Round-Optimal Fully Secure Distributed Key Generation

Jonathan Katz*

Dfns
jkatz@dfns.co, jkatz2@gmail.com

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 not fully secure: they either allow a single corrupted party to *bias* the key, or are not *robust* and allow a single malicious party to prevent successful generation of a key.

We explore the round complexity of fully secure DKG in the honest-majority setting where it is feasible. We show the impossibility of one-round, unbiased DKG protocols (even satisfying weaker notions of security), regardless of any prior setup. On the positive side, we show various round-optimal protocols for fully secure 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 cryptography has developed threshold protocols for the ECDSA, Schnorr, and BLS signature schemes, as well as protocols for distributed key generation (DKG) [45, 8, 18, 22, 21, 34, 43, 38, 2, 29, 7, 13, 26, 46, 3, 1, 15, 28, 41, 39, 35] 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 their standardization [11, 6].

DKG protocols have been studied in both the synchronous [45, 8, 18, 22, 21, 29, 7, 13, 26, 46, 3, 41, 39, 35] and asynchronous [38, 2, 1, 15, 28] 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 model in this paper.

* This work was not part of my University of Maryland duties.

We focus here on the round complexity of *fully secure* 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 fixed generator of a cyclic group $\mathbb{G}$) such that the parties hold $(t+1)$-out-of-n secret shares $\{\sigma_i\}$ of the private exponent x. We also ensure that parties can compute public "commitments" $\{g^{\sigma_i}\}$ to each others' shares, as is often required by threshold protocols. "Full security" for a DKG protocol is defined in a simulation-based framework via an appropriate ideal functionality (see below and Section 3). A fully secure protocol must, in particular, prevent the adversary from being able to *bias* the generated public key y. It also requires *robustness* (aka *guaranteed output delivery*); namely, the honest parties must always produce correct output, even in the presence of malicious behavior. We assume an honest majority since fully secure DKG protocols are impossible[1] otherwise.

Perhaps surprisingly, there are few explicit constructions of fully secure DKG protocols in the literature; the only examples we are aware of are the protocols of Gennaro et al. [22] and Boneh and Shoup [5, Section 22.4.2]. Most existing DKG protocols—even in the honest-majority setting—allow bias (e.g., [18, 35]) and/or are not robust (e.g., [26, 39]). Indeed, it is challenging to both prevent bias and achieve robustness in few rounds. The most round-efficient explicit construction of a fully secure DKG protocol appears to be the 6-round protocol by Gennaro et al. [22]. One could apply known results [25, 24, 14] 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.

1.1 Our Results

As noted above, we work in a simulation-based framework and define full security for DKG protocols via a corresponding ideal functionality that (in particular) prevents bias and ensures guaranteed output delivery. We also show in Appendix A several other ideal functionalities for DKG one might consider. Although such simulation-based definitions seem to us the most natural way to define security, several recent works have instead given (different) game-based definitions that are quite complex, somewhat difficult to interpret or compare to one another, and hard to use in a modular fashion when designing threshold protocols relying on them. We believe our definitional treatment, though perhaps obvious to those familiar with MPC, will be useful for future work.

We then consider the round complexity of fully secure DKG in the honest-majority setting. On the negative side, we show in Section 4 the impossibility of one-round unbiased DKG, where "unbiased" roughly means that the public

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

Protocol	**Setup**	**Rounds**	**Assumptions**
Π_1	CRS + PKI	2	NIZK + PKE
Π_{CRS}	CRS	2	NIZK + MP-NIKE
Π_{ROM}	ROM + 1-round preprocessing	2	—
$\Pi_{\text{1-round}}$	CRS + 2-round preprocessing	1	NIZK + OWF

Table 1. Summary of protocols in this paper. Guide to acronyms not defined in the introduction: NIZK = non-interactive zero knowledge proofs, PKE = public-key encryption, OWF = one-way functions. Note that the ROM implies a CRS, which is why we list it as a form of setup.

key is close to uniformly distributed regardless of adversarial behavior. Our impossibility result (1) holds even for non-robust DKG protocols, and 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 bias of any one-round DKG protocol. To the best of our knowledge, the impossibility result does not follow from prior work; in particular, existing lower bounds on the round complexity of MPC with guaranteed output delivery [20, 25, 44, 24, 14] take advantage of the fact that honest parties have input, and do not seem to extend to the case of no-input functionalities like DKG.

On the positive side, we show several constructions of round-optimal fully secure DKG protocols that offer tradeoffs in terms of their computational efficiency, necessary setup, and cryptographic assumptions (cf. Figure 1):

1. In Section 5, we show a framework for constructing 2-round, fully secure DKG protocols assuming a PKI and a CRS. Although the same round complexity could be obtained using prior work on generic MPC, our framework uses weaker cryptographic assumptions and can lead to more-efficient constructions. In particular, we propose a reasonably efficient instantiation of our framework based on the El Gamal and Paillier encryption schemes in the random-oracle model.
2. In Section 6, we show a 2-round, fully secure DKG protocol based on a CRS alone. 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 $n > 6$, it relies on multiparty non-interactive key exchange (MP-NIKE), something currently known to exist only based on strong assumptions.[2] (We refer to Koppula et al. [40] for a survey of known results.) It also has $O(\binom{n}{t})$ complexity, 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 *impossibility* of 2-round fully secure DKG in the CRS model.

[2] For $n = 3, 4$ the standard decisional Diffie-Hellman assumption suffices, and when $n = 5, 6$ the decisional bilinear Diffie-Hellman assumption can be used.

3. We show in Section 7 a fully secure protocol in the random-oracle model (ROM) with the following property: 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.[3] A simple modification of the protocol can also be proven adaptively secure. This approach relies on combinatorial techniques from pseudorandom secret sharing [12], and is extremely efficient for small values of n typically used in practice even though its asymptotic complexity is $O(\binom{n}{t})$. *We thus believe this protocol is a competitive choice for many real-world applications of DKG.*
4. Finally, and somewhat surprisingly given our impossibility result, we show in Section 8 a fully secure DKG protocol that requires only a *single* round. This protocol does not violate the impossibility result claimed earlier since it is *not* unbiased. This may itself seem surprising, as the ideal functionality used to define fully secure DKG *is* unbiased (since it chooses a uniform public key). There is no contradiction since this protocol generates a *pseudorandom* (rather than a truly random) public key, and is thus not unbiased.

   This protocol relies on a CRS. In addition, similar to the previous protocol, it requires[4] two rounds of preprocessing after which the parties can generate an unbounded number of keys; it also has complexity $O(\binom{n}{t})$.

In all cases our simulation-based proofs of security do not use rewinding; hence our protocols are also universally composable.

We leave as interesting open questions whether there is a two-round, fully secure DKG protocol in the plain model, or whether there is an efficient such protocol in the ROM without preprocessing. As far as we know the first question is open even for sub-optimal corruption threshold $t < n/3$.

2 Background

2.1 Preliminaries

Notation. We let $\mathbb{G}$ be a cyclic group of prime order q and let $g \in \mathbb{G}$ be a fixed generator. We let $\mathbb{Z}_q$ be the field $\{0, \ldots, q-1\}$ modulo q, 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 $P_1, \ldots, P_n$, at most t of whom are corrupted. We work in the standard synchronous communication

[3] Alternately, one can view it as a 3-round protocol in the ROM or a 2-round protocol in the ROM given certain correlated randomness. In practice, however, we believe it would be run in the manner described.

[4] As before, it could alternately be viewed as a 3-round protocol in the CRS model or, if one prefers, as a 1-round protocol given certain correlated randomness.

model where parties are connected by pairwise private and authenticated channels in addition to a public broadcast channel.[5] 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.

Defining round complexity. The round complexity of a protocol execution in the synchronous model is straightforward to define. However, some protocols may 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 *optimistic* round complexity (when all parties are honest) and *worst-case* round complexity (which holds for arbitrary adversarial behavior). All the protocols in this paper 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. I.e., 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 such 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 $x = \mathsf{interpolate}(0, S, \{\sigma_i\}_{i \in S})$.

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 $g^{f(k)} = \prod_{i \in S} \left(g^{f(i)}\right)^{\lambda_{i,k}^S}$. Overloading notation slightly, we also denote this by $g^{f(k)} = \mathsf{interpolate}(k, S, \{g^{f(i)}\}_{i \in S})$.

Feldman verifiable secret sharing. Feldman's variant of Shamir secret sharing [17] allows parties to verify that they have received shares consistent with a

[5] 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. To the best of our knowledge, all work on synchronous DKG (with the exception of [46]) assumes broadcast.

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 broadcasts $y_0 := g^{\sigma_0}, \ldots, y_t := g^{\sigma_t}$ (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 $g^{\sigma_i} \overset{?}{=} \mathsf{interpolate}(i, S, \{y_j\}_{j \in S})$. 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 [36] 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 [16, 37]. 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 any efficient adversary $\mathcal{A}$, the following is negligible
$$\Big|\Pr[\mathsf{crs} \leftarrow \mathsf{GenCRS} : \mathcal{A}^{\mathcal{P}^*(\mathsf{crs},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1] \\ - \Pr[(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_1 : \mathcal{A}^{\mathsf{Sim}_2^*(\mathsf{td},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1]\Big|,$$
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 that even if an adversary is given multiple simulated proofs, it cannot generate a new, valid proof for a false statement. This is formalized by requiring that if the adversary outputs a (new) valid proof for a 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 $(x, w) \notin R$ or $i \notin \mathcal{H}$. Otherwise, it returns $\mathsf{Sim}_2(\mathsf{td}, i, x)$.

(b) The second oracle takes input (i, x, π) and returns $\bot$ 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 if it holds that $(x, w) \in R$ (and returns $\bot$ otherwise).

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

3 Defining Secure 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 (UC) framework. (As noted earlier, the security proofs for all our protocols use straight-line simulation, so apply also in the UC setting.) We assume static corruptions.

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 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 party terminates, it outputs the value 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), no-input 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. $\mathcal{S}$ controls what corrupted parties send to $\mathcal{F}$, and observes all values that $\mathcal{F}$ sends to corrupted parties. (Note $\mathcal{F}$ is aware of the set $\mathcal{C}$ of corrupted parties.) $\mathcal{S}$ may also send messages directly to, and receive messages directly from, $\mathcal{F}$. (The effect of those messages depends on $\mathcal{F}$.)
3. An honest party outputs the value sent to it by $\mathcal{F}$.
4. The adversary $\mathcal{S}$ outputs 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

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

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. (If some σ_i is not sent, set it to 0.)
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.

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

distinguisher D can distinguish $\textsc{real}_{\Pi,\mathcal{A}}$ and $\textsc{ideal}_{\mathcal{F},\mathcal{S}}$. In the concrete setting we adopt here, one can quantify security by bounding the running times of $\mathcal{A}, \mathcal{S}$, and D as well as the acceptable distinguishing advantage 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) $\mathcal{A}$ 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 protocols 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 can be used by parties to prove correctness of their actions in a subsequent protocol using the generated key), and this is incorporated in 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 functionality. In the synchronous setting we consider, some form of synchrony is also (implicitly) assumed in the ideal world as well. This means, in particular, that the ideal functionality can identify when the adversary has failed to send some share in step 1, and can proceed accordingly.

$\mathcal{F}^{t,n}_{\mathsf{KeyGen}}$ 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 an adversary who eavesdrops on an execution of Π (but corrupts no parties) may learn the public key y and the parties' commitments Y. This is acceptable, as those values are considered public.

$\mathcal{F}^{t,n}_{\mathsf{KeyGen}}$ assumes a Shamir secret sharing of the private key x for concreteness, but could be modified in the natural way for other secret-sharing schemes, e.g.,

n-out-of-n additive sharing of x when $t = n-1$. 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 (cf. functionality $\widehat{\mathcal{F}}_{\mathsf{KeyGen}}$ in Appendix A), but we are not aware of any (natural[6]) application of key generation where the distinction is important, 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-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 (i.e., setup) is used across multiple executions of a protocol, 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 here, but remark that it is not hard to verify that our protocols satisfy this requirement.

4 Impossibility of One-Round (Unbiased) DKG Protocols

Here we rule out the existence of one-round *unbiased* DKG protocols, where unbiased means that the public key computed in an honest execution of the protocol is close to uniform in $\mathbb{G}$. (We define this more formally below.) We prove this by showing the impossibility of one-round (unbiased) coin tossing, even if only a single party is corrupted. Our result shows that for any one-round coin-tossing protocol it is always possible for some corrupted party to bias the coin. Since the attack does not require the corrupted party to abort, it rules out even weaker notions of coin tossing (and hence DKG) that do not require robustness. Our result also holds regardless of any prior setup the parties have, and holds even in idealized models (e.g., the random-oracle model).

We remark that existing impossibility results for collective coin tossing [4], relying on analyzing the influence of boolean functions [33], could potentially also be used to rule out one-round unbiased DKG 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.

We now proceed with the proof. Given a one-round, n-party DKG protocol Π, we can construct a coin-tossing protocol by having the parties run Π and then output 0 (resp., 1) if the public key lies in $\mathbb{G}_0$ (resp., $\mathbb{G}_1$), where $\mathbb{G}_0, \mathbb{G}_1$ is some partition of $\mathbb{G}$. Let $f(r_1, \ldots, r_n)$ denote[7] the output of the protocol when parties

[6] It is possible to show contrived scenarios where the difference matters.

[7] We leave the dependence of f on the partition implicit. Note that f depends on any prior setup the parties have, including any oracles to which they have access.

run the protocol honestly, each using the randomness indicated. We say Π is *unbiased* if there is a partition of $\mathbb{G}$ such that $\Pr[f(r_1, \ldots, r_n) = 1] \approx 1/2$. For simplicity in what follows, we assume $\Pr[f(r_1, \ldots, r_n) = 1] = 1/2$.

Consider the following strategy by a corrupted party P_i to bias the outcome of the 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. (We are here using the fact that the adversary is *rushing*.) The probability that this strategy results in output b is exactly

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

Since $\Pr[f(r_1, \ldots, r_n) = b] = \frac{1}{2}$, this means P_i can bias the outcome toward some bit if

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

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

$\Pr[f(r_1, \ldots, r_n) = b] = \frac{1}{2}$ implies $\Pr\left[f(r_1, \ldots, r_n) \neq f(r'_1, \ldots, r'_n)\right] = \frac{1}{2}$ as well. Also, note that $f(r_1, \ldots, r_n) \neq f(r'_1, \ldots, r'_n)$ implies

$$\exists i: \; f(r'_1, \ldots, r'_{i-1}, r_i, \ldots, r_n) \neq f(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n).$$

Therefore,

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

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

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

But this exactly shows that (1) is noticeable.

5 Two-Round Protocols in the PKI+CRS Model

In this section we show fully secure, 2-round DKG protocols assuming a PKI and a CRS. 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 5.3 we show how to realize a stronger DKG functionality, still using only two rounds.

5.1 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 $j \in [n]$, broadcast $y_{i,j} := g^{\sigma_{i,j}}$ and send $\sigma_{i,j}$ to P_j over a private channel.

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 party P_j. Letting $S = [t+1]$, parties also compute the public key as $\mathsf{interpolate}(0, S, \{y_j\}_{j\in S})$.

The above description assumes semi-honest behavior. While parties can verify that a party P_i broadcasted consistent 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 another party who sent it an incorrect share, and a second round for honest parties to respond to complaints.[8] Protocols relying on complaints seem to inherently require at least three rounds.

A natural idea is to have parties use NIZK proofs to publicly prove correct behavior. However, such proofs will be useless 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 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:

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.

[8] 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.

Parties compute the public key, their shares, and commitments to other parties' shares as before, excluding the contributions from parties whose NZIK proofs fail to verify. (We omit details.) This exactly corresponds to having each party act as a dealer in a *publicly verifiable secret-sharing scheme* (PVSS) [48] (see [23] for a recent survey). Using PVSS (or something similar) in the context of distributed key generation has been proposed in several other works [18, 5, 26, 35]; none of those achieve fully secure DKG with the exception of Boneh and Shoup [5, Section 22.4.2], who achieve it using many more rounds (see below).

NIZK proofs force parties to either behave correctly or (effectively) abort. But they are not enough to make the protocol fully secure! Indeed, in the protocol sketched above 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 (where all parties act as a sender in a broadcast protocol, but are forced to choose their messages independently), 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 still 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 then *broadcast those values along with an NIZK proof that they 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. The adversary is unable to bias the key by aborting in the first round because it is unable to compute the key until the second round. On the other hand, aborts in the second round cannot introduce bias since the public key is *fixed* 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. Simulation is possible due to the NIZK proofs in the second round, which allow the simulator to "force" the final key to any desired value.

We formalize the resulting protocol in Figure 2. The protocol relies on zero-knowledge proofs for the following NP relations:

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

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

The protocol also relies on public-key encryption with perfect correctness with overwhelming probability for honestly generated keys.

$\Pi_1^{t,n}$

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

Round 1: Each party P_i does the following:
1. Choose uniform $x_i \in \mathbb{Z}_q$ and compute $\{\sigma_{i,j}\}_{j\in[n]} \leftarrow \mathsf{SS}_t(x_i)$. For $j \in [n]$ 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 $\pi_i \leftarrow \mathcal{P}(\mathsf{crs}, i, \{(\mathsf{pk}_j, c_{i,j})\}_{j\in[n]}, \{(\sigma_{i,j}, \omega_{i,j})\}_{j\in[n]})$.
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: 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 $t+1$ lowest indices in $\mathcal{I}'$. 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 $(y, \sigma_i, (y_1, \dots, y_n))$.

Fig. 2. A 2-round DKG protocol in the PKI+CRS model, parameterized by t, n. Relations $R_L, R_{L'}$ 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 $R_L, R_{L'}$. 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 $\mathcal{A}$:

Setup: $\mathcal{S}$ runs $\mathcal{A}$ to obtain a set $\mathcal{C}$ of corrupted parties with $|\mathcal{C}| \leq t$. 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 $\leq t$ with $f_j(i) = \sigma_{j,i}$ for $i \in \mathcal{H}$. (If no such f_j 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: 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[9] $\mathcal{C}$, $(y, \{\sigma_i\}_{i\in\mathcal{H}}, (y_1, \ldots, y_n))$, and the output of $\mathcal{A}$.

Experiment Expt_1. In this experiment we modify Expt_0 as follows: during setup, we now generate $\mathsf{crs}, \mathsf{crs}'$ (along with $\mathsf{td}, \mathsf{td}'$) 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.

[9] 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 experiment those values will be identical.

Indistinguishability of this experiment and Expt_0 follows immediately from the zero-knowledge property 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 $|\mathcal{H}| > t+1$.) Then, 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**.** Now 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 Expt_2 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 still 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 $i \in \mathcal{I}$ a polynomial f_i of degree at most t is defined; 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 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, if 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 $\textsc{ideal}_{\mathcal{F}^{t,n}_{\mathsf{KeyGen}},\mathcal{S}}$. $\square$

5.2 An Instantiation using Paillier Encryption

We briefly sketch a protocol based on the Paillier (additively homomorphic) encryption scheme that can be viewed as inspired by Π_1. (Several prior works [18, 31, 42, 19, 31] show how to construct PVSS from Paillier encryption, and our construction is similar in that regard.) Each party publishes a Paillier public key; correctness of public keys can be demonstrated with existing NIZK proofs if desired [7]. The CRS also 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

$$(\{y_{i,j}\}_{j=0}^{t}, \{\sigma_{i,j}\}_{j\in[n]}) \leftarrow \mathsf{FVSS}_t(x_i).$$

It then computes $C_{i,j} \leftarrow \mathsf{Enc}_h(y_{i,j})$ for $j \in S$, and $c_{i,j} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j}(\sigma_{i,j})$ 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 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 with an NIZK proof (cf. [7]) that the discrete logarithm of y_i is equal to the value encrypted by $c_i^* \overset{\text{def}}{=} \prod_{j\in\mathcal{I}} c_{j,i}$.

Output determination: Parties compute output as in Π_1.

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

5.3 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 their contributions (which are added together to give the private key), the parties now generate *shares of shares* of their contributions. Corrupted parties do not learn their shares of the private key until the second round, by which time they are already committed to the shares they generated and distributed in the first round. 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 second-level sharing $\{\sigma_{i,j,k}\}_{k\in[n]} \leftarrow \mathsf{SS}_t(\sigma_{i,j})$. 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 pk_j, and broadcast the resulting ciphertext along with $y_{j,k} := g^{\sigma'_{j,k}}$. Also 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 $t+1$ lowest 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 $y_i := \mathsf{interpolate}(0, \mathcal{I}'', \{y_{i,k}\}_{k\in\mathcal{I}''})$.
3. Set $y := \mathsf{interpolate}(0, \mathcal{I}'', \{y_i\}_{i\in\mathcal{I}''})$.
4. 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* $\widehat{\mathcal{F}}^{t,n}_{\mathsf{KeyGen}}$.

6 A Two-Round Protocol in the CRS Model

We show here a 2-round robust DKG protocol based on a CRS alone. The protocol requires $(t+1)$-party NIKE to tolerate t corrupted parties. 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.

A challenge that arises in the context of robust coin tossing, as in the case of robust DKG, is the need to prevent the adversary from biasing the outcome. To obtain a 2-round robust protocol, we need to ensure (roughly) that the outcome is determined at the end of the first round regardless of what the corrupted parties do in the second round. At the same time, it must not be possible for the corrupted parties to learn the outcome at the end of the first round, since otherwise they could bias the result then. To prevent such bias in the case of 3-party coin tossing, we use the following idea: 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 party P_i does not prevent computation of the key, since the remaining honest parties *on their own* can collectively compute the shared keys 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 a value, that value is treated as the identity element.)

Round 2: Each party P_i does the following: for $j \neq i$, compute $k_{i,j} := h_{j,i}^{x_{i,j}}$; then 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 the natural robust coin-tossing functionality. To do so, we describe an ideal-world adversary $\mathcal{S}$ corresponding to any real-world adversary $\mathcal{A}$, assuming for simplicity that $\mathcal{A}$ corrupts P_1. Adversary $\mathcal{S}$ receives $y \in \mathbb{G}$ from the ideal functionality. It simulates an execution of the protocol with $\mathcal{A}$ by running the first round of the protocol honestly, and computing $k_{\{1,2\}}, k_{\{1,3\}}$. Then $\mathcal{S}$ sets

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

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

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 to the corresponding parties via private channel. 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 $\mathsf{msg}_1, \ldots, \mathsf{msg}_t$ 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 in 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 $k = (x, \omega) \in \mathbb{Z}_q \times \mathbb{Z}_q^t$.

Note that Diffie-Hellman key exchange (based on the decisional Diffie-Hellman assumption) is a 2-party NIKE, while Joux's protocol [32] (based on the decisional bilinear Diffie-Hellman assumption) gives a 3-party NIKE.

Let $\mathbb{S}_{t+1,n}$ denote the collection of all subsets of $[n]$ of size $t+1$. We can extend the earlier idea to the general case by running $(t+1)$-party key exchange among all sets in $\mathbb{S}_{t+1,n}$; see Figure 3. (For notational simplicity we assume $\mathsf{FVSS}_t(x)$ outputs commitments to x and all n shares, instead of only commitments to x and the first t shares; one can derive the former from the latter, anyway.)

$\Pi_{\text{CRS}}^{t,n}$

We assume a CRS 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 s.t. P_j broadcasted 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 $y_{S,k} := y_{j,S,k}$ for $k = 0, \ldots, n$.
2. Set $\sigma_i := \sum_{S\in\mathbb{S}_{t+1,n}} \sigma_{S,i}$, and $y_k := \prod_{S\in\mathbb{S}_{t+1,n}} y_{S,k}$ for $k = 0, \ldots, n$.
3. Output $(y_0, \sigma_i, (y_1, \ldots, y_n))$.

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

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

We defer a proof to the full version of this work. Note that the protocol realizes the stronger functionality $\widehat{\mathcal{F}}_{\mathsf{KeyGen}}$.

7 A Two-Round Protocol Using Preprocessing

Here we show a fully secure DKG protocol that requires one round of preprocessing, following which it is possible to generate an unbounded number of keys via a 2-round protocol. (Alternately, one may view it as a 2-round protocol assuming trusted setup, or as a 3-round protocol with no setup.) A drawback of the protocol in theory is that is has complexity $O(\binom{n}{t})$; for small values of n, t encountered in practice, however, the protocol is extremely efficient.

Our protocol is based on pseudorandom secret sharing (PRSS) [12] (see also [47, Section 19.4]). A non-interactive PRSS-based protocol for sharing a random secret is well known, but to the best of our knowledge it has not been previously observed that (1) it is possible to (easily) add robustness to that protocol, or that (2) robustness is possible even in the absence of a trusted dealer to distribute the initial PRSS shares.

We begin by recalling the non-interactive PRSS-based protocol for sharing a random secret. Let $\mathbb{S}_{n-t,n}$ be the collection of all subsets of $[n]$ of size $n-t$. For $S \in \mathbb{S}_{n-t,n}$, let $Z_S \in \mathbb{Z}_q[X]$ be the polynomial of degree at most t satisfying $Z_S(0) = 1$ and $Z_S(i) = 0$ for $i \in [n] \setminus S$. (Each Z_S is publicly known.) In an initialization phase a trusted dealer does the following: for every $S \in \mathbb{S}_{n-t,n}$ a uniform key $k_S \in \{0,1\}^\kappa$ is chosen, and each party $i \in S$ is given k_S.

Let $F : \{0,1\}^\kappa \times \{0,1\}^* \to \mathbb{Z}_q$ be a pseudorandom function. The sharing of a secret indexed by a nonce[10] N is done by having each P_i compute its share

$$\sigma_i := \sum_{S \in \mathbb{S}_{n-t,n} \,:\, i \in S} F_{k_S}(N) \cdot Z_S(i). \tag{2}$$

To see that this is a correct $(t+1)$-out-of-n Shamir secret sharing, let f_N be the polynomial

$$f_N(X) \overset{\text{def}}{=} \sum_{S \in \mathbb{S}_{n-t,n}} F_{k_S}(N) \cdot Z_S(X) \tag{3}$$

of degree at most t. Then note that for all $i \in [n]$ we have

$$f_N(i) = \sum_{S \in \mathbb{S}_{n-t,n}} F_{k_S}(N) \cdot Z_S(i) = \sum_{S \in \mathbb{S}_{n-t,n} \,:\, i \in S} F_{k_S}(N) \cdot Z_S(i) = \sigma_i.$$

The value x_N being shared is

$$x_N \overset{\text{def}}{=} f_N(0) = \sum_{S \in \mathbb{S}_{n-t,n}} F_{k_S}(N) \cdot Z_S(0) = \sum_{S \in \mathbb{S}_{n-t,n}} F_{k_S}(N). \tag{4}$$

Since any set $\mathcal{C} \subset [n]$ of t corrupted parties has no information about $k_{[n]\setminus\mathcal{C}}$, this means that x_N is computationally indistinguishable from a uniform element of $\mathbb{Z}_q$ given the view of the parties in $\mathcal{C}$.

The above protocol is robust because it is non-interactive. But it is not a key-generation protocol since, although it allows the parties to compute a sharing of a secret x, it does not allow them to compute $y \overset{\text{def}}{=} g^x$ without further interaction.

[10] The nonce does not need to be random, only non-repeating. It could be a counter, or a session id, or derived in some other agreed-upon fashion by the parties.

Moreover, the protocol as described assumes a trusted dealer who sets up the initial keys. We show how to address both these issues.

At a high level, the idea is as follows. During a preprocessing phase, a designated party in each $S \in \mathbb{S}_{n-t,n}$ chooses a uniform key k_S and sends it over a private channel to each party in S. Let $k_{i,S}$ be the key that P_i receives from the designated party P_S for set S. If P_S is corrupted, it may choose k_S nonuniformly; this will not affect security since P_S learns k_S anyway (regardless of how it is chosen). A more serious problem is that P_S might send different keys to different (honest) parties in S; we will see below how the protocol deals with this. One crucial point, however, is that at least one set $S_{\mathcal{H}} \in \mathbb{S}_{n-t,n}$ contains only honest parties; the key $k_{S_{\mathcal{H}}}$ for that set will be uniform, unknown to the adversary, and shared correctly among all parties in $S_{\mathcal{H}}$.

The key-generation protocol itself has a simple structure. In the first round, each party P_i computes $\hat{y}_{i,S} := g^{F_{k_{i,S}}(N)}$ for each set S of which they are a member, and broadcasts a "commitment" $h_{i,S} := H(\hat{y}_{i,S})$ to that value, where H is a cryptographic hash function. For each set $S \in \mathbb{S}_{n-t,n}$ there are now two possibilities: either all parties in S broadcasted the same value, which we may simply call h_S, or parties in S broadcasted different values. In the latter case all parties simply exclude the set S from further consideration. Let $\mathcal{I} \subseteq \mathbb{S}_{n-t,n}$ be the collection of non-excluded sets.

In the second round, each party P_i broadcasts $\{\hat{y}_{i,S}\}$ for each $S \in \mathcal{I}$ of which they are a member. Each party then sets $\hat{y}_S$ (for $S \in \mathcal{I}$) equal to any valid preimage of h_S that was broadcast. (We discuss below why such a value is guaranteed to exist.) Parties compute their output as in the non-interactive PRSS scheme described earlier, but now summing only over sets in $\mathcal{I}$. Specifically, P_i computes its share as

$$\sigma_i := \sum_{S \in \mathcal{I} \,:\, i \in S} F_{k_{i,S}}(N) \cdot Z_S(i)$$

(compare to (2)), and all parties compute the public key $y := \prod_{S \in \mathcal{I}} \hat{y}_S$ and the commitments $y_j := \prod_{S \in \mathcal{I} \,:\, j \in S} \hat{y}_S^{Z_S(j)}$ for all $j \in [n]$. See Figure 4 for details.

Before discussing security, we briefly sketch correctness. Let $\{\hat{y}_S\}_{S \in \mathcal{I}}$ be the values used by parties to compute the public key and the commitments, and let $x_S \overset{\text{def}}{=} \log_g \hat{y}_S$. Then

$$\log_g y_i = \sum_{S \in \mathcal{I} \,:\, i \in S} x_S \cdot Z_S(i)$$

(compare to (2)), and so the exponents of the $\{y_j\}_{j \in [n]}$ do indeed lie on the degree-t polynomial $f(X) = \sum_{S \in \mathcal{I}} x_S \cdot Z_S(X)$ (compare to (3)). Moreover, the exponent of the public key y is

$$\log_g y = \sum_{S \in \mathcal{I}} x_S = f(0) \tag{5}$$

(compare to (4)), as required. As for the shares computed by the (honest) parties, note that for each $S \in \mathcal{I}$ all honest parties in S must have broadcast the same value h_S in round 1; collision-resistance of H thus implies that every honest party P_i in S holds the same value of $F_{k_{i,S}}(N) = \log_g \hat{y}_S = x_S$. Thus, honest parties' shares are consistent with the publicly computed information.

As for security, one crucial property of the protocol is that—as in the protocols from the previous sections—the public key y is *fully determined* at the end of the first round (regardless of the actions of the corrupted parties in the second round), even though it cannot yet be *computed* by the adversary. Roughly, this is because for each set $S \in \mathbb{S}_{n-t,n}$ containing a corrupted party there are two possibilities: either there is agreement in the $\{h_{i,S}\}_{i \in S}$ or not. In the latter case, the effect is simply to exclude S from $\mathcal{I}$. In the former case, collision-resistance of H ensures that the common value h_S cannot be "opened" to conflicting values $\hat{y}_{i,S} \neq \hat{y}_{j,S}$; moreover, the attacker cannot choose to prevent a preimage of h_S from being revealed since S contains at least one honest party. This shows that the attacker cannot bias the key or prevent it from being computed. Secrecy of the private key holds since $S_{\mathcal{H}}$ is always in $\mathcal{I}$, and hence the pseudorandom contribution of the key $k_{S_{\mathcal{H}}}$ used by the set of honest parties is always included in the computation of the private key (cf. (5)).

```
$\Pi^{t,n}_{\mathrm{ROM}}$

Let $H : \mathbb{G} \to \{0,1\}^\kappa$ be a cryptographic hash function.

Preprocessing: For each $S \in \mathbb{S}_{n-t,n}$, the lowest-index party in $S$ chooses a
    uniform key $k_S \in \{0,1\}^\kappa$ and sends it (over a private channel) to each
    party $P_i$, $i \in S$. Each party $P_i$ sets $k_{i,S}$ to the value thus received.
    If $P_i$ does not receive $k_S$ for some $S \in \mathbb{S}_{n-t,n}$ with $i \in S$, it sets $k_{i,S} := 0$.
Key generation: To generate a key for a nonce $N$, each party $P_i$ does:
    Round 1: For all $S \in \mathbb{S}_{n-t,n}$ with $i \in S$, compute $\hat{y}_{i,S} := g^{F_{k_{i,S}}(N)}$ and
        $h_{i,S} := H(\hat{y}_{i,S})$. Then broadcast $\{h_{i,S}\}_{S \in \mathbb{S}_{n-t,n} : i \in S}$.
        If for some $S \in \mathbb{S}_{n-t,n}$ and $j \in S$, party $P_j$ fails to broadcast $h_{j,S}$,
        then set $h_{j,S} := \perp$.
    Round 2: Initialize $\mathcal{I} := \emptyset$. Then for each $S \in \mathbb{S}_{n-t,n}$, do:
            If there exists $h_S$ with $h_{j,S} = h_S$ for all $j \in S$, add $S$ to $\mathcal{I}$.
        Broadcast $\{\hat{y}_{i,S}\}_{S \in \mathcal{I} : i \in S}$.
    Output determination: Each party $P_i$ does:
        1. For each $S \in \mathcal{I}$, if some party $P_j$ with $j \in S$ broadcasted $\hat{y}_{j,S}$
           with $H(\hat{y}_{j,S}) = h_S$ then set $\hat{y}_S := \hat{y}_{j,S}$.
        2. Set $\sigma_i := \sum_{S \in \mathcal{I} : i \in S} F_{k_{i,S}}(N) \cdot Z_S(i)$.
        3. For $j \in [n]$ set $y_j := \prod_{S \in \mathcal{I} : j \in S} \hat{y}_S^{Z_S(j)}$. Set $y := \prod_{S \in \mathcal{I}} \hat{y}_S$.
        4. Output $(y, \sigma_i, (y_1, \ldots, y_n))$.
```

Fig. 4. A DKG protocol in the ROM, parameterized by t, n.

Theorem 4. *Let F be a pseudorandom function, and model H as a random oracle. Then for $t < n/2$, protocol $\Pi^{t,n}_{\mathrm{ROM}}$ t-securely realizes $\mathcal{F}^{t,n}_{\mathsf{KeyGen}}$.*

Proof. We define a simulator $\mathcal{S}$, given black box access to an adversary $\mathcal{A}$, as follows. (Queries to H, whether by honest parties or by $\mathcal{A}$, are handled in the natural way unless otherwise specified.)

Preprocessing: $\mathcal{S}$ runs $\mathcal{A}$ to obtain a set $\mathcal{C}$ of corrupted parties with $|\mathcal{C}| \leq t$. Let $\mathcal{H} := [n] \setminus \mathcal{C}$ be the set of honest parties. Then:

- For each $S \in \mathbb{S}_{n-t,n}$ with $S \subseteq \mathcal{H}$, do nothing.
- For each $S \in \mathbb{S}_{n-t,n}$ with $S \not\subseteq \mathcal{H}$ in which the lowest-indexed party P_i is honest, choose uniform $k_S \in \{0,1\}^\kappa$ and send k_S to all parties in $S \cap \mathcal{C}$ on behalf of P_i. Also set $k_{i,S} := k_S$ for all $i \in S \cap \mathcal{H}$.
- For each $S \in \mathbb{S}_{n-t,n}$ with $S \not\subseteq \mathcal{H}$ in which the lowest-indexed party is corrupted, receive $\{k_{i,S}\}_{i \in S \cap \mathcal{H}}$ from $\mathcal{A}$. Set $k_S := k_{i,S}$ for arbitrary index $i \in S \cap \mathcal{H}$.

Key generation: Let N be the nonce. Then:

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

- For each $S \in \mathbb{S}_{n-t,n}$ with $S \subseteq \mathcal{H}$: choose uniform $h_S \in \{0,1\}^\kappa$, and for all $i \in S$ broadcast $h_{i,S} := h_S$.
- For each $S \in \mathbb{S}_{n-t,n}$ with $S \not\subseteq \mathcal{H}$, and each $i \in S \cap \mathcal{H}$, run the protocol honestly. (I.e., compute $\hat{y}_{i,S} := g^{F_{k_{i,S}}(N)}$ followed by $h_{i,S} := H(\hat{y}_{i,S})$; then broadcast $h_{i,S}$.)

In response, for $S \in \mathbb{S}_{n-t,n}$ with $S \not\subseteq \mathcal{H}$ and $i \in S \cap \mathcal{C}$, the adversary $\mathcal{A}$ broadcasts $h_{i,S}$. (If $\mathcal{A}$ fails to send some such $h_{i,S}$, then set $h_{i,S} := \perp$.)

Initialize $\mathcal{I} := \emptyset$. Then for each $S \in \mathbb{S}_{n-t,n}$ do:

If there exists h_S such that $h_{i,S} = h_S$ for all $i \in S$, add S to $\mathcal{I}$.

(Note that $\mathcal{I}$ contains all $S \in \mathbb{S}_{n-t,n}$ with $S \subseteq \mathcal{H}$.)

For all $i \in \mathcal{C}$, compute $\sigma_i := \sum_{S \in \mathcal{I} \,:\, i \in S} F_{k_S}(N) \cdot Z_S(i)$. Send $\{\sigma_i\}_{i \in \mathcal{C}}$ to $\mathcal{F}^{t,n}_{\mathsf{KeyGen}}$ and receive in return y and $Y = (y_1, \ldots, y_n)$.

For all $S \in \mathcal{I}$ with $S \not\subseteq \mathcal{H}$, set $\hat{y}_S := g^{F_{k_S}(N)}$. Then choose uniform $\{\hat{y}_S\}_{S \in \mathcal{I},\, S \subseteq \mathcal{H}}$ subject to the constraint that $\prod_{S \in \mathcal{I}} \hat{y}_S = y$. Program H so that $H(\hat{y}_S) = h_S$ for each $S \in \mathcal{I}, S \subseteq \mathcal{H}$.

Round 2: For each $i \in \mathcal{H}$, broadcast $\{\hat{y}_S\}_{S \in \mathbb{S}_{n-t,n} \,:\, i \in S}$. Output whatever $\mathcal{A}$ outputs.

We show that $\textsc{real}_{\Pi^{t,n}_{\text{ROM}}, \mathcal{A}}$ is indistinguishable from $\textsc{ideal}_{\mathcal{F}^{t,n}_{\mathsf{KeyGen}}, \mathcal{S}}$ via a sequence of hybrid experiments. Key observations we rely on, which follow from the fact that $|\mathcal{C}| \leq t < n/2$, are that (1) every $S \in \mathbb{S}_{n-t,n}$ contains at least one honest party, and (2) there is at least one $S \in \mathbb{S}_{n-t,n}$ containing only honest parties. Let Expt_0 refer to experiment $\textsc{real}_{\Pi^{t,n}_{\text{ROM}}, \mathcal{A}}$.

Experiment Expt_1**.** In this experiment, modify Expt_0 in the following way. For each $S \in \mathbb{S}_{n-t,n}$ with $S \subseteq \mathcal{H}$, do the following:

- During initialization, do nothing. (In particular, do not choose any key k_S.)
- In round 1 of key generation, choose uniform $\hat{y}_S \in \mathbb{G}$ and then for each $i \in S$ set $\hat{y}_{i,S} := \hat{y}_S$. Run the rest of the protocol honestly.

It follows immediately from the fact that F is a pseudorandom function that Expt_0 and Expt_1 are indistinguishable.

Experiment Expt_2. We now introduce the following modification to Expt_1. For each $S \in \mathbb{S}_{n-t,n}$ with $S \subseteq \mathcal{H}$, do the following during key generation: in round 1, choose uniform $h_S \in \{0,1\}^\kappa$ and set $h_{i,S} := h_S$ for all $i \in S$. Each $i \in S$ broadcasts $\hat{y}_{i,S}$ in round 2 as before, where $\hat{y}_S$ is defined as in Expt_1. Now, however, H is then programmed so that $H(\hat{y}_S) = h_S$.

If H is modeled as a random oracle, the only difference between Expt_2 and Expt_1 occurs if $\mathcal{A}$ ever queries $H(\hat{y}_S)$ for some $S \subseteq \mathcal{H}$ before $\hat{y}_S$ is broadcast in round 2. Since each such $\hat{y}_S$ is uniform in $\mathbb{G}$, the probability of that event is negligible and so Expt_2 and Expt_1 are indistinguishable.

Experiment Expt_3. Now modify the output-determination step of Expt_2 as follows: for each $S \in \mathcal{I}$ with $S \not\subseteq \mathcal{H}$, set $\hat{y}_S = g^{F_{k_{i,S}}(N)}$ for arbitrary $i \in S \cap \mathcal{H}$. Note that an observable difference between Expt_3 and Expt_2 can only possibly occur if for some $S \in \mathbb{S}_{n-t,n}$, $i \in S \cap \mathcal{H}$, and $j \in S \cap \mathcal{C}$, parties P_i, P_j broadcast $h_{i,S} = h_{j,S}$ in round 1 and $\hat{y}_{i,S} \neq \hat{y}_{j,S}$ in round 2 but $H(\hat{y}_{i,S}) = H(\hat{y}_{j,S})$. Collision-resistance of H—which follows when H is modeled as a random oracle—thus implies that Expt_3 and Expt_2 are indistinguishable.

Experiment Expt_4. Now, instead of choosing uniform and independent values $\{\hat{y}_S\}_{S \in \mathbb{S}_{n-t,n}, S \subseteq \mathcal{H}}$, we instead choose a uniform value $y \in \mathbb{G}$ and then choose uniform $\{\hat{y}_S\}_{S \in \mathbb{S}_{n-t,n}, S \subseteq \mathcal{H}}$ subject to the constraint that $\prod_{S \in \mathcal{I}} \hat{y}_S = y$. This is perfectly indistinguishable from Expt_3, and it can be verified that this experiment is identical to $\text{IDEAL}_{\mathcal{F}^{t,n}_{\mathsf{KeyGen}}, \mathcal{S}}$. □

Achieving adaptive security. Assuming secure erasures, it is easy to achieve adaptive security for the above protocol by having each party P_i make a simple change: after computing $\{F_{k_{i,S}}(N)\}_{i \in S}$ in round 1, update all keys by setting $k_{i,S} := H'(k_{i,S})$ for all S with $i \in S$, where $H' : \{0,1\}^\kappa \to \{0,1\}^\kappa$ is an independent random oracle.

8 A One-Round Protocol Using Preprocessing

We show here that, given certain setup, it is possible to construct a *one*-round fully secure DKG protocol. This does not contradict the impossibility result from Section 4 since the protocol we construct will not be unbiased—in fact, the key will be a deterministic function of the setup. Such a protocol can still be fully secure as long as the key is *computationally indistinguishable from uniform* even conditioned on the setup given to the corrupted parties, as will be the case in our protocol. (The astute reader may notice that the protocol in the previous section is not unbiased, either.) As in the previous section, we show that it is possible to generate the appropriate setup in a preprocessing phase run by the parties themselves. Here, however, that preprocessing phase requires two rounds.

Note that the impossibility result in Section 4 is still interesting since any protocol relying on setup that does not involve private correlations between the parties (e.g., the CRS+PKI models and/or the ROM) will likely be unbiased.

$\Pi^{t,n}_{\text{1-round}}$

We assume a CRS used for the required NIZK proofs.

Preprocessing:

Round 1: For each $S \in \mathbb{S}_{n-t,n}$, the lowest-index party in S chooses uniform $k_S \in \{0,1\}^\kappa$ and $\omega_S \in \{0,1\}^*$, computes $\mathsf{com}_S := \mathsf{Com}(k_S; \omega_S)$, broadcasts com_S, and sends k_S, ω_S by private channel to all P_i, $i \in S$.

Round 2: Each party P_i does the following: for each $S \in \mathbb{S}_{n-t,n}$ with $i \in S$, if P_S did not broadcast com_S or did not send k_S, ω_S to P_i, or if it did but $\mathsf{com}_S \neq \mathsf{Com}(k_S; \omega_S)$, then broadcast $(\mathsf{complaint}, S)$.

Let $\mathcal{I} = \{S \in \mathbb{S}_{n-t,n} \mid$ no party in S broadcast $(\mathsf{complaint}, S)\}$.

Key generation: To generate a key for nonce N, each party P_i does:

Round 1: For each $S \in \mathcal{I}$ with $i \in S$, compute $\hat{y}_{i,S} := g^{F_{\mathsf{sk}_{i,S}}(N)}$. Broadcast $\hat{y}_{i,S}$, along with an NIZK proof $\pi_{i,S}$ that $\hat{y}_{i,S}$ is computed correctly relative to $\mathsf{com}_{i,S}$.

Output determination: Each party P_i does:

1. For each $S \in \mathcal{I}$, if some P_j with $j \in S$ broadcasted $\hat{y}_{j,S}$ along with a valid proof $\pi_{j,S}$ relative to $\mathsf{com}_{j,S}$, then set $\hat{y}_S := \hat{y}_{j,S}$.
2. Set $\sigma_i := \sum_{S \in \mathcal{I} \,:\, i \in S} F_{k_{i,S}}(N) \cdot Z_S(i)$.
3. For $j \in [n]$ set $y_j := \prod_{S \in \mathcal{I} \,:\, j \in S} \hat{y}_S^{Z_S(j)}$. Set $y := \prod_{S \in \mathcal{I}} \hat{y}_S$.
4. Output $(y, \sigma_i, (y_1, \ldots, y_n))$.

Fig. 5. A DKG protocol in the CRS model, parameterized by t, n.

The protocol we describe is similar to the protocol in the previous section, and in particular we again rely on non-interactive PRSS. Let $\mathbb{S}_{n-t,n}$ continue to denote the collection of all subsets of $[n]$ of size $n-t$ and, for $S \in \mathbb{S}_{n-t,n}$, let $Z_S \in \mathbb{Z}_q[X]$ be the polynomial of degree at most t satisfying $Z_S(0) = 1$ and $Z_S(i) = 0$ for $i \in [n] \backslash S$. As in the previous protocol, during a preprocessing phase a designated party P_S in each $S \in \mathbb{S}_{n-t,n}$ chooses $k_S \in \{0,1\}^\kappa$ and sends it (over a private channel) to all parties in S. Now, however, P_S additionally broadcasts a commitment com_S to k_S, and sends the decommitment information to all parties in S. (It suffices for the commitment scheme to be computationally hiding and binding. In the CRS model, a statistically binding non-interactive commitment scheme can be constructed from one-way functions.) In the second round of preprocessing, any party in S who does not receive a correct decommitment to com_S broadcasts a complaint; any set S for which there is a complaint is excluded from consideration from then on. Let $\mathcal{I}$ be the collection of non-excluded sets.

To generate a key for a nonce N during the key-generation phase, each party P_i simply broadcasts $\hat{y}_{i,S} := g^{F_{k_{i,S}}(N)}$ for each set $S \in \mathcal{I}$ to which they belong, plus an NIZK proof of correctness relative to $\mathsf{com}_{i,S}$. Let $\hat{y}_S \in \{\hat{y}_{i,S}\}_{i \in S}$ be such that the associated proof verifies; there will always be one such value (since each set S includes at least one honest party) and there cannot be more than one (by soundness of the NIZK proof). As in the previous protocol, all parties then compute the public key $y := \prod_{S \in \mathcal{I}} \hat{y}_S$ and the commitments $y_j := \prod_{S \in \mathcal{I} \,:\, j \in S} \hat{y}_S^{Z_S(j)}$ for all $j \in [n]$. See Figure 5 for details.

We defer a proof of the following to the full version of this work.

Theorem 5. *Let F be a pseudorandom function, and assume $\Pi^{t,n}_{\text{1-round}}$ uses a secure commitment scheme and an identity-based simulation-sound NIZK proof system. Then for $t < n/2$, protocol $\Pi^{t,n}_{\text{1-round}}$ t-securely realizes $\mathcal{F}^{t,n}_{\mathsf{KeyGen}}$.*

In the full version, we show how to modify the protocol to work in the plain model while keeping the round complexity (in both the preprocessing and key-generation phases) the same. In particular, we can avoid assuming a CRS by having the parties generate a suitable CRS as part of preprocessing [27].

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.
3. R. Bacho and J. Loss. On the adaptive security of the threshold BLS signature scheme. In *29th ACM Conf. on Comp. and Comm. 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 Comp. and Comm. 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, *Adv. 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 *Adv. 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 Symp. 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. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *2nd Theory of Cryptography Conference—TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, 2005.
13. 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`.

14. I. Damgård, B. Magri, D. Ravi, L. Siniscalchi, and S. Yakoubov. Broadcast-optimal two round MPC with an honest majority. In *Adv. in Cryptology—Crypto 2021, Part II*, volume 12826 of *LNCS*, pages 155–184. Springer, 2021.
15. S. Das, T. Yurek, Z. Xiang, A. K. Miller, L. Kokoris-Kogias, and L. Ren. Practical asynchronous distributed key generation. In *IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE, 2022.
16. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Adv. in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, 2001.
17. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 427–437. IEEE, 1987.
18. P.-A. Fouque and J. Stern. One round threshold discrete-log key generation without private channels. In *4th Intl. Workshop on Theory and Practice in Public Key Cryptography—PKC 2001*, volume 1992 of *LNCS*, pages 300–316. Springer, 2001.
19. R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *25th ACM Conf. on Comp. and Comm. Security (CCS)*, pages 1179–1194. ACM Press, 2018.
20. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. On 2-round secure multiparty computation. In *Adv. in Cryptology—Crypto 2002*, volume 2442 of *LNCS*, pages 178–193. Springer, 2002.
21. 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.
22. 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.
23. 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`.
24. 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.
25. S. D. Gordon, F.-H. Liu, and E. Shi. Constant-round MPC with fairness and guarantee of output delivery. In *Adv. in Cryptology—Crypto 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, 2015.
26. J. Groth. Non-interactive distributed key generation and key resharing, 2021. Available at `https://eprint.iacr.org/2021/339`.
27. J. Groth and R. Ostrovsky. Cryptography in the multi-string model. In *Adv. in Cryptology—Crypto 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, 2007.
28. J. Groth and V. Shoup. Design and analysis of a distributed ECDSA signing service, 2022. Available at `https://eprint.iacr.org/2022/506`.
29. 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.
30. Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *Adv. in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, 2014.

31. 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.
32. A. Joux. A one-round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, 2004.
33. J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions. In *29th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 68–80. IEEE, 1988.
34. A. Kate, Y. Huang, and I. Goldberg. Distributed key generation in the wild, 2012. Available at `https://eprint.iacr.org/2012/377`.
35. A. Kate, E. Mangipudi, P. Mukherjee, H. Saleem, and S. Thyagarajan. Non-interactive VSS using class groups and application to DKG, 2023. Available at `https://eprint.iacr.org/2023/451`.
36. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, 3rd edition.* Chapman & Hall/CRC Press, 2020.
37. 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.
38. 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 Comp. and Comm. Security (CCS)*, pages 1751–1767. ACM Press, 2020.
39. 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`.
40. 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.
41. Y. Lindell. Simple three-round multiparty Schnorr signing with full simulatability, 2022. Available at `https://eprint.iacr.org/2022/374`.
42. Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *25th ACM Conf. on Comp. and Comm. Security (CCS)*, pages 1837–1854. ACM Press, 2018.
43. W. Neji, K. Blibech Sinaoui, and N. Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Secur. Commun. Networks*, 9(17):4585–4595, 2016.
44. A. Patra and D. Ravi. On the exact round complexity of secure three-party computation. In *Adv. in Cryptology—Crypto 2018, Part II*, volume 10992 of *LNCS*, pages 425–458. Springer, 2018.
45. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Adv. in Cryptology—Crypto '91*, volume 576 of *LNCS*, pages 129–140. Springer, 1992.
46. N. Shrestha, A. Bhat, A. Kate, and K. Nayak. Synchronous distributed key generation without broadcasts, 2021. Available at `https://eprint.iacr.org/2021/1635`.
47. N. Smart. *Cryptography Made Simple.* Springer, 2016.
48. 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. While we do not claim any particular novelty for any of the functionalities we present, we do think that having them all in one place and following a common format will be useful for researchers pursuing future work in this area.

Note that all functionalities are implicitly parameterized by t and n, though we leave this implicit to reduce notational clutter.

We begin by showing in Figure 6 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. $\widehat{\mathcal{F}}_{\mathsf{KeyGen}}$ is arguably a more natural functionality for DKG than $\mathcal{F}_{\mathsf{KeyGen}}$, though—as we have already noted—using the latter does not seem to have any practical disadvantages while potentially allowing for more-efficient protocols.

$\widehat{\mathcal{F}}_{\mathsf{KeyGen}}$

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.

Fig. 6. Alternate ideal functionality for fully secure key generation.

One can, of course, also consider weaker ideal functionalities. This is essential in the dishonest-majority setting where full security is impossible to achieve; it may also be interesting for exploring tradeoffs between efficiency and security. For simplicity, the remaining functionalities we introduce allow the adversary 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.

In Figure 7 we consider two non-robust functionalities. The first variant, denoted $\mathcal{F}_{\mathsf{KeyGen}}^{\perp}$, corresponds to a "secure-with-abort" version of $\mathcal{F}_{\mathsf{KeyGen}}$ where the 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. We also define $\mathcal{F}_{\mathsf{KeyGen}}^{\perp,\mathsf{fair}}$, a version of the key-generation functionality that also lacks guaranteed output delivery, but forces the adversary to determine whether to abort *independent* of the value of the key. We conjecture that the DKG protocol recently constructed by Komlo et al. [39] could be shown to realize $\mathcal{F}_{\mathsf{KeyGen}}^{\perp,\mathsf{fair}}$.

Either of the above functionalities could be augmented to also have "identifiable abort" [30], meaning that some corrupted party is identified in case the functionality is aborted. This could be done by requiring an adversary who sends an abort message to also specify some index $i \in \mathcal{C}$ which would then be sent to all honest parties along with $\perp$.

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

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.

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

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 $\sigma_i \leftarrow \mathbb{Z}_q$ 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.

Fig. 7. Non-robust key-generation functionalities.

Finally, in Figure 8 we consider a DKG functionality $\mathcal{F}_{\mathsf{KeyGen}}^{\Delta}$ that explicitly allows for (a limited type of) adversarial *bias*. (For simplicity, we incorporate robustness but one could also consider weaker versions without it.) Specifically, it allows an adversary who has corrupted at least one party to specify an additive shift Δ that is added to an (unknown) key x'. We conjecture that a non-robust version of this functionality is realized by the DKG protocols of [21, 29], and that the protocols they consider (shown to be secure when using those DKG protocols) can be proven secure in a corresponding hybrid model.

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

1. Choose $x' \leftarrow \mathbb{Z}_q$ and compute $y' := g^{x'}$.
2. Send y' to the adversary, and receive $(\Delta, \{\sigma_i\}_{i \in \mathcal{C}})$ in return.
3. If $|\mathcal{C}| \geq 1$ set $x := x' + \Delta$; otherwise, set $x := x'$. Set $y := g^x$.
4. Compute $\{\sigma_i\}_{i=0}^{n} \leftarrow \mathsf{SS}_t(x)$. For $i \in [n]$, set $y_i := g^{\sigma_i}$; let $Y = (y_1, \ldots, y_n)$.
5. For $i \in [n]$, send (y, σ_i, Y) to P_i. Also send (y, Y) to the adversary.

Fig. 8. Ideal functionality for key generation allowing additive bias.