynchronous protocols can only tolerate $t < n/3$, and generally have higher round complexity than protocols designed specifically for the synchronous setting.

of malicious behavior. The relative lack of attention given to robustness may be due to an excessive focus on the n-out-of-n case—or, more generally, the dishonest-majority setting—where robustness is impossible. (Indeed, most recent work on threshold cryptography has focused on that setting.) Or, the fact that robustness requires $2t + 1$ parties to participate in an execution of the protocol may be viewed as a waste of resources (even though having $t + 1$ parties run a protocol that doesn't produce any output is more wasteful). Alternately, there may be a belief that robustness is easy to achieve by re-running the protocol among a different group of participants when an execution fails. These arguments for neglecting robustness are questionable, and robustness is often a critical property that must be explicitly ensured in many practical deployments of threshold cryptography. For one thing, besides improved security an additional benefit of distributing a key is increased availability, which requires robustness. And while it is often possible to add robustness to existing, non-robust protocols by running the protocol multiple times [40], doing so is inefficient and sometimes leads to subtle security flaws (e.g., an attacker may be able to bias the output or learn multiple outputs). If robustness is required, it should be incorporated directly.

In this work, we focus on the round complexity of robust DKG protocols in the discrete-logarithm setting. Roughly speaking, the goal in this context is for n parties to distributively generate a public key $y = g^x$ (where g is a generator of a cyclic group $\mathbb{G}$) such that the parties hold $(t+1)$-out-of-n Shamir secret shares $\{\sigma_i\}$ of the private exponent x. Robustness means that a public key y is always generated, and honest parties always hold correct shares of the corresponding x. In fact, we aim for more, both in terms of functionality (we also require that parties can compute public "commitments" $\{g^{\sigma_i}\}$ to each others' shares, as is often required by threshold protocols) and security (which we define in terms of simulatability relative to an appropriate ideal functionality).

Few synchronous DKG protocols in the literature achieve robustness. To the best of our knowledge, the most round-efficient explicit construction of a robust DKG protocol is the 6-round protocol by Gennaro et al. [20]. One could apply known results [23, 22, 13] for generic secure multiparty computation (MPC) with guaranteed output delivery in the honest-majority setting to obtain a 3-round DKG protocol assuming a common reference string (CRS), or a 2-round protocol assuming a CRS and a public-key infrastructure (PKI), but the resulting protocols would not be particularly efficient; moreover, they would require strong primitives (like fully homomorphic encryption or indistinguishability obfuscation) and cryptographic assumptions beyond those typically assumed in the discrete-logarithm setting. Komlo et al. [34] recently showed a 4-round protocol that is fair but not robust; although they do not consider simulation-based definitions, it seems plausible that their protocol realizes functionality $\mathcal{F}_{\mathsf{KeyGen}}^{\perp,\mathsf{fair}}$ (cf. Appendix A).

1.1 Our Results

We work in a simulation-based framework, and define robustness for DKG protocols via a corresponding ideal functionality for fully secure key generation that (in particular) ensures guaranteed output delivery. For completeness, we also discuss in Appendix A several other ideal functionalities for key generation one might consider. Although such simulation-based definitions seem the most natural way to define security in this context, several recent works have instead given (different) game-based definitions that are often quite complex and somewhat difficult to interpret. We hope that our definitional treatment, while not new, provides useful clarity for future work.

Since robustness is impossible[2] in the dishonest-majority setting, we assume an honest majority.

[2]This follows by a reduction from coin tossing to DKG, plus the well-known impossibility result of Cleve [10].

In that setting, we show several constructions of round-efficient robust DKG protocols that offer tradeoffs in terms of their computational efficiency, necessary setup, and cryptographic assumptions:

1. In Section 4, we show a framework for constructing 2-round, robust DKG protocols assuming a PKI and a CRS. In particular, we improve upon the recent work of Komlo et al. [34] in terms of both round complexity and security. Although the same round complexity could be obtained using prior work on generic MPC, our framework uses weaker cryptographic assumptions and leads to more-efficient constructions. In particular, we propose an efficient instantiation of our framework based on the El Gamal and Paillier encryption schemes in the random-oracle model.

2. In Section 5, we show how to construct a 2-round, robust DKG protocol based on a CRS alone. (This implies a similar protocol with no setup in the random-oracle model.) This is particularly interesting since, to the best of our knowledge, such a result does not follow from existing results on generic MPC. One drawback of this construction is that, for[3] $n > 3$, it relies on multiparty non-interactive key exchange (NIKE), something currently known to exist only based on strong assumptions. (We refer to Koppula et al. [35] for a survey of known results.) It also has complexity linear in $\binom{n}{t}$, and thus technically only solves the problem for a constant number of parties. As such, we view this result as primarily demonstrating the difficulty of proving the *impossibility* of 2-round (robust) DKG in the CRS model.

3. In the full version of the paper, we show a protocol in the random-oracle model and no setup that has the following properties: After one round of preprocessing (run by the parties themselves), the parties can generate an unbounded number of keys via repeated invocations of a 2-round protocol.

In all cases our simulation-based proofs of security do not use rewinding; hence our protocols are also universally composable (subject to caveats regarding the use of the random oracle in the UC framework). Our final protocol can also be modified easily so as to be adaptively secure.

Complementing the above results, we also prove the impossibility of one-round (robust) DKG. Such a result does not follow from prior work, as existing lower bounds on the round complexity of MPC with guaranteed output delivery [18, 23, 38, 22, 13] take advantage of the fact that honest parties have input, and thus those bounds do not extend to no-input functionalities like DKG. Our negative result is obtained by observing that a DKG protocol implies coin tossing with no additional rounds, and then proving impossibility of one-round coin-tossing protocols. While such a result may seem intuitively obvious, our impossibility result (1) holds even for non-robust DKG protocols, and applies even when there is only a single corrupted party; (2) rules out one-round protocols regardless of any prior setup or idealized models (like the random-oracle model) used; and (3) gives quantitative bounds on the inherent insecurity of any one-round DKG protocol.

We leave it as an interesting open question whether there exist two-round, robust DKG protocols with no setup and without random oracles, or whether such protocols exist in the CRS model based on weaker assumptions than multiparty NIKE. Positive results would be interesting even for a constant number of parties $n > 3$, and even for sub-optimal corruption thresholds (e.g., $t < n/3$).

[3] For $n = 3$ the standard decisional Diffie-Hellman assumption suffices.

2 Background

2.1 Preliminaries

Notation. We let $\mathbb{G}$ denote a cyclic group of prime order q and let $g \in \mathbb{G}$ be a fixed generator. We let $\mathbb{Z}_q = \{0, \ldots, q-1\}$ (viewed as a field), and $[k] = \{1, \ldots, k\}$. We write "$\leftarrow$" for probabilistic assignment and ":=" for deterministic assignment. In particular, if R is a randomized algorithm then $y \leftarrow R(x)$ means that we run R on input x and a uniform random tape to obtain y, whereas $y := R(x; \omega)$ means that we (deterministically) run R on input x and random tape ω to obtain y.

System and communication model. We assume n parties $P_1, \ldots, P_n$, with $t < n$ a bound on the number of corrupted parties. We work in the standard synchronous communication model where parties are connected by pairwise private and authenticated channels in addition to a public broadcast channel.[4] We always consider a *rushing* adversary, by which we mean that in each round the corrupted parties receive all messages sent by honest parties in that round before having to send their own messages. In some cases we rely on a public-key infrastructure (PKI), by which we mean that all parties hold the same vector $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ of public keys and each honest party P_i holds the secret key sk_i associated with pk_i. Parties who are corrupted at the outset may generate their public keys in an arbitrary fashion, possibly depending on public keys of the honest parties.

On defining round complexity. The round complexity of a protocol execution in the synchronous model is fairly straightforward to define. However, some protocols run for a different number of rounds in different executions. For some protocols the round complexity is a random variable. In other work, a distinction is made between the *optimistic* round complexity (when all parties are honest) and the *worst-case* round complexity (which holds for arbitrary adversarial behavior). For example, the recent protocol by Komlo et al. [34] uses only 3 rounds when all parties are honest, but even a single corrupted party can cause the protocol to use 4 rounds. All the protocols we describe run for a fixed number of rounds in every execution.

2.2 Cryptographic Building Blocks

Shamir secret sharing. We use the standard notion of Shamir secret sharing. To share a secret $x \in \mathbb{Z}_q$ in a $(t+1)$-out-of-n fashion, a dealer chooses uniform coefficients $f_1, \ldots, f_t \in \mathbb{Z}_q$ and forms the polynomial $f(X) := x + \sum_{i=1}^{t} f_i \cdot X^i$; it then defines shares $\{\sigma_i\}_{i=1}^{n}$ by setting $\sigma_i := f(i)$, and distributes σ_i to P_i via a private channel. It is useful to also define $\sigma_0 := f(0) = x$ (though note that σ_0 is not a share that is sent to any party), and we write $\{\sigma_i\}_{i=0}^{n} \leftarrow \mathsf{SS}_t(x)$ to denote the process by which these values are generated. No information about x is revealed by the shares of any t parties, but $x = \sigma_0$ can be reconstructed from the shares of any $t+1$ parties. More generally, it is possible to reconstruct the value of f at any point from its values at any $t+1$ points using Lagrange interpolation. This means that for any $(t+1)$-size set $S \subset \mathbb{Z}_q$ and any $k \in \mathbb{Z}_q$ it is possible to (publicly) compute coefficients $\{\lambda_{i,k}^{S}\}_{i \in S}$ such that $f(k) = \sum_{i \in S} \lambda_{i,k}^{S} \cdot f(i)$. We denote the interpolation of the value of $f(k)$ from the values $\{f(i)\}_{i \in S}$ by $\mathsf{interpolate}(k, S, \{f(i)\}_{i \in S})$. In particular, when $S \subset [n]$ we have $x = \mathsf{interpolate}(0, S, \{\sigma_i\}_{i \in S})$.

[4] We leave for future work the question of round-efficient, robust DKG without a broadcast channel. Note that difficulties associated with achieving robustness arise even when a broadcast channel is available, and so it makes sense to decouple the problem of agreement from the problem of robustness.

It is a standard fact that interpolation can also be done "in the exponent," i.e., given any $(t+1)$-size set $S \subset \mathbb{Z}_q$, the values $\{g^{f(i)}\}_{i \in S}$, and $k \in \mathbb{Z}_q$, it is possible to compute $g^{f(k)} = \prod_{i \in S} \left(g^{f(i)}\right)^{\lambda^S_{i,k}}$. Overloading notation slightly, we also denote this by $\mathsf{interpolate}(k, S, \{g^{f(i)}\}_{i \in S})$.

Feldman verifiable secret sharing. Feldman's variant of Shamir secret sharing [16] allows parties to verify that they have received shares consistent with a polynomial of the correct degree. We describe a slight variant of the usual scheme that is functionally equivalent. Here, the dealer generates $\{\sigma_i\}_{i=0}^n$ as above, and then broadcasts the $t+1$ values $y_0 := g^{\sigma_0}, \ldots, y_t := g^{\sigma_t}$ (in addition to sending σ_i to P_i via a private channel, as before). We write $(\{y_i\}_{i=0}^t, \{\sigma_i\}_{i=1}^n) \leftarrow \mathsf{FVSS}_t(x)$ to denote the process by which the indicated values are generated. Given the broadcasted information, P_i can check correctness of the share σ_i it received from the dealer by setting $S := \{0, \ldots, t\}$ and then verifying that $g^{\sigma_i} \stackrel{?}{=} \mathsf{interpolate}(i, S, \{y_j\}_{j \in S})$. The behavior of P_i in case verification fails (including the case when P_i does not receive anything from the dealer) is protocol-dependent.

Feldman's scheme leaks the value $y_0 = g^x$, but for our applications this will not be a problem.

CPA-secure encryption. We use the standard notion of CPA-security [31] for a public-key encryption scheme defined by algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Non-interactive zero-knowledge (NIZK) proofs. We rely on a variant of (unbounded) simulation-sound NIZK proofs [15]. We give an informal definition, and refer elsewhere [15, 32] for formal details. Let R be an NP relation. A collection of efficient algorithms $(\mathsf{GenCRS}, \mathcal{P}, \mathcal{V}, \mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{KE})$ is an *ID-based simulation-sound NIZK proof system for R* if the following hold:

- **Completeness:** For all $(x, w) \in R$ and all $i \in [n]$,

$$\Pr[\mathsf{crs} \leftarrow \mathsf{GenCRS}; \pi \leftarrow \mathcal{P}(\mathsf{crs}, i, x, w) : \mathcal{V}(\mathsf{crs}, i, x, \pi) = 1] = 1.$$

- **Adaptive, multi-theorem zero knowledge:** For every efficient adversary $\mathcal{A}$, we have

$$\Pr[\mathsf{crs} \leftarrow \mathsf{GenCRS} : \mathcal{A}^{\mathcal{P}^*(\mathsf{crs},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1] \approx \Pr[(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_1 : \mathcal{A}^{\mathsf{Sim}_2^*(\mathsf{td},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1],$$

  where $\mathcal{P}^*(\mathsf{crs}, i, x, w)$ returns $\mathcal{P}(\mathsf{crs}, i, x, w)$ if $(x, w) \in R$ (and $\perp$ otherwise) and $\mathsf{Sim}_2^*(\mathsf{td}, i, x, w)$ returns $\mathsf{Sim}_2(\mathsf{td}, i, x)$ if $(x, w) \in R$ (and $\perp$ otherwise).

- **Unbounded, identity-based simulation soundness:** The standard notion of simulation soundness requires, essentially, that even if an adversary is given multiple simulated proofs, it cannot generate a new, valid proof for a false statement. This is formalized via a knowledge-extraction requirement, so that whenever the adversary outputs a (new) valid proof for some statement x, a knowledge extractor KE can extract a witness corresponding to x. We also bind proofs to identities, and require that an adversary given multiple simulated proofs with respect to one set of identities $\mathcal{H}$ cannot generate a valid proof (whether new or not) for a false statement with respect to any identity outside of $\mathcal{H}$.

  More formally, the success probability of every efficient adversary $\mathcal{A}$ in the following experiment should be small:

  1. Run $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_1$, and choose a uniform bit $b \in \{0, 1\}$.
  2. Run $\mathcal{A}(\mathsf{crs})$ to obtain a set $\mathcal{H} \subset [n]$. Then give $\mathcal{A}$ access to two oracles:

(a) The first oracle takes input (i, x, w) and returns $\perp$ if $i \notin \mathcal{H}$ or $(x, w) \notin R$. Otherwise, it returns $\mathsf{Sim}_2(\mathsf{td}, i, x)$.

(b) The second oracle takes input (i, x, π) and returns $\perp$ if $i \in \mathcal{H}$ or $\mathcal{V}(\mathsf{crs}, i, x, \pi) = 0$. Otherwise:

- If $b = 0$ it returns 1.
- If $b = 1$ it computes $w \leftarrow \mathsf{KE}(\mathsf{td}, i, x, \pi)$ and returns 1 iff $(x, w) \in R$ (and returns $\perp$ otherwise).

3. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, and succeeds iff $b = b'$.

3 Defining Secure (Robust) Distributed Key Generation

We use a standard simulation-based notion of security that we briefly summarize below. For self-containment, our definition is for stand-alone security, but it could easily be adapted to the universal composability framework. For simplicity, we assume a static corruption model.

Fix some n-party DKG protocol Π. A real-world execution of the protocol in the presence of an adversary $\mathcal{A}$ proceeds as follows (note that parties running Π have no initial input):

1. $\mathcal{A}$ specifies a set $\mathcal{C} \subset [n]$ of corrupted parties.

2. The honest parties run Π with the corrupted parties. Honest parties follow the protocol as prescribed, while the actions of the corrupted parties are controlled by $\mathcal{A}$.

3. When an honest parties terminates, it outputs a value as prescribed by the protocol.

4. The *view* of $\mathcal{A}$ in this execution consists of the randomness used by $\mathcal{A}$, any messages sent to any of the corrupted parties by an honest party, and any messages sent on the broadcast channel by an honest party.

We let $\textsc{real}_{\Pi,\mathcal{A}}$ be the random variable consisting of (1) the identities $\mathcal{C}$ of the corrupted parties, (2) the vector of outputs of the honest parties, and (3) the view of $\mathcal{A}$ at the end of the execution.

Fix an n-party (randomized) functionality $\mathcal{F}$. An ideal execution of $\mathcal{F}$ in the presence of an adversary $\mathcal{S}$ proceeds as follows:

1. $\mathcal{S}$ specifies a set $\mathcal{C} \subset [n]$ of corrupted parties.

2. Honest parties interact with $\mathcal{F}$ according to the prescribed interface. $\mathcal{S}$ controls what a corrupted party sends to $\mathcal{F}$, and observes all values that $\mathcal{F}$ sends to a corrupted party.

3. An honest party outputs the value sent to it by $\mathcal{F}$.

4. The adversary $\mathcal{S}$ may output an arbitrary function of its view.

We let $\textsc{ideal}_{\mathcal{F},\mathcal{S}}$ be the random variable consisting of (1) the identities $\mathcal{C}$ of the corrupted parties, (2) the vector of outputs of the honest parties, and (3) the output of $\mathcal{S}$.

We say that protocol Π *t-securely realizes* $\mathcal{F}$ if for any efficient adversary $\mathcal{A}$ corrupting at most t parties there is an efficient adversary $\mathcal{S}$ such that no efficient distinguisher D can distinguish $\textsc{real}_{\Pi,\mathcal{A}}$ and $\textsc{ideal}_{\mathcal{F},\mathcal{S}}$. In the concrete setting we adopt here, one could quantify security by bounding the running times of $\mathcal{A}, \mathcal{S}$, and D as well as the acceptable distinguishing advantage

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

Let $\mathcal{C}'$ be an arbitrary set of size t with $\mathcal{C} \subseteq \mathcal{C}' \subset [n]$.

1. Receive $\{\sigma_i\}_{i \in \mathcal{C}}$ from the adversary.
2. Choose $x \leftarrow \mathbb{Z}_q$ and set $y := g^x$. Choose uniform $\sigma_i \in \mathbb{Z}_q$ for $i \in \mathcal{C}' \setminus \mathcal{C}$.
3. Let f be the polynomial of degree at most t such that $f(0) = x$ and $f(i) = \sigma_i$ for $i \in \mathcal{C}'$. Set $\sigma_i := f(i)$ for $i \in [n] \setminus \mathcal{C}'$.
4. For $i \in [n]$, set $y_i := g^{\sigma_i}$. Let $Y = (y_1, \ldots, y_n)$.
5. For $i \in [n]$, send (y, σ_i, Y) to P_i. Also send (y, Y) to the adversary.

Figure 1: Ideal functionality for fully secure key generation, parameterized by t, n.

of D. In an asymptotic setting, one would instead provide parties with a security parameter κ as input, and parameterize the random variables $\textsc{real}_{\Pi,\mathcal{A}}$ and $\textsc{ideal}_{\mathcal{F},\mathcal{S}}$ by κ; security would then require that for any probabilistic polynomial-time (PPT) $\mathcal{A}$ corrupting at most t parties there is a PPT adversary $\mathcal{S}$ such that no PPT distinguisher D (possibly with access to non-uniform auxiliary input) can distinguish $\textsc{real}_{\Pi,\mathcal{A}}(\kappa)$ and $\textsc{ideal}_{\mathcal{F},\mathcal{S}}(\kappa)$ with advantage that is not negligible (in κ).

Given this definitional framework, we can define security for key-generation protocols by defining an appropriate ideal functionality. In our context, the basic requirement is for the ideal functionality to choose a uniform private key $x \in \mathbb{Z}_q$ and give each party P_i the corresponding public key $y = g^x$ along with P_i's share σ_i in a $(t+1)$-out-of-n sharing of x. Many threshold cryptosystems also require the parties to each hold a vector of "commitments" $Y = (g^{\sigma_1}, \ldots, g^{\sigma_n})$ to the shares of the other parties (such commitments are often used by parties to prove correctness of their actions in a subsequent protocol using the generated key), and this is incorporated into the ideal functionality as well. We ensure robustness by defining the ideal functionality such that it always provides output to the honest parties. These requirements are encapsulated by the ideal functionality $\mathcal{F}_{\mathsf{KeyGen}}$ shown in Figure 1 that corresponds to "fully secure" key generation.

Notes on the definition. Functionality $\mathcal{F}_{\mathsf{KeyGen}}$ does not assume the adversary corrupts exactly t parties. In particular, the adversary may corrupt *no* parties, and in that case (y, Y) is given to the adversary in step 5. (That part of step 5 is redundant if at least one party is corrupted.) Translated to the security of a protocol Π realizing $\mathcal{F}_{\mathsf{KeyGen}}$, this means that an adversary who eavesdrops on an execution of Π (but corrupts no parties) is allowed to learn the public key y and the parties' commitments Y. This is acceptable, as those values are generally treated as public.

A more subtle aspect of $\mathcal{F}_{\mathsf{KeyGen}}$ is that it allows the adversary to choose the shares of the corrupted parties in step 1; we stress that the remaining shares are still uniform subject to that constraint. One could strengthen the functionality to prevent this behavior (see Appendix A), but we are not aware of any (natural[5]) application of key generation where the difference matters, and weakening the functionality as we have done potentially allows for more-efficient protocols. Alternately, one could consider an even weaker definition where the adversary is allowed to choose its shares *after* learning the public key y. In general, one advantage of working in the simulation-

[5]It is not difficult to show contrived counterexamples: e.g., one could have a threshold signing protocol Π' in which all parties broadcast their share if any component of the commitment vector Y is the identity. Π' is clearly insecure if the adversary can choose its key shares.

based framework is that it is simple to define other notions of security for key-generation protocols (by giving different ideal functionalities), and very clear what security properties are being added or sacrificed. We provide other examples of alternate ideal functionalities in Appendix A.

The multi-session extension of $\mathcal{F}_{\mathsf{KeyGen}}$. When shared state is used across multiple executions of a protocol (as is the case in some of our protocols), technically one should show that repeated execution of the protocol securely realizes the *multi-session extension* of the corresponding ideal functionality [9]. We do not formalize this notion here, but remark that it is not hard to verify that our protocols satisfy this requirement.

4 Two-Round Protocols in the PKI+CRS Model

In this section we show robust, 2-round DKG protocols, assuming a PKI and a common reference string. We first describe a general framework for constructing 2-round protocols realizing $\mathcal{F}_{\mathsf{KeyGen}}$, and then discuss a concrete instantiation of this framework based on Paillier encryption and efficient zero-knowledge proofs used previously in the context of threshold cryptography [7]. In Section 4.3 we show how to realize a stronger DKG functionality, still using only two rounds.

4.1 A General Framework

The starting point of our protocol is the usual approach of having every party act as the dealer in a $(t+1)$-out-of-n secret sharing scheme, and then having the parties homomorphically combine the results. This approach yields a 1-round protocol, in which each party P_i does the following:

1. Choose a uniform value $x_i \in \mathbb{Z}_q$ and compute $\{\sigma_{i,j}\}_{j=0}^{n} \leftarrow \mathsf{SS}_t(x_i)$.
2. For each $j \in [n]$, broadcast $y_{i,j} := g^{\sigma_{i,j}}$ and send $\sigma_{i,j}$ to P_j over a private channel.[6]

Each party P_i computes its share $\sigma_i := \sum_{j\in[n]} \sigma_{j,i}$ and the commitment $y_j := \prod_{k\in[n]} y_{k,j}$ to the share of any (other) party P_j. Letting $S = [t+1]$, all parties can also compute the public key as $\mathsf{interpolate}(0, S, \{y_j\}_{j\in S})$.

The above description assumes semi-honest behavior. While all parties can verify that each party P_i broadcasted correct information (by checking that the exponents of the $\{y_{i,j}\}_{j=1}^{n}$ lie on a degree-t polynomial), the protocol does nothing to address a malicious adversary who sends an incorrect share to another party. This can be addressed using two additional rounds: one round in which a party can *complain* about some other party who sent it incorrect shares, and a second round that allows honest parties to respond to complaints.[7] Protocols based on complaints seem to inherently require at least three rounds (if not more).

A natural idea is to have parties use NIZK proofs to publicly prove correct behavior. Note, however, that such proofs will be of no use for proving correctness of values sent over private channels. (Parties already have the ability to check correctness of values they receive, but now we want parties to additionally be able to check correctness of the values sent to all other parties.) To account for this, we can modify the protocol so that instead of sending shares over (ideal) private

[6]Feldman verifiable secret sharing could be used to reduce the number of broadcast messages, but that optimization is not important for our current high-level discussion.

[7]While this addresses the particular problem of incorrect shares, the resulting protocol is not fully secure as it still suffers from the bias problem discussed below.

channels, parties instead send shares encrypted using a public-key encryption scheme. This requires parties to distribute public encryption keys, which can be done using either an additional round or by assuming a PKI. Using this approach we obtain the 1-round protocol in which each party P_i does as follows:

1. Choose a uniform value $x_i \in \mathbb{Z}_q$ and compute $\{\sigma_{i,j}\}_{j=0}^{n} \leftarrow \mathsf{SS}_t(x_i)$.
2. For each $j \in [n]$, broadcast $y_{i,j} := g^{\sigma_{i,j}}$ and an encryption of $\sigma_{i,j}$ under the public key of P_j. Additionally, give an NIZK proof of correct behavior.

Parties compute the public key, their shares, and commitments to other parties' shares as before, excluding the contributions from any parties whose NZIK proofs fail to verify. (We omit the details.) This exactly corresponds to having each party act as the dealer in a *publicly verifiable secret-sharing scheme* (PVSS) [42] (see [21] for a recent survey). The idea of using PVSS or something similar in the context of distributed key generation was also proposed by Boneh and Shoup [5, Section 22.4.2] and Groth [24]; the former achieve robustness using an approach requiring many more rounds (see below), while the latter does not achieve robustness at all.

NIZK proofs force parties to either behave correctly or (effectively) abort. But they are not enough to make the protocol secure! Indeed, even a single corrupted party can *bias* the public key by waiting until all other parties have sent their messages, locally running the protocol (honestly) multiple times, and then selecting which messages to send based on the public keys it computes in those executions. (Recall we assume a rushing adversary.) A natural way to address this is to have each party commit to g^{x_i} in the first round, and then give NIZK proofs relative to that commitment in a second round; this would not fully address the problem, however, since it would still allow an adversary to bias the resulting public key by deciding whether to *abort* (i.e., refuse to open its commitment) in the second round. Boneh and Shoup [5, Section 22.4.2] address this by assuming simultaneous broadcast, which can in turn be instantiated via a multi-round protocol. It does not seem possible to obtain a 2-round protocol using that approach.

Instead, we modify the protocol so that only encrypted shares—and no $\{y_{i,j}\}$ values—are sent in the first round. Parties continue to send NIZK proofs of correct behavior as before, and are excluded if their proofs fail to verify. Then, in the second round, each party uses the shares it received from all non-excluded parties to compute appropriate $\{y_{i,j}\}$ values, and broadcasts those values along with an NIZK proof that those values were computed correctly. (In fact, it suffices for each party P_i to just broadcast the commitment $y_i = g^{\sigma_i}$ to its final share σ_i.) As intuition for security, note first that because of the NIZK proofs adversarial behavior is effectively limited to aborting. Aborts in the first round cannot bias the key because the public key cannot be computed until the second round. On the other hand, aborts in the second round cannot introduce bias since the public key is *defined* at the end of the first round, in the sense that the same public key y will be computed by the honest parties regardless of what the malicious parties do in the second round. This is because the presence of $t + 1$ honest parties ensures that sufficiently many correct commitments will be broadcast to allow the public key to be computed. We formalize the resulting protocol in Figure 2. The protocol relies on zero-knowledge proofs for the following NP relations:

$$R_L = \left\{ \left(\{(\mathsf{pk}_j, c_j)\}_{j\in[n]},\ \ \{(\sigma_j, \omega_j)\}_{j\in[n]}\right) : \begin{array}{c} \exists \text{ polynomial } f \text{ of degree at most } t \text{ such that} \\ \forall j\ f(j) = \sigma_j \wedge c_j := \mathsf{Enc}_{\mathsf{pk}_j}(\sigma_j; \omega_j) \end{array} \right\}$$

$$R_{L'} = \left\{ \left((\mathsf{pk}, \{c_j\}_{j\in\mathcal{I}}, y),\ \ \mathsf{sk}\right) : \begin{array}{c} \mathsf{sk} \text{ is a secret key corresponding to } \mathsf{pk}; \\ y = g^{\sum_{i\in\mathcal{I}} \mathsf{Dec}_{\mathsf{sk}}(c_i)} \end{array} \right\}.$$

$\Pi_1^{t,n}$

We assume a PKI (with pk_i being the public key of P_i) and a common random string $\mathsf{crs}, \mathsf{crs}'$.

Round 1: Each party P_i does the following:

1. Choose uniform $x_i \in \mathbb{Z}_q$ and compute $\{\sigma_{i,j}\}_{j\in[n]} \leftarrow \mathsf{SS}_t(x_i)$. Then for all $j \in [n]$ choose uniform $\omega_{i,j} \in \{0,1\}^*$ and compute $c_{i,j} := \mathsf{Enc}_{\mathsf{pk}_j}(\sigma_{i,j}; \omega_{i,j})$.
2. Compute $\pi_i \leftarrow \mathcal{P}(\mathsf{crs}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j\in[n]}, \{(\sigma_{i,j}, \omega_{i,j})\}_{j\in[n]})$.
3. Broadcast $\{c_{i,j}\}_{j\in[n]}$ and π_i.

Round 2: Let $\mathcal{I} := \{i \in [n] : \mathcal{V}(\mathsf{crs}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j\in[n]}, \pi_i) = 1\}$. Each party P_i then does:

1. For $j \in \mathcal{I}$, compute $\sigma_{j,i} := \mathsf{Dec}_{\mathsf{sk}_i}(c_{j,i})$. Set $\sigma_i := \sum_{j\in\mathcal{I}} \sigma_{j,i}$ and $y_i := g^{\sigma_i}$.
2. Compute $\pi'_i \leftarrow \mathcal{P}'(\mathsf{crs}', i, (\mathsf{pk}_i, \{c_{j,i}\}_{j\in\mathcal{I}}, y_i), \mathsf{sk}_i)$.
3. Broadcast y_i and π'_i.

Output determination: Let $\mathcal{I}' := \{i \in \mathcal{I} : \mathcal{V}'(\mathsf{crs}', i, (\mathsf{pk}_i, \{c_{j,i}\}_{j\in\mathcal{I}}, y_i), \pi'_i) = 1\}$. Each party P_i then does:

1. For $j \in \mathcal{I}'$, let y_j be the value broadcast by P_j in round 2.
2. Let $\mathcal{I}''$ be the $t+1$ smallest indices in $\mathcal{I}'$. (If $|\mathcal{I}'| < t+1$, abort.) For $j \in [n] \setminus \mathcal{I}'$, set $y_j := \mathsf{interpolate}(j, \mathcal{I}'', \{y_i\}_{i\in\mathcal{I}''})$. Set $y := \mathsf{interpolate}(0, \mathcal{I}'', \{y_i\}_{i\in\mathcal{I}''})$.
3. Output $(y, \sigma_i, (y_1, \ldots, y_n))$.

Figure 2: A 2-round DKG protocol in the PKI+CRS model, parameterized by t, n. Languages L, L' associated with $\mathcal{P}, \mathcal{P}'$ are described in the text.

Theorem 1. *Assume* $(\mathsf{Gen}, \mathsf{Enc})$ *is a perfectly correct, CPA-secure encryption scheme and* $\mathcal{P}, \mathcal{P}'$ *are identity-based simulation-sound NIZK proof systems for relations* $R_L, R_{L'}$ *defined above. Then for* $t < n/2$*, protocol* $\Pi_1^{t,n}$ *t-securely realizes* $\mathcal{F}_{\mathsf{KeyGen}}^{t,n}$.

Proof. We define a simulator $\mathcal{S}$, given black-box access to an adversary $\mathcal{A}$, as follows:

Setup: $\mathcal{S}$ runs $\mathcal{A}$ to obtain a set $\mathcal{C}$ of corrupted parties with $|\mathcal{C}| \le t$. Let $\mathcal{H} := [n] \setminus \mathcal{C}$. Then $\mathcal{S}$ runs $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_1$ and $(\mathsf{crs}', \mathsf{td}') \leftarrow \mathsf{Sim}'_1$ and, for $i \in \mathcal{H}$, runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}$. It gives $\mathsf{crs}, \mathsf{crs}'$, and $\{\mathsf{pk}_i\}_{i\in\mathcal{H}}$ to $\mathcal{A}$. In return, $\mathcal{A}$ outputs $\{\mathsf{pk}_i\}_{i\in\mathcal{C}}$.

Round 1: To simulate the first round, $\mathcal{S}$ does:

1. For all $i \in \mathcal{H}$ do:
   (a) For $j \in \mathcal{C}$, choose uniform $\sigma_{i,j} \in \mathbb{Z}_q$ and compute $c_{i,j} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j}(\sigma_{i,j})$.
   (b) For $j \in \mathcal{H}$, compute $c_{i,j} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j}(0)$.
   (c) Compute $\pi_i \leftarrow \mathsf{Sim}_2(\mathsf{td}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j\in[n]})$.
   (d) Give $\{c_{i,j}\}_{j\in[n]}$ and π_i to $\mathcal{A}$ as the message broadcast by P_i.
2. In response, $\mathcal{A}$ sends $\{c_{i,j}\}_{j\in[n]}$ and π_i for all $i \in \mathcal{C}$. (If some corrupted party P_i aborts, it will be anyway be excluded from $\mathcal{C}_\mathcal{I}$ below.)

Let $\mathcal{C}_\mathcal{I} := \{i \in \mathcal{C} : \mathcal{V}(\mathsf{crs}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j\in[n]}, \pi_i) = 1\}$ and $\mathcal{I} := \mathcal{C}_\mathcal{I} \cup \mathcal{H}$. For $j \in \mathcal{C}_\mathcal{I}$ do:

- For $i \in \mathcal{H}$, compute $\sigma_{j,i} := \mathsf{Dec}_{\mathsf{sk}_i}(c_{j,i})$; then let f_j be the polynomial of degree at most t with $f_j(i) = \sigma_{j,i}$ for $i \in \mathcal{H}$. (If no such polynomial exists, abort.)

For $j \in \mathcal{C}$ compute $\sigma_j := \sum_{i \in \mathcal{H}} \sigma_{i,j} + \sum_{i \in \mathcal{C}_\mathcal{I}} f_i(j)$. Send $\{\sigma_j\}_{j \in \mathcal{C}}$ to $\mathcal{F}^{t,n}_{\mathsf{KeyGen}}$, and receive in return y and $Y = (y_1, \ldots, y_n)$.

Round 2: For $i \in \mathcal{H}$ do:

1. Compute $\pi'_i \leftarrow \mathsf{Sim}'_2(\mathsf{td}', i, (\mathsf{pk}_i, \{c_{j,i}\}_{j \in \mathcal{I}}, y_i))$.
2. Give y_i and π'_i to $\mathcal{A}$ as the message broadcast by P_i.

Output whatever $\mathcal{A}$ outputs.

We show that $\text{REAL}_{\Pi_1^{t,n},\mathcal{A}}$ is indistinguishable from $\text{IDEAL}_{\mathcal{F}^{t,n}_{\mathsf{KeyGen}},\mathcal{S}}$ via a sequence of hybrid experiments. We start by explicitly describing an experiment Expt_0 that corresponds to $\text{REAL}_{\Pi_1^{t,n},\mathcal{A}}$.

Experiment Expt_0. This experiment is defined as follows:

Setup: $\mathcal{A}$ outputs a set $\mathcal{C}$ of corrupted parties; let $\mathcal{H} = [n] \setminus \mathcal{C}$. Run $\mathsf{crs} \leftarrow \mathsf{GenCRS}$ and $\mathsf{crs}' \leftarrow \mathsf{GenCRS}'$ and, for $i \in \mathcal{H}$, run $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}$. Give $\mathsf{crs}, \mathsf{crs}'$, and $\{\mathsf{pk}_i\}_{i \in \mathcal{H}}$ to $\mathcal{A}$, who outputs $\{\mathsf{pk}_i\}_{i \in \mathcal{C}}$.

Round 1: For $i \in \mathcal{H}$ do:

1. Choose $x_i \leftarrow \mathbb{Z}_q$ and compute $\{\sigma_{i,j}\}_{j \in [n]} \leftarrow \mathsf{SS}_t(x_i)$. For all $j \in [n]$, choose $\omega_{i,j} \leftarrow \{0,1\}^*$ and compute $c_{i,j} := \mathsf{Enc}_{\mathsf{pk}_j}(\sigma_{i,j}; \omega_{i,j})$.
2. Compute $\pi_i \leftarrow \mathcal{P}(\mathsf{crs}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j \in [n]}, (\sigma_{i,j}, \omega_{i,j}))$.

Give $\{c_{i,j}\}_{i \in \mathcal{H}, j \in [n]}$ and $\{\pi_i\}_{i \in \mathcal{H}}$ to $\mathcal{A}$. In response, $\mathcal{A}$ sends $\{c_{i,j}\}_{i \in \mathcal{C}, j \in [n]}$ and $\{\pi_i\}_{i \in \mathcal{C}}$. (If some corrupted party aborts, it will be excluded from $\mathcal{C}_\mathcal{I}$.)

Round 2: Let $\mathcal{C}_\mathcal{I} := \{i \in \mathcal{C} : \mathcal{V}(\mathsf{crs}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j \in [n]}, \pi_i) = 1\}$ and $\mathcal{I} := \mathcal{C}_\mathcal{I} \cup \mathcal{H}$. For all $i \in \mathcal{H}$ do:

1. For $j \in \mathcal{I}$, compute $\sigma_{j,i} := \mathsf{Dec}_{\mathsf{sk}_i}(c_{j,i})$. Set $\sigma_i := \sum_{j \in \mathcal{I}} \sigma_{j,i}$ and $y_i := g^{\sigma_i}$.
2. Compute $\pi'_i \leftarrow \mathcal{P}'(\mathsf{crs}', i, (\mathsf{pk}_i, \{c_{j,i}\}_{j \in \mathcal{I}}, y_i), \mathsf{sk}_i)$.

Give $\{(y_i, \pi'_i)\}_{i \in \mathcal{H}}$ to $\mathcal{A}$. In response, $\mathcal{A}$ sends $\{(y_i, \pi'_i)\}_{i \in \mathcal{C}_\mathcal{I}}$.

Output determination: Let $\mathcal{I}' := \{i \in \mathcal{I} : \mathcal{V}'(\mathsf{crs}', i, (\mathsf{pk}_i, \{c_{j,i}\}_{j \in \mathcal{I}}, y_i), \pi'_i) = 1\}$. Then:

1. For $j \in \mathcal{I}'$, let y_j be the corresponding value sent by P_j in round 2.
2. Let $\mathcal{I}''$ be the $t+1$ smallest indices in $\mathcal{I}'$. (Since $\mathcal{H} \subseteq \mathcal{I}'$, such a set $\mathcal{I}''$ must exist.) For $j \in [n] \setminus \mathcal{I}'$, set $y_j := \mathsf{interpolate}(j, \mathcal{I}'', \{y_i\}_{i \in \mathcal{I}''})$. Set $y := \mathsf{interpolate}(0, \mathcal{I}'', \{y_i\}_{i \in \mathcal{I}''})$.

The output of the experiment is[8] $\mathcal{C}$, $(y, \{\sigma_i\}_{i \in \mathcal{H}}, (y_1, \ldots, y_n))$, and the output of $\mathcal{A}$.

[8]Technically the output includes the values of y and $(y_1, \ldots, y_n)$ output by each honest party, but it is easy to see that for this protocol those values will always be identical.

Experiment Expt_1**.** In this experiment we modify Expt_0 as follows: during setup, we now generate $\mathsf{crs}, \mathsf{crs}'$ (along with state $\mathsf{td}, \mathsf{td}'$) using simulators $\mathsf{Sim}_1, \mathsf{Sim}_1'$, respectively. Then, honest parties use Sim_2 in place of $\mathcal{P}$ in round 1, and use Sim_2' in place of $\mathcal{P}'$ in round 2.

Indistinguishability of this experiment and Expt_0 follows immediately from zero-knowledge of $\mathcal{P}, \mathcal{P}'$.

Experiment Expt_2**.** We modify Expt_1 as follows. Before round 2, for all $j \in \mathcal{C}_\mathcal{I}$ run the knowledge extractor $\mathsf{KE}(\mathsf{td}, j, \{(\mathsf{pk}_i, c_{j,i})\}_{i\in[n]}, \pi_j)$ to obtain $\{\sigma_{j,i}\}_{i\in\mathcal{H}}$; if those values do not lie on a polynomial f_j of degree at most t, abort. (We remark that if $|\mathcal{H}| = t+1$ then an abort can never occur here; however, the check is relevant if $|\mathcal{H}| > t+1$.) Otherwise, in the first step of round 2 do the following for all $i \in \mathcal{H}$: for $j \in \mathcal{C}_\mathcal{I}$, let $\sigma_{j,i}$ be the value extracted; for $j \in \mathcal{H}$, let $\sigma_{j,i}$ be the value chosen in round 1. Then compute $\sigma_i := \sum_{j\in\mathcal{I}} \sigma_{j,i}$ and $y_i := g^{\sigma_i}$ as before.

Indistinguishability of this experiment and Expt_1 follows from simulation-soundness of $\mathcal{P}$ and perfect correctness of the encryption scheme.

Experiment Expt_3**.** Here we modify Expt_2 by changing the first step of round 1 so that for $i, j \in \mathcal{H}$ we compute $c_{i,j}$ as $c_{i,j} \leftarrow \mathsf{Enc}_{pk_j}(0; \omega_{i,j})$. Indistinguishability of this and the previous experiment follows from CPA-security of the encryption scheme.

Experiment Expt_4**.** Here we revert the change made in Expt_2 by computing $\{\sigma_{j,i}\}_{j\in\mathcal{C}_\mathcal{I}, i\in\mathcal{H}}$ as $\sigma_{j,i} := \mathsf{Dec}_{\mathsf{sk}_i}(c_{j,i})$. (We continue to abort if for some $j \in \mathcal{C}_\mathcal{I}$ the $\{\sigma_{j,i}\}_{i\in\mathcal{H}}$ do not lie on a polynomial of degree at most t.) As before, indistinguishability of this and the previous experiment follow from simulation-soundness of $\mathcal{P}$ and perfect correctness of the encryption scheme.

Experiment Expt_5**.** We modify Expt_4 in the following way. Let $\mathcal{C}'$ be an arbitrary set of size t with $\mathcal{C} \subseteq \mathcal{C}' \subset [n]$. In the first step of round 1, for each $i \in \mathcal{H}$ and $j \in \mathcal{C}$ choose uniform $\sigma_{i,j} \in \mathbb{Z}_q$. Then before round 2, for each $i \in \mathcal{H}$ do:

1. Choose uniform $x_i \in \mathbb{Z}_q$ and, for $j \in \mathcal{C}' \setminus \mathcal{C}$, choose uniform $\sigma_{i,j} \in \mathbb{Z}_q$.
2. Let f_i be the polynomial of degree at most t with $f_i(0) = x_i$ and $f_i(j) = \sigma_{i,j}$ for $j \in \mathcal{C}'$.
3. For $j \in [n] \setminus \mathcal{C}'$ set $\sigma_{i,j} := f_i(j)$.

The $\{\sigma_{j,i}\}_{i,j\in\mathcal{H}}$ values thus defined are then used in the first step of round 2.

Since all that has changed was to defer from round 1 to round 2 the choice of the $\{x_i\}_{i\in\mathcal{H}}$ (and shares $\{\sigma_{i,j}\}_{i\in\mathcal{H}, j\in\mathcal{C}'\setminus\mathcal{C}}$ that are not used in round 1), information-theoretic security of Shamir secret sharing implies that Expt_5 is perfectly indistinguishable from Expt_4.

Experiment Expt_6**.** Note that in Expt_5, if the experiment is not aborted by the beginning of round 2 then for all $i \in \mathcal{I}$ we have defined a polynomial f_i of degree at most t; moreover, for all $i \in \mathcal{H}$ the value σ_i computed in the first step of round 2 satisfies $\sigma_i = \sum_{j\in\mathcal{I}} f_j(i)$. In Expt_6 we introduce the following additional step after round 2: for all $i \in \mathcal{C}$, compute $\sigma_i := \sum_{j\in\mathcal{I}} f_j(i)$; then abort if $g^{\sigma_i} \neq y_i$ for some $i \in \mathcal{C} \cap \mathcal{I}'$. The output determination step is also modified so that, when the experiment has not aborted, we set $y := g^{\sum_{j\in\mathcal{I}} f_j(0)}$ and $y_i := g^{\sum_{j\in\mathcal{I}} f_j(i)}$ for $i \in [n]$.

Simulation-soundness of $\mathcal{P}'$ and perfect correctness of the encryption scheme imply that Expt_6 is indistinguishable from Expt_5.

Experiment Expt_7**.** Observe that in Expt_6 the values $\{\sigma_i\}_{i\in\mathcal{C}}$ can be computed after round 1: this is so even though the polynomials $\{f_i\}_{i\in\mathcal{H}}$ are not yet defined at that point, because the values

$\{f_i(j)\}_{i\in\mathcal{H},j\in\mathcal{C}}$ *are* defined at that point. With the $\{\sigma_i\}_{i\in\mathcal{C}}$ thus defined, we now modify Expt_6 in the following way: in the first step of round 2, choose uniform $x \in \mathbb{Z}_q$ and for $i \in \mathcal{C}' \setminus \mathcal{C}$ choose uniform $\sigma_i \in \mathbb{Z}_q$; let f be the polynomial of degree at most t with $f(0) = x$ and $f(i) = \sigma_i$ for $i \in \mathcal{C}'$. Then set $\sigma_i := f(i)$ for $i \in \mathcal{H} \setminus \mathcal{C}'$.

It is easy to see that Expt_7 is perfectly indistinguishable from Expt_6, and moreover that Expt_7 is statistically indistinguishable from $\text{IDEAL}_{\mathcal{F}^{t,n}_{\mathsf{KeyGen}},\mathcal{S}}$. □

4.2 An Instantiation using Paillier Encryption

We briefly sketch a protocol that can be viewed as an instantiation of Π_1 based on the Paillier (additively homomorphic) encryption scheme. (Note that several prior works [37, 17, 28] also show how to construct PVSS protocols from Paillier encryption.) Each party publishes a Paillier public key; correctness of public keys can be demonstrated existing NIZK proofs if desired [7]. Additionally, the common random string now includes a uniform $h \in \mathbb{G}$ that will be used as a public key for the El Gamal encryption scheme. We let $\mathsf{Enc}_{\mathsf{pk}_i}(\cdot)$ denote Paillier encryption using the public key pk_i of P_i, and let $\mathsf{Enc}_h(\cdot)$ denote El Gamal encryption using h. Let $S = \{0\} \cup [t]$. The protocol then proceeds as follows:

Round 1: Each P_i chooses uniform $x_i \in \mathbb{Z}_q$ and computes $(\{y_{i,j}\}_{j\in S}, \{\sigma_{i,j}\}_{j\in[n]}) \leftarrow \mathsf{FVSS}_t(x_i)$. It then computes $C_{i,j} \leftarrow \mathsf{Enc}_h(y_{i,j})$ for $j \in S$, and $c_{i,j} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j}(\sigma_j)$ for $j \in [n]$. It broadcasts $\{C_{i,j}\}_{j\in S}$ and $\{c_{i,j}\}_{j\in[n]}$. Additionally, for each $j \in [n]$, it broadcasts an NIZK proof (cf. [7]) that the value encrypted in $c_{i,j}$ is equal to the discrete logarithm of the value encrypted in $\mathsf{interpolate}(j, S, \{C_{i,k}\}_{k\in S})$ (where we again overload notation to let $\mathsf{interpolate}$ refer to homomorphic interpolation of El Gamal ciphertexts).

Round 2: Each party P_i computes $\mathcal{I}$, σ_i, and y_i analogously to the way those values are computed in Π_1. It then broadcasts y_i along with an NIZK proof (cf. [7]) that the discrete logarithm of y_i is equal to the value encrypted by $c^*_i \stackrel{\text{def}}{=} \prod_{j\in\mathcal{I}} c_{j,i}$.

Output determination: Parties compute output as in Π_1.

We leave optimization and implementation of this approach to future work.

4.3 Realizing a Stronger Ideal Functionality

We can adapt our framework to realize the stronger functionality $\widehat{\mathcal{F}}_{\mathsf{KeyGen}}$ (cf. Appendix A), still using only two rounds. Since we view this as primarily of theoretical interest, we only provide a sketch of the details. The main idea is that instead of having the parties each generate shares of a public key (which are then added together), we now have the parties generate *shares of shares* of a public key. By doing so, corrupted parties do not learn their shares of the public key until the second round, by which time they are already committed to the shares they generated and distributed in the first round. Thus, the protocol proceeds as follows:

Round 1: Each party P_i does the following: Choose uniform $x_i \in \mathbb{Z}_q$ and compute the first-level sharing $\{\sigma_{i,j}\}_{j\in[n]} \leftarrow \mathsf{SS}_t(x_i)$. Then for $j \in [n]$, compute $\{\sigma_{i,j,k}\}_{k\in[n]} \leftarrow \mathsf{SS}_t(\sigma_{i,j})$. For $k \in [n]$, encrypt the shares $\{\sigma_{i,j,k}\}_{j\in[n]}$ using the public key pk_k and broadcast all the resulting ciphertexts. Also give an NIZK proof of correct behavior.

Round 2: Let $\mathcal{I}$ be the set of parties whose round-1 proofs verify. Each party P_k then does:

1. For $j \in [n]$ do:
   (a) For $i \in \mathcal{I}$, recover $\sigma_{i,j,k}$ by decrypting the corresponding ciphertext. Then compute $\sigma'_{j,k} := \sum_{i \in \mathcal{I}} \sigma_{i,j,k}$.
   (b) Encrypt $\sigma'_{j,k}$ using the public key pk_j, and broadcast the resulting ciphertext. Also broadcast $y_{j,k} := g^{\sigma'_{j,k}}$.
   (c) Give an NIZK proof of correct behavior.

Output determination: Let $\mathcal{I}' \subseteq \mathcal{I}$ be the set of parties whose round-2 proofs verify, and let $\mathcal{I}''$ be the $t+1$ smallest indices in $\mathcal{I}'$. Each P_j then does:

1. For $k \in \mathcal{I}''$, recover $\sigma'_{j,k}$ by decrypting the corresponding ciphertext. Then set $\sigma_j := \mathsf{interpolate}(0, \mathcal{I}'', \{\sigma'_{j,k}\}_{k \in \mathcal{I}''})$.
2. For $i \in [n]$, set $y_i := \mathsf{interpolate}(0, \mathcal{I}'', \{y_{i,k}\}_{k \in \mathcal{I}''})$. Set $y := \mathsf{interpolate}(0, \mathcal{I}'', \{y_i\}_{i \in \mathcal{I}''})$.
3. Output $(y, \sigma_i, (y_1, \ldots, y_n))$.

A proof of the following is similar to the proof of Theorem 1.

Theorem 2. *Assume* $(\mathsf{Gen}, \mathsf{Enc})$ *is a perfectly correct, CPA-secure encryption scheme and identity-based simulation-sound NIZK proof systems are used. Then for* $t < n/2$*, the protocol above* t*-securely realizes* $\widehat{\mathcal{F}}^{t,n}_{\mathsf{KeyGen}}$.

5 A Two-Round Protocol in the CRS Model

We show here how $(t+1)$-party NIKE can be used to construct a 2-round, robust DKG protocol tolerating t corrupted parties. (However, the protocol has complexity linear in $\binom{n}{t}$.) We build up to this result by first discussing the case $n = 3, t = 1$, where 2-party NIKE corresponds to Diffie-Hellman key exchange. In that setting, we begin by describing a robust 3-party protocol for generating a uniform group element y ("coin tossing"), and then show how to extend it to a full-fledged DKG protocol.

For the coin-tossing protocol, the idea is that in the first round each pair of parties runs an instance of Diffie-Hellman key exchange; in the second round, each party broadcasts the key it shares with each other party (with NIZK proofs used to ensure correctness). The product of all the shared keys is the common output. Of course, a corrupted party may abort in the second round; the crucial observation that ensures robustness, however, is that such an abort by a single party P_i does not prevent computation of the key, since the remaining honest parties on their own can collectively compute any shared keys that P_i was supposed to broadcast. In more detail, the protocol works as follows:

Round 1: Each party P_i does the following: for $j \neq i$, choose $x_{i,j} \leftarrow \mathbb{Z}_q$, set $h_{i,j} := g^{x_{i,j}}$, and broadcast $h_{i,j}$.

If a party fails to broadcast some value, that value is treated as the identity element.

Round 2: Each party P_i does the following: for $j \neq i$, compute $k_{i,j} := h_{j,i}^{x_{i,j}}$; broadcast $k_{i,j}$ along with an (identity-based simulation-sound) NIZK proof $\pi_{i,j}$ that $k_{i,j}$ was computed correctly.

Output determination: For each unordered pair $\{i,j\}$, let $k_{\{i,j\}} \in \{k_{i,j}, k_{j,i}\}$ be the value for which the associated round-2 proof is valid. Output $y := k_{\{1,2\}} \cdot k_{\{1,3\}} \cdot k_{\{2,3\}}$.

We provide a brief sketch that this protocol 1-securely realizes a robust coin-tossing functionality. Specifically, we describe an ideal-world adversary $\mathcal{S}$ corresponding to any real-world adversary $\mathcal{A}$. Assume for simplicity that $\mathcal{A}$ corrupts P_1. Adversary $\mathcal{S}$ receives $y \in \mathbb{G}$ from the ideal functionality. It then simulates an execution of the protocol with $\mathcal{A}$ by running the first round of the protocol honestly, and computing $k_{\{1,2\}}, k_{\{1,3\}}$ at the end of the first round. Then $\mathcal{S}$ sets

$$k_{\{2,3\}} := y \cdot k_{\{1,2\}}^{-1} \cdot k_{\{1,3\}}^{-1}$$

and broadcasts $k_{2,3} := k_{3,2} := k_{\{2,3\}}$ along with simulated proofs of correctness in the second round. (It also broadcasts $k_{2,1}, k_{3,1}$ with honestly generated proofs of correctness.)

We can extend this idea to obtain a full-fledged DKG protocol by having the parties use the (shared) random values $k_{\{1,2\}}, k_{\{1,3\}}, k_{\{2,3\}}$ as randomness for an instance of secret sharing that they run in the second round. That is, the value $k_{\{1,2\}}$ will be used by both P_1 and P_2 to derive a secret $x_{\{1,2\}}$ and shares $\{\sigma_{\{1,2\},i}\}$; both parties will broadcast commitments $\{g^{\sigma_{\{1,2\},i}}\}$ to the shares (along with NIZK proofs of correctness) and send the shares themselves to the corresponding parties. A corrupted party can abort in the second round, but as before this does not matter since at least one party in each pair of parties is guaranteed to be honest.

Generalizing to arbitrary n. The protocol can be generalized to arbitrary n and $t < n/2$ assuming the existence of $(t+1)$-party NIKE. This consists of algorithms $(\mathsf{NIKE}_1, \mathsf{NIKE}_2)$ where:

- NIKE_1 is a randomized algorithm that outputs a pair of values $(\mathsf{st}, \mathsf{msg})$.
- NIKE_2 is a deterministic algorithm that takes as input st and values $\mathsf{msg}_1, \ldots, \mathsf{msg}_t$ and outputs a value k.

For correctness, we require that if we have $t+1$ independent invocations of NIKE_1 to obtain

$$(\mathsf{st}_1, \mathsf{msg}_1), \ldots, (\mathsf{st}_{t+1}, \mathsf{msg}_{t+1}) \leftarrow \mathsf{NIKE}_1,$$

then it holds that

$$\mathsf{NIKE}_2(\mathsf{st}_1, \{\mathsf{msg}_i\}_{i\in[t+1]\setminus\{1\}}) = \cdots = \mathsf{NIKE}_2(\mathsf{st}_{t+1}, \{\mathsf{msg}_i\}_{i\in[t+1]\setminus\{t+1\}}).$$

Security requires that $\mathsf{NIKE}_2(\mathsf{st}_1, \{\mathsf{msg}_i\}_{i\in[t+1]\setminus\{1\}})$ be indistinguishable from a uniform element chosen from the appropriate domain, even given $\{\mathsf{msg}_i\}_{i\in[t+1]}$. For our purposes, we view the key k output by NIKE_2 as a pair $k = (x, \omega) \in \mathbb{Z}_q \times \mathbb{Z}_q^t$.

The DKG protocol is described in Figure 3. In the figure, we let $\mathbb{S}_{t+1,n}$ denote the collection of all subsets of $[n]$ of size $t+1$. For notational simplicity we assume here that $\mathsf{FVSS}_t(x)$ outputs commitments to x and all n shares, instead of only outputting commitments to x and the first t shares; note that one can derive the former from the latter, anyway.

Theorem 3. *Assume a secure $(t+1)$-party NIKE and an identity-based simulation-sound NIZK proof system are used. Then for $t < n/2$, protocol $\Pi_{\mathrm{CRS}}^{t,n}$ t-securely realizes $\mathcal{F}_{\mathsf{KeyGen}}^{t,n}$.*

$\Pi^{t,n}_{\text{CRS}}$

We assume a common random string used for the required NIZK proofs.

Round 1: Each party P_i does the following: for all $S \in \mathbb{S}_{t+1,n}$ such that $i \in S$: run $(\mathsf{st}_{i,S}, \mathsf{msg}_{i,S}) \leftarrow \mathsf{NIKE}_1$ and broadcast $\mathsf{msg}_{i,S}$.

If a party fails to broadcast some message, it is treated as some canonical (valid) message.

Round 2: Each party P_i does the following for all $S \in \mathbb{S}_{t+1,n}$ such that $i \in S$:

1. Compute $(x_S, \omega_S) := \mathsf{NIKE}_2(\mathsf{st}_{i,S}, \{\mathsf{msg}_{j,S}\}_{j \in S \setminus \{i\}})$.
2. Compute $\left(\{y_{i,S,j}\}_{j=0}^n, \{\sigma_{i,S,j}\}_{j\in[n]}\right) := \mathsf{FVSS}_t(x_S; \omega_S)$, along with an NIZK proof $\pi_{i,S}$ that the values $\{y_{i,S,j}\}_{j=0}^n$ were computed correctly based on $\{\mathsf{msg}_{i,S}\}_{i\in S}$.
3. Broadcast $\{y_{i,S,j}\}_{j=0}^n$ and $\pi_{i,S}$. For $j \in [n]$, send $\sigma_{i,S,j}$ to P_j via private channel.

Output determination: Each party P_i does:

1. For each $S \in \mathbb{S}_{t+1,n}$ do:
   (a) Let $j \in S$ be such that P_j broadcasted $\{y_{j,S,k}\}_{k=0}^n$ and a valid proof $\pi_{j,S}$, and $g^{\sigma_{j,S,i}} = y_{j,S,i}$. Set $\sigma_{S,i} := \sigma_{j,S,i}$, and for $k = 0, \ldots, n$ set $y_{S,k} := y_{j,S,k}$.
2. Set $\sigma_i := \sum_{S\in\mathbb{S}_{t+1,n}} \sigma_{S,i}$, and for $k = 0, \ldots, n$ set $y_k := \prod_{S\in\mathbb{S}_{t+1,n}} y_{S,k}$.
3. Output $(y_0, \sigma_i, (y_1, \ldots, y_n))$.

Figure 3: A 2-round DKG protocol in the CRS model, parameterized by t, n.

6 Impossibility of One-Round DKG Protocols

Here we rule out the existence of one-round DKG protocols. We prove our result by showing the impossibility of one-round coin tossing for any number of parties, even when only a single party is corrupted. (It is immediate that any DKG protocol realizing $\mathcal{F}_{\mathsf{KeyGen}}$ can be used for coin tossing, with no additional rounds.) In our impossibility result we show that for any one-round coin-tossing protocol it is always possible for a corrupted party to bias the outcome of the coin. Since the attack we demonstrate does not require the corrupted party to abort, it rules out even weaker notions of DKG that do not require robustness. Our result also holds regardless of any prior setup the parties may have, and even in the random-oracle model. Specifically, we show:

Theorem 4. *There is no 1-round protocol that 1-securely realizes $\mathcal{F}^{\perp}_{\mathsf{KeyGen}}$ (cf. Appendix A).*

We remark that existing impossibility results for collective coin tossing [4], relying on analyzing the influence of boolean functions [29], can also be used to rule out one-round coin-tossing protocols. However, a direct application of those results would only show that an *all-powerful* adversary can bias the outcome; our result holds even for *computationally* bounded adversaries. Moreover, we obtain quantitatively stronger bounds on the bias a corrupted party can achieve than what follows from those results.

Consider the following natural strategy by a corrupted party P_i to bias the outcome of a coin-tossing protocol toward a particular bit b: based on the messages of the other parties and local randomness r_i, compute the output that would result from running the protocol honestly using r_i. If the result is b, then run the protocol honestly using r_i; otherwise, sample fresh randomness r'_i and run the protocol honestly using r'_i. (Note that we are here using the fact that the adversary is *rushing.*) If we let $f(r_1,\dots,r_n)$ denote[9] the output[10] when parties run the protocol honestly using the randomness indicated, then the probability that this strategy results in output b is exactly

$$\Pr\left[f(r_1,\dots,r_n)=b\right]+\Pr\left[f(r_1,\dots,r_n)=\bar{b}\bigwedge f(r_1,\dots,r_{i-1},r'_i,r_{i+1},\dots,r_n)=b\right].$$

Assuming $\Pr[f(r_1,\dots,r_n)=b]=\frac{1}{2}$ for simplicity (it is easy to see this is not essential for the proof), this means that P_i can bias the outcome toward some bit if

$$\Pr_{r_1,\dots,r_n,r'_i}\left[f(r_1,\dots,r_{i-1},r_i,r_{i+1},\dots,r_n)\neq f(r_1,\dots,r_{i-1},r'_i,r_{i+1},\dots,r_n)\right] \tag{1}$$

is large. We show this must be the case for some i.

If $\Pr[f(r_1,\dots,r_n)=b]=\frac{1}{2}$, then $\Pr\left[f(r_1,\dots,r_n)\neq f(r'_1,\dots,r'_n)\right]=\frac{1}{2}$ as well. Note that $f(r_1,\dots,r_n)\neq f(r'_1,\dots,r'_n)$ implies that $f(r'_1,\dots,r'_{i-1},r_i,\dots,r_n)\neq f(r'_1,\dots,r'_{i-1},r'_i,r_{i+1},\dots,r_n)$ for some $i\in[n]$. Therefore,

$$\begin{aligned}
\frac{1}{2} &= \Pr\left[f(r_1,\dots,r_n)\neq f(r'_1,\dots,r'_n)\right]\\
&\leq \Pr\left[\begin{array}{c} f(r_1,\dots,r_n)\neq f(r'_1,r_2,\dots,r_n)\\ \bigvee f(r'_1,r_2,\dots,r_n)\neq f(r'_1,r'_2,r_3,\dots,r_n)\\ \vdots \\ \bigvee f(r'_1,\dots,r'_{n-1},r_n)\neq f(r'_1,\dots,r'_n)\end{array}\right]\\
&\leq \Pr[f(r_1,\dots,r_n)\neq f(r'_1,r_2,\dots,r_n)]\\
&\quad +\Pr[f(r'_1,r_2,\dots,r_n)\neq f(r'_1,r'_2,r_3,\dots,r_n)]\\
&\quad \vdots\\
&\quad +\Pr[f(r'_1,\dots,r'_{n-1},r_n)\neq f(r'_1,\dots,r'_n)],
\end{aligned}$$

which implies that, for some $i\in[n]$,

$$\Pr\left[f(r'_1,\dots,r'_{i-1},r_i,r_{i+1},\dots,r_n)\neq f(r'_1,\dots,r'_{i-1},r'_i,r_{i+1},\dots,r_n)\right]\geq\frac{1}{2n}.$$

But this exactly gives a lower bound on (1).

References

[1] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation, 2022. Available at `https://eprint.iacr.org/2022/1759`.

[2] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation. In *40th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 363–373. ACM Press, 2021.

[9] Formally, f depends on any prior (honestly generated) setup the parties may have, including any oracles to which they have access.

[10] For simplicity, we assume the protocol has perfect correctness so all parties always agree on the output. Our proof can be easily modified to handle protocols with small probability of disagreement among the parties.

[3] R. Bacho and J. Loss. On the adaptive security of the threshold BLS signature scheme. In *29th ACM Conf. on Computer and Communications Security (CCS)*, pages 193–207. ACM Press, 2022.

[4] M. Ben-Or and N. Linial. Collective coin flipping. *Adv. Computing Research*, 5:91–115, 1989.

[5] D. Boneh and V. Shoup. A graduate course in applied cryptography, v. 0.6, 2023. Available at `http://toc.cryptobook.us`.

[6] L. Brandão and R. Peralta. NIST first call for multi-party threshold schemes. Technical report, National Institute of Standards and Technology, 2023. Internal Report (IR 8214C ipd). Available at `https://doi.org/10.6028/NIST.IR.8214C.ipd`.

[7] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *27th ACM Conf. on Computer and Communications Security (CCS)*, pages 1769–1787. ACM Press, 2020.

[8] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In M. J. Wiener, editor, *Advances in Cryptology—Crypto '99*, volume 1666 of *LNCS*, pages 98–115. Springer, 1999.

[9] R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology—Crypto 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.

[10] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369. ACM Press, 1986.

[11] D. Connolly, C. Komlo, I. Goldberg, and C. Wood. Two-round threshold Schnorr signatures with FROST, 2023. IRTF draft-irtf-cfrg-frost-12. Available at `https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost`.

[12] E. Crites, C. Komlo, and M. Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures, 2021. Available at `https://eprint.iacr.org/2021/1375`.

[13] I. Damgård, B. Magri, D. Ravi, L. Siniscalchi, and S. Yakoubov. Broadcast-optimal two round MPC with an honest majority. In *Advances in Cryptology—Crypto 2021, Part II*, volume 12826 of *LNCS*, pages 155–184. Springer, 2021.

[14] S. Das, T. Yurek, Z. Xiang, A. K. Miller, L. Kokoris-Kogias, and L. Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE, 2022.

[15] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, 2001.

[16] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 427–437. IEEE, 1987.

[17] R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *25th ACM Conf. on Computer and Communications Security (CCS)*, pages 1179–1194. ACM Press, 2018.

[18] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. On 2-round secure multiparty computation. In *Advances in Cryptology—Crypto 2002*, volume 2442 of *LNCS*, pages 178–193. Springer, 2002.

[19] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure applications of Pedersen's distributed key generation protocol. In *Cryptographers' Track—RSA 2003*, volume 2612 of *LNCS*, pages 373–390. Springer, 2003.

[20] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007. Preliminary version in Eurocrypt '99.

[21] C. Gentry, S. Halevi, and V. Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Advances in Cryptology—Eurocrypt 2022, Part I*, volume 13275 of *LNCS*, pages 458–487. Springer, 2022. Available at `https://eprint.iacr.org/2021/1397`.

[22] A. Goel, A. Jain, M. Prabhakaran, and R. Raghunath. On communication models and best-achievable security in two-round MPC. In *19th Theory of Cryptography Conference—TCC 2021, Part II*, volume 13043 of *LNCS*, pages 97–128. Springer, 2021.

[23] S. D. Gordon, F.-H. Liu, and E. Shi. Constant-round MPC with fairness and guarantee of output delivery. In *Advances in Cryptology—Crypto 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, 2015.

[24] J. Groth. Non-interactive distributed key generation and key resharing, 2021. Available at `https://eprint.iacr.org/2021/339`.

[25] J. Groth and V. Shoup. Design and analysis of a distributed ECDSA signing service, 2022. Available at `https://eprint.iacr.org/2022/506`.

[26] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Aggregatable distributed key generation. In *Advances in Cryptology—Eurocrypt 2021, Part I*, volume 12696 of *LNCS*, pages 147–176. Springer, 2021.

[27] Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, 2014.

[28] M. Jhanwar, A. Venkateswarlu, and R. Safavi-Naini. Paillier-based publicly verifiable (non-interactive) secret sharing. *Designs, codes and Cryptography*, 73(2):529–546, 2014.

[29] J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions. In *29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–80. IEEE, 1988.

[30] A. Kate, Y. Huang, and I. Goldberg. Distributed key generation in the wild, 2012. Available at `https://eprint.iacr.org/2012/377`.

[31] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, 3rd edition.* Chapman & Hall/CRC Press, 2020.

[32] J. Katz, R. Ostrovsky, and M. O. Rabin. Identity-based zero knowledge. In *4th Intl. Conf. on Security in Communication Networks — SCN 2004*, volume 3352 of *LNCS*, pages 180–192. Springer, 2004.

[33] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *27th ACM Conf. on Computer and Communications Security (CCS)*, pages 1751–1767. ACM Press, 2020.

[34] C. Komlo, I. Goldberg, and D. Stebila. A formal treatment of distributed key generation, and new constructions, 2023. Available at `https://eprint.iacr.org/2023/292`.

[35] V. Koppula, B. Waters, and M. Zhandry. Adaptive multiparty NIKE. In *20th Theory of Cryptography Conference—TCC 2022, Part II*, volume 13748 of *LNCS*, pages 244–273. Springer, 2022.

[36] Y. Lindell. Simple three-round multiparty Schnorr signing with full simulatability, 2022. Available at `https://eprint.iacr.org/2022/374`.

[37] Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *25th ACM Conf. on Computer and Communications Security (CCS)*, pages 1837–1854. ACM Press, 2018.

[38] A. Patra and D. Ravi. On the exact round complexity of secure three-party computation. In *Advances in Cryptology—Crypto 2018, Part II*, volume 10992 of *LNCS*, pages 425–458. Springer, 2018.

[39] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—Crypto '91*, volume 576 of *LNCS*, pages 129–140. Springer, 1992.

[40] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In *29th ACM Conf. on Computer and Communications Security (CCS)*, pages 2551–2564. ACM Press, 2022.

[41] N. Shrestha, A. Bhat, A. Kate, and K. Nayak. Synchronous distributed key generation without broadcasts, 2021. Available at `https://eprint.iacr.org/2021/1635`.

[42] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology—Eurocrypt '96*, volume 1070 of *LNCS*, pages 190–199. Springer, 1996.

A Alternate Ideal Functionalities for Key Generation

As discussed in Section 3, one can define different notions of security for distributed key generation by specifying different ideal functionalities. We explore several such possibilities here.

For completeness, we show in Figure 4 an alternate ideal functionality $\widehat{\mathcal{F}}_{\mathsf{KeyGen}}$ for fully secure key generation. This functionality is stronger than $\mathcal{F}_{\mathsf{KeyGen}}$ in that it does not give the adversary the ability to choose its own shares.

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

1. Choose $x \leftarrow \mathbb{Z}_q$ and compute $y := g^x$.
2. Compute $\{\sigma_i\}_{i=0}^n \leftarrow \mathsf{SS}_t(x)$. For $i \in [n]$, set $y_i := g^{\sigma_i}$; let $Y = (y_1, \ldots, y_n)$.
3. For $i \in [n]$, send (y, σ_i, Y) to P_i. Also send (y, Y) to the adversary.

Figure 4: Alternate ideal functionality for fully secure key generation, parameterized by t, n.

Although our main interest in this paper is robust key generation, it is useful to consider non-robust notions of security for DKG protocols. For one thing, it is impossible to achieve robustness when $t \geq n/2$; even when $t < n/2$, it may be possible to achieve weaker notions of security via more-efficient protocols. Moreover, some of the weaker definitions we discuss below correspond to what is achieved by protocols in prior work. For simplicity, we only define functionalities in which the adversary is allowed to choose its own shares; of course, each of the functionalities we consider could also be defined in a way that prevents the attacker from doing so.

The first variant we consider, denoted $\mathcal{F}^{\perp}_{\mathsf{KeyGen}}$, corresponds to a "secure-with-abort" version of $\mathcal{F}_{\mathsf{KeyGen}}$ where an adversary can abort the protocol and prevent the honest parties from receiving output. This functionality allows the adversary to make its decision about whether to abort based on the public key returned by the functionality, and hence allows the attacker to bias the public key. (That is, the attacker has the ability to bias the distribution of the public key conditioned on a public key being computed by the honest parties.) We also define $\mathcal{F}^{\perp,\mathsf{fair}}_{\mathsf{KeyGen}}$, a version of the key-generation functionality that does not have guaranteed output delivery, but forces the adversary to determine whether to abort independent of the value of the key. See Figure 5 for details.

Either of the above definitions could be strengthened to also incorporate the notion of *identifiable abort* [27]. It is also possible to define variants of the key-generation functionality that ensure robustness, but are weaker than $\mathcal{F}_{\mathsf{KeyGen}}$ in that they allow the attacker to bias the public key. We leave a full consideration of such variants to future work.

$\mathcal{F}^{\perp}_{\mathsf{KeyGen}}$

Let $\mathcal{C}'$ be an arbitrary set of size t with $\mathcal{C} \subseteq \mathcal{C}' \subset [n]$.

1. Receive $\{\sigma_i\}_{i\in\mathcal{C}}$ from the adversary.
2. Choose $x \leftarrow \mathbb{Z}_q$ and set $y := g^x$. Choose uniform $\sigma_i \in \mathbb{Z}_q$ for $i \in \mathcal{C}' \setminus \mathcal{C}$.
3. Let f be the polynomial of degree at most t such that $f(0) = x$ and $f(i) = \sigma_i$ for $i \in \mathcal{C}'$. Set $\sigma_i := f(i)$ for $i \in [n] \setminus \mathcal{C}'$.
4. For $i \in [n]$ set $y_i := g^{\sigma_i}$. Let $Y = (y_1, \ldots, y_n)$.
5. Send (y, Y) to the adversary. The adversary responds with either abort or $\mathsf{continue}$. If the adversary sent with abort and $|\mathcal{C}| \geq 1$ then send $\perp$ to all honest parties and stop. Otherwise, for $i \in [n]$ send (y, σ_i, Y) to P_i.

$\mathcal{F}^{\perp,\mathsf{fair}}_{\mathsf{KeyGen}}$

Let $\mathcal{C}'$ be an arbitrary set of size t with $\mathcal{C} \subseteq \mathcal{C}' \subset [n]$.

1. The adversary sends either abort or $\{\sigma_i\}_{i\in\mathcal{C}}$. If the adversary sent abort and $|\mathcal{C}| \geq 1$ then send $\perp$ to all honest parties and stop. Otherwise, continue below.
2. Choose $x \leftarrow \mathbb{Z}_q$ and set $y := g^x$. Choose uniform $\sigma_i \in \mathbb{Z}_q$ for $i \in \mathcal{C}' \setminus \mathcal{C}$.
3. Let f be the polynomial of degree at most t such that $f(0) = x$ and $f(i) = \sigma_i$ for $i \in \mathcal{C}'$. Set $\sigma_i := f(i)$ for $i \in [n] \setminus \mathcal{C}'$.
4. For $i \in [n]$, set $y_i := g^{\sigma_i}$. Let $Y = (y_1, \ldots, y_n)$.
5. For $i \in [n]$, send (y, σ_i, Y) to P_i. Send (y, Y) to the adversary.

Figure 5: Non-robust key-generation functionalities, parameterized by t, n.