On the Adaptive Security of the Threshold BLS Signature Scheme

Renas Bacho
CISPA Helmholtz Center for Information Security
bacho@cispa.de

Julian Loss
CISPA Helmholtz Center for Information Security
loss@cispa.de

ABSTRACT

Threshold signatures are a crucial tool for many distributed protocols. As shown by Cachin, Kursawe, and Shoup (PODC '00), schemes with *unique signatures* are of particular importance, as they allow to implement distributed coin flipping very efficiently and without any timing assumptions. This makes them an ideal building block for (inherently randomized) asynchronous consensus protocols. The threshold-BLS signature of Boldyreva (PKC '03) is both unique and very compact, but unfortunately lacks a security proof against adaptive adversaries. Thus, current consensus protocols either rely on less efficient alternatives or are not adaptively secure. In this work, we revisit the security of the threshold BLS signature by showing the following results, assuming t *adaptive* corruptions:

- We give a modular security proof that follows a two-step approach: 1) We introduce a new security notion for distributed key generation protocols (DKG). We show that it is satisfied by several protocols that previously only had a *static security proof.* 2) Assuming *any* DKG protocol with this property, we then prove unforgeability of the threshold BLS scheme. Our reductions are *tight* and can be used to substantiate real-world parameter choices.
- To justify our use of strong assumptions such as the algebraic group model (AGM) and the hardness of one-more-discrete logarithm (OMDL), we prove two impossibility results: 1) Without the AGM, there is no *tight* security reduction from $(t+1)$-OMDL. 2) *Even in the AGM*, $(t+1)$-OMDL is the weakest assumption from which *any* (possibly loose) security reduction exists.

KEYWORDS

Threshold Signatures, BLS Signatures, Algebraic Group Model

1 INTRODUCTION

Threshold signatures are a special type of digital signature that allows a sufficiently large set of $t+1$ signers to jointly create a compact signature σ on a message m. At the same time, it should be infeasible for t or less signers to create a signature on m. For this reason, t is usually referred to as the *threshold* of the scheme. In this manner, one can create a very size-efficient proof that at least $t+1$ parties have signed m. This makes threshold signatures an important building block for storage-sensitive systems such as blockchain protocols. Another intriguing application of threshold signatures is *distributed coin flipping*. Using a threshold signature scheme with unique signatures (per message m and public key pk) and a non-interactive signing procedure, one can efficiently agree on an unpredictable and unbiasable coin $b \in \{0,1\}$ among n parties $P_1, \ldots, P_n$ as follows:

- Each party P_i (non-interactively) creates a share σ_i of some predetermined message m and sends σ_i to everybody.
- Upon collecting $t+1$ shares σ_i, a party locally reconstructs the signature σ for m.
- All parties can now derive the coin via $b := \mathsf{LSB}(\mathsf{H}(\sigma))$, where H is a suitable randomness extractor, e.g., a hash function (modelled as a random oracle).

Note that in the above construction, the uniqueness property is crucially used in two places. First, it ensures that all parties agree on the *same signature* σ, and, by extension, on the same coin b. Second, uniqueness prevents a malicious adversary from biasing the outcome of the coin b by sending or withholding particular signature shares. Finally, σ (and therefore b) remains unpredictable to an adversary controlling at most t parties up until the point where the first honest party P_i participates in the coin flip by sending its share σ_i. These combined features make (unique and non-interactive) threshold signatures a crucial tool for the design of efficient randomized consensus protocols [3, 5, 16, 34, 47]. This applies particularly to the fully asynchronous network setting, where consensus is known to be *impossible* unless randomized protocols are used [28].

Static vs. Adaptive Corruptions. Cachin et al. [16] were the first to realize the enormous potential of unique threshold signatures for building efficient asynchronous consensus algorithms. Their signature of choice was the threshold version of the full-domain-hash RSA signature [54]. However, this scheme is only secure against a *static adversary* who chooses all corrupted parties at the beginning of the protocol (after observing their public keys). Modern systems, on the other hand, often require security against a much more powerful *adaptive adversary* who dynamically corrupts parties over time by observing the flow of the protocol execution. In addition, RSA signatures are rather large, taking up an order of magnitude more storage space than schemes based on elliptic curve cryptography. Therefore, an appealing alternative is the much more size-efficient scheme of Boldyreva [12], which is based on the BLS signature scheme. Unfortunately, however, Boldyreva's scheme also lacks an adaptive security proof. To overcome these limitations, Libert et al. [41] proposed an adaptively secure construction based on Boldyreva's scheme, which, thus far, has served as the state-of-the-art for building adaptively secure consensus protocols [3, 5, 34, 47]. While their construction is still far more size-efficient than an RSA signature, it is roughly twice as expensive to store and verify as signatures in Boldyreva's original scheme. In addition, while Boldyreva's signature is compatible with modern BLS libraries [1], Libert et al.'s scheme lacks such a compatibility. (There is, however, an efficient implementation of their scheme available at [50].) Motivated by the above discussion, we ask: *What are the adaptive security guarantees of the threshold BLS signature scheme?*

1.1 Handling Adaptive Corruptions

Our starting point is the construction of Libert et al. who gave the first adaptively secure, non-interactive, and unique threshold signature scheme in the random oracle model. Their adaptive security argument also extends to the distributed key generation (DKG) phase that sets up the shared keys for parties in the system. The established way of proving security for a DKG protocol is to argue that the messages that are exchanged as part of the protocol reveal nothing further about the distributed secret key *sk* (beyond what is already revealed by *pk*) [33]. This can be done by providing an efficient *simulator* Sim that, on input *pk*, provides a properly distributed view of an execution of the DKG protocol in which parties agree on the public key *pk*. In case corruptions are static, Sim also gets the set of corrupted parties as input. Assuming that Sim provides a perfect simulation (as indeed is often the case), this technique works even against an information-theoretic adversary who can compute *sk* from *pk* by brute force. Clearly, such an adversary could easily forge a signature with respect to *sk*, so what does this mean?

The Challenge of Adaptive Corruptions. One of the main insights of Libert et al. is to prove unforgeability of their scheme directly by reducing from a computational assumption. In this manner, their proof bypasses many of the issues encountered in the DKG literature when having to deal with adaptive corruptions. However, one central issue still remains: Sim needs to simulate correctly distributed internal states (including secret keys) of parties upon corruption. Existing DKG protocols overcome this problem by relying on heavy tools such as non-committing encryption and erasures [19], [35]. This is a common and often frustrating issue to deal with in the context of simulation based security proofs. Namely, even if a statically secure protocol can not be simulated, it is far from clear wether it would actually be *insecure* in the presence of adaptive corruptions. This issue is also quite prominent in the context of Threshold BLS signatures, even when a trusted dealer distributes the keys. Recall that in order to create a signature share σ_i on message m, party P_i computes $\mathsf{H}(m)^{sk_i}$. Here, $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{G}$ is a hash function that is modelled as a random oracle, $\mathbb{G}$ is a cyclic group of known prime order p, and $sk_i \in \mathbb{Z}_p$ denotes P_i's secret key share. The corresponding public key shares of parties are g^{sk_i}, where g is a known generator of $\mathbb{G}$. Now, assume that keys have been set up in such a way that for all $i \in [n]$, $sk_i = f(i)$ and $sk = f(0)$ for some suitable polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Then one can compute a signature σ that verifies relative to $pk = g^{sk}$ from $t+1$ shares $\sigma_1, \ldots, \sigma_{t+1}$ by interpolating f in the exponent of g. It can be verified by checking the symmetric pairing equation $e(\sigma, g) = e(\mathsf{H}(m), pk)$. When dealing with adaptive corruptions, the issue is now that by sending a share $\sigma_i = \mathsf{H}(m)^{sk_i}$ an honest party P_i *commits* itself to its secret key sk_i. Hence, the simulator Sim must output sk_i upon P_i being adaptively corrupted. This, however, is challenging: if Sim knew sk_i for all $i \in [n]$, then it would also know sk. On the other hand, if it *does not know* at least $n - t$ of the sk_i, then it might fail during simulation.

Libert et al.'s Approach. Libert et al. circumvent this problem as follows. Their scheme uses asymmetric pairing groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ of order p. They define their secret keys as $sk_i = (f(i), g(i), h(i), j(i)) \in \mathbb{Z}_p^4$, where f, g, h, j are independent polynomials of degree t. A signature share on m is then computed as $\sigma_i = (z_i, r_i) \in \mathbb{G}^2$, where $z_i = \mathsf{H}_1^{f(i)}(m) \cdot \mathsf{H}_2^{g(i)}(m)$ and $r_i = \mathsf{H}_1^{h(i)}(m) \cdot \mathsf{H}_2^{j(i)}(m)$ and $\mathsf{H}_1, \mathsf{H}_2$ are independent random oracles. *pk* is correspondingly set as $(g_1, g_2) = \left(g_z^{f(0)} \cdot g_r^{g(0)}, g_z^{h(0)} \cdot g_r^{j(0)}\right) \in \hat{\mathbb{G}}^2$, where g_z and g_r are two random generators of $\hat{\mathbb{G}}$. Similar as for threshold BLS, one can can compute a signature $\sigma = (z, r)$ from $t+1$ shares by interpolation in the exponent and verify it by checking whether $e(z, g_z) \cdot e(r, g_r) \cdot e(\mathsf{H}_1(m), g_1) \cdot e(\mathsf{H}_2(m), g_2) = 1$. In this manner, Libert et al.'s signature is *computationally unique* under the so-called double-pairing assumption (see [46] for a proof). The latter implies that it should be hard to find $(z', r') \neq (z, r)$ which also satisfies the above equation for the same m. At the same time, this flexibility is what allows their reduction (to the symmetric external Diffie-Hellman assumption) to go through. Namely, as their signatures do not commit the signer to a secret key, they can efficiently simulate a secret key at the appropriate point in the simulation where a party becomes corrupted. Unfortunately, the technique of Libert et al. does not work in the context of the original threshold BLS signature. Hence, to deal with adaptive corruptions, a completely new approach is required.

1.2 Adaptive Security from Oracle-Aided Simulatability

We begin by describing our idea for the simplified case when a trusted dealer computes and distributes the keys according to Boldyreva's original description of the threshold BLS scheme. Let us briefly recall our desired security notion of *unforgeability under chosen message attacks* in the context of threshold signatures. In this game, the adversary first observes the public key shares pk_i of all parties P_i. (In case the keys are distributed via some DKG protocol, the adversary also observes its execution as part of the game). Next, it repeatedly gets access to a signing oracle, which takes in a pair (i, m) and returns a signature share σ_i that is valid under pk_i. The adversary can also adaptively corrupt any hitherto uncorrupted party P_i, upon which it learns P_i's secret key sk_i (and any other internal variables held by P_i at the point of corruption). The adversary is considered successful if it can produce a forgery on a message m^* for which it has observed a total of fewer than $t+1$ shares from corruptions and signing queries.

Reducing from One-More Discrete Logarithm. As already explained, the critical difficulty is to simulate the values of sk_i upon an adaptive corruption. Our key idea is to aid the simulator by giving it t-time access to a discrete logarithm oracle $\mathsf{DL}_g(\cdot)$ which, on input $h = g^x \in \mathbb{G}$, returns the discrete logarithm x of h to base g. To facilitate such an oracle in our simulation, we reduce from the *one-more discrete logarithm (OMDL) assumption of degree* $t+1$. Recall that in the OMDL assumption of degree k, the adversary is given an instance $(g^{x_1}, \ldots, g^{x_k})$ and gets $(k-1)$-time access to $\mathsf{DL}_g(\cdot)$. It is considered successful if it can produce the values of $x_1, \ldots, x_k \in \mathbb{Z}_p$. Moreover, we rely on the algebraic group model (AGM) [29] to obtain a suitable system of linear equations that allow to solve the OMDL instance. Both of these tools have recently been popular choices for proving involved cryptosystems [30, 37, 51, 56]. However, one might wonder whether such strong assumptions are truly necessary for proving adaptive security of the threshold BLS signature.

The Necessity of Strong Assumptions. We answer this question positively. Concretely, we show that there is no algebraic, non-rewinding reduction from the OMDL assumption of degree t (or lower) to the adaptive security of the threshold BLS scheme with corruption threshold t. Our impossibility result follows the common metareduction template [22, 36]: assuming an efficient reduction R as above, we show that one can obtain an efficient solver M (the metareduction) for the OMDL problem of degree t. As OMDL is assumed to be hard for any polynomial degree t, it follows that such a reduction R can not exist. Our metareduction also applies to reductions which are not fully black-box. In particular, we rule out reductions which themselves may rely on the AGM. We can combine our result with the separation of Bauer et al. [9] who showed that in the AGM, the OMDL assumption of any degree is a stronger assumption than any conceivable static assumption. Hence, our result shows that even in the AGM, the OMDL assumption of degree $t + 1$ is both necessary and sufficient to prove security, unless one uses a rewinding reduction. This, however, would have a devastating impact on the tightness of the reduction, and would require much larger parameters for concrete security than what is used in real-world implementations for BLS signatures [1]. In contrast to this, our reduction in the AGM is tight and hence justifies parameters currently used in practice. To further justify our reliance on the AGM, we also provide a metareduction along the lines of Coron [22] to prove that there does not exist a tight black-box reduction to the one-more discrete logarithm assumption of degree $t + 1$ in the plain random oracle model. We remark that Coron's original metareduction did not consider reductions from interactive assumptions such as OMDL. Unsurprisingly, giving a reduction R access to the oracle $\mathsf{DL}_g(\cdot)$ complicates matters significantly, as we now have to simulate $\mathsf{DL}_g(\cdot)$ to R as part of our metareductions.

Replacing the Trusted Dealer. We now turn our attention to the more realistic setting in which no trusted dealer is available for setting up keys. In this setting, existing DKG protocols are either only statically secure or can be used exclusively with signature schemes that do not commit a party to her secret key sk_i whenever she uses it to issue a signature share σ_i. Fortunately, we can appropriately modify our ideas from above so as to handle adaptive corruptions in the unforgeability game even when it is extended with a DKG phase. In some more detail, we begin by proposing a new (and rather weak) security definition for DKG protocols that we refer to as *oracle-aided simulatability.* Informally, this definition asserts the existence of an efficient simulator Sim that can simulate an execution of the DKG protocol (with adaptive corruptions), given some number of queries to a discrete logarithm oracle $\mathsf{DL}_g(\cdot)$. While this definition may seem somewhat artificial at first glance, we show that it is actually sufficient to prove unforgeability against chosen message attacks for the threshold BLS scheme with adaptive corruptions. For the proof, we show a reduction from the OMDL assumption of appropriate degree. The reduction runs Sim internally so as to provide a simulation of the DKG protocol as part of the broader simulation of the unforgeability experiment. To emulate the oracle $\mathsf{DL}_g(\cdot)$ toward Sim, the reduction simply forwards any query Sim directs to $\mathsf{DL}_g(\cdot)$ to its own discrete logarithm oracle.

We stress that oracle-aided simulatability is a security notion for DKG protocols and can be proven completely independently from the context of threshold BLS signatures. Thus, our definition adds a useful layer of modularity: it allows future DKG designers to build protocols that can be directly integrated with our (adaptive) security proofs. To motivate our new notion even further, we show that for several DKG protocols from the literature that do not satisfy full simulatability with adaptive corruptions, it is possible to show oracle-aided simulatability. Finally, we show that all of our metareductions also apply to DKGs with oracle-aided simulability (recall that our above discussion was for a trusted dealer).

1.3 Related Work

Threshold signatures were first conceived by Desmedt [26]. They have recently received significant attention [6, 8, 14, 17, 18, 23, 24, 31, 32, 39, 40, 42–44], mainly in the context of blockchain systems and cryptocurrency wallets. Most of these works focus on the ECDSA and Schnorr threshold signatures, as these are the most widely used schemes in major cryptocurrencies. Note however, that these schemes do not have unique signatures and hence do not lend themselves to distributed coin flipping. A closely related (and also very active) line of research has also studied *multi-signatures* [7, 10, 13, 23, 27, 51, 52]. These can be seen as a threshold signature scheme where the threshold t is always set to $n - 1$, i.e., signing always requires *all parties* to contribute. In contrast to threshold signatures, multi-signatures usually focus on obtaining compact n-out-of-n signatures using parties' native public keys for signing. Thus, no trusted dealer or DKG is necessary to run these protocols.

Distributed Key Generation. There are numerous DKG protocols when the underlying network is synchronous [19, 33, 35, 53, 55]. Among these, only the protocols of Canetti et al. [19] and Jarecki and Lysyanskaya [35] provide adaptive security. Both of these works rely on heavy assumptions and/or cryptographic tools such as non-committing encryption. All of these protocols (except that of Shrestha et al. [55]) rely on public broadcast channels being available. On the other hand, the asynchronous setting has only recently been explored by works of Kokoris-Kogias et al. [38], Abraham et al. [4], and Das et al.[25]. Among these, only the work of Kokoris-Kogias et al. provides adaptive security, but is substantially less efficient than its statically secure alternatives. We also remark that the asynchronous DKG of Abraham et al. [4] produces a group element as the shared secret key rather than a field element and hence can not be used for most conventional signature schemes. This drawback was resolved by Das et al. [25] without increasing the total communication cost of $O(\lambda n^3)$ bits.

VRFs and Distributed Coin Flipping. Distributed randomness generation is an integral component of many distributed protocols. This applies in particular to the asynchronous model, where most distributed protocols of interest are inherently randomized. Asynchronous coin flips rely either on verifiable secret sharing [15, 20] or threshold signatures [3, 5, 16, 34, 47]. Synchronous protocols [2, 3, 48] can rely on a simpler alternative of flipping coins via *verifiable random functions (VRF)* [49]. On input a message m and a secret key sk, a VRF F produces a pseudorandom string $r = F(sk, m)$ along with a proof ϱ that can be used to verify correct generation of r.

To agree on a single bit among n parties in the kth round, a party P_i computes $r_i = F(sk_i, j)$ along with a proof ϱ_i. It then samples $b_i \leftarrow \{0,1\}$ uniformly and sends (b_i, r_i, ϱ_i) to everybody at the beginning of the round. Parties wait to receive messages from other parties until the end of the round. All parties derive the coin as $b := b_j$ where $j = \min_i\{r_i\}$ and the minimum goes over all r_i for which ϱ_i correctly verified. If j belongs to an honest party, then all honest parties indeed agree on a random bit b. Moreover, since F produces pseudorandom outputs and there is a unique output per party and coin flip j, this happens with probability $p \geq \frac{1}{2}$ when the majority of the parties is honest, i.e., b can not be biased.[1] In an asynchronous network, however, this approach utterly fails, as there is no notion of a synchronous round. Note that in order to guarantee liveness of the protocol, every honest party can only wait for up to $2n/3 + 1$ messages from other parties. But in that case, the adversary simply delays the $n/3$ messages (b_i, r_i, ϱ_i) of honest parties with the lowest r_i values. The honest parties receive the remaining $n/3 + 1$ messages from honest parties and $n/3 - 1$ messages from the adversary. Now, with overwhelming probability in n, the smallest r_i always belong to a corrupt party and hence the coin is almost completely under the control of the adversary. Note that this issue does not occur for threshold signatures, as parties can all reconstruct the same coin after receiving only $t+1$ messages from other parties. Some works such as [11, 21] have shown how to circumvent this issue by either relying on strong setup assumptions or assuming a non-standard version of the asynchronous model in which the adversary can not reorder messages of honest parties arbitrarily.

2 PRELIMINARIES AND DEFINITIONS

In this chapter, we introduce basic notation, definitions, and our model in which we will work.

2.1 General Notation

Let λ denote the security parameter. Throughout this paper, we assume that global parameters $par = (\mathbb{G}, \mathbb{G}_T, p, g, e)$ are fixed and known to all parties. Here, $\mathbb{G}$ is a cyclic group of prime order p generated by g and endowed with a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. For concrete choices, we will assume $\lambda = 128$ and that $\mathbb{G}$ is instantiated with a 256-bit elliptic curve. We denote by $\mathbb{G}^*$ the set $\mathbb{G} \setminus \{1\}$ where 1 is the neutral element of $\mathbb{G}$. We denote the set of integers by $\mathbb{Z}$, the set of positive integers by $\mathbb{N}$, the group of integers modulo p by $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ and its multiplicative unit group by $\mathbb{Z}_p^*$. We denote the set of integers from a to b by $[a, b]$ and the set of positive integers from 1 to a by $[a]$. We define the Vandermonde matrix $V(x_1, \ldots, x_r)$ for the $r \geq 1$ numbers $x_1, \ldots, x_r \in \mathbb{Z}_p$ as

$$V(x_1, \ldots, x_r) := \begin{pmatrix} 1 & x_1^1 & x_1^2 & \cdots & x_1^{r-1} \\ \vdots & & & & \vdots \\ 1 & x_r^1 & x_r^2 & \cdots & x_r^{r-1} \end{pmatrix},$$

which is known to be invertible if and only if the x_i are pairwise distinct. For an element x in a set S, we write $x \leftarrow S$ to indicate that x was sampled from S uniformly at random. All our algorithms may be randomized (unless stated otherwise) and written in uppercase letters. By $x \leftarrow \mathsf{A}(x_1, \ldots, x_n)$ we mean running algorithm A on inputs $(x_1, \ldots, x_n)$ and uniformly random coins and then assigning the output to x. If A has oracle access to some algorithm B during its execution, we write $x \leftarrow \mathsf{A}^{\mathsf{B}}(x_1, \ldots, x_n)$. Finally, we write $\mathbf{G}^{\mathsf{A}}$ to denote the output of the experiment $\mathbf{G}$ involving algorithm A.

2.2 Assumptions and Definitions

In this section, we introduce our model and the one-more discrete logarithm assumption, which will be the hardness assumption on which some of our results are based.

The Communication Model. We consider a set of n parties P_1, $\ldots, P_n$ (modelled as PPT machines). We assume that the parties are connected by a complete network of bilateral private and authenticated channels. Additionally, the parties have access to a dedicated broadcast channel. We assume synchronous communication: parties have access to a global clock and computation proceeds in synchronized rounds of known length Δ. When an honest party sends a message m at the beginning of a round (over either a bilateral channel or via broadcast), the message is guaranteed to be received by the end of the round.

We note that we focus on synchronous protocols only in this work, since many of the most well-known DKG protocols are synchronous. However, we stress that our methods are equally applicable to asynchronous DKG protocols, such as the ADKG protocol of Das et al. [25].

The Adversary. We assume an adversary (also modelled as a PPT machine) who can corrupt up to $t < n/2$ out of the n parties in the network. We consider a malicious adversary that may cause corrupted parties to deviate from the protocol arbitrarily. Our adversary is *adaptive*, i.e., it chooses the corrupted parties at any time during the execution of the protocol. When it corrupts a party, we assume that it can delete or substitute any undelivered messages that this party previously sent (while being honest). We assume that the adversary has full control over the network, subject to the worst-case network delay Δ. This means that it can observe and deliver messages sent to and from honest parties far quicker than in time Δ. In particular, we assume the adversary to be *rushing*: in any synchronous round of a protocol execution, it can observe the messages of all the uncorrupted parties and then decide what messages it wants to deliver to honest parties for that round.

The Random Oracle Model (ROM). We assume the random oracle model. In this model, a hash function H is treated as an idealized random function. Concretely, H is modelled as an oracle with the following properties. The oracle internally keeps a list H for bookkeeping purposes. At the beginning, all entries of H are set to $\perp$. On input m from the domain of H, the oracle first checks whether $H[m] \neq \perp$. If so, it returns $H[m]$. Otherwise, it sets $H[m]$ to a uniformly random value in the codomain of H and then returns $H[m]$. We write q_h to denote the maximum number of allowed hash queries, i.e., the number of times the adversary may query the oracle H.

The Algebraic Group Model (AGM). The algebraic group model was introduced by Fuchsbauer, Kiltz, and Loss [29] as a model in between the generic group model (GGM) and the standard model.

[1] We remark that for most consensus protocols, it is sufficient to agree on b with some constant probability. Threshold signatures let parties agree on coins with probably close to 1.

In the AGM, all algorithms are treated as *algebraic*. This means that whenever an algorithm outputs a group element, it must also output a representation of that element relative to all of the inputs the algorithm has received up to that point. This captures the intuition that any reasonable algorithm should know how it computes its outputs from its inputs.

Definition 2.1 (Algebraic algorithm). An algorithm A is called *algebraic* (over group $\mathbb{G}$) if for all group elements $\zeta \in \mathbb{G}$ that A outputs, it additionally outputs a vector $\vec{z} = (z_0, \ldots, z_m)$ of integers such that $\zeta = \prod_i g_i^{z_i}$, where $(g_0, \ldots, g_m)$ is the list of group elements A has received so far (w.l.o.g. $g_0 = g$).

Algebraic Black-Box Reductions. An algorithm R is called a *black-box reduction* from problem $\mathbf{P}_2$ to problem $\mathbf{P}_1$ if for any algorithm A solving $\mathbf{P}_1$, algorithm R^{A} solves $\mathbf{P}_2$ with black-box/oracle access to A during its execution. If the algorithm A happens to be algebraic, then R (called an *algebraic black-box reduction*) additionally has access to the representation of the output elements of A (relative to all of the inputs A has received up to that point). We assume algebraic black-box reductions for our metareduction results. Such reductions have previously been introduced and studied, see e.g. [9, 37].

The One-More Discrete Logarithm Assumption. A mathematical hardness assumption that finds wide-ranging application in modern cryptography is the *one-more discrete logarithm (OMDL) assumption*. It is the foundation for the security analysis of identification protocols, blind signature and multi-signature schemes, such as blind Schnorr signatures. Beyond that, OMDL is also assumed for various impossibility results of certain reductions. In the following, we denote by $\mathsf{DL}_g(\cdot)$ an oracle that on input $h = g^x \in \mathbb{G}$ returns the discrete logarithm x of h to base g.

Definition 2.2 (One-More Discrete Logarithm Problem). For $n \in \mathbb{N}$ and an algorithm A, define experiment n-$\mathbf{OMDL}^{\mathsf{A}}$ as follows:

- **Setup.** For $i \in [n]$, sample $(z_1, \ldots, z_n) \leftarrow \mathbb{Z}_p^n$ and set $\xi_i := g^{z_i} \in \mathbb{G}$.
- **Online Phase.** Run A on input $(par, \xi_1, ..., \xi_n)$. A gets access to oracle $\mathsf{DL}_g(\cdot)$.
- **Output Determination.** When A returns $(z_1', ..., z_n')$, the experiment returns 1 if the following conditions are satisfied (otherwise, it returns 0):
  - $z_i' = z_i$ for all $i \in [n]$,
  - $\mathsf{DL}_g(\cdot)$ was queried at most $n - 1$ times.

We say that the one-more discrete logarithm problem of degree n is (ε, T)-hard if for all algorithms A running in time at most T, $\Pr[n\text{-}\mathbf{OMDL}^{\mathsf{A}} = 1] \leq \varepsilon$. Conversely, we say that an algorithm A (ε, T)-solves the one-more discrete logarithm problem of degree n if it runs in time at most T and $\Pr[n\text{-}\mathbf{OMDL}^{\mathsf{A}} = 1] > \varepsilon$.

3 THRESHOLD SIGNATURES

Threshold cryptography is a fundamental multiparty paradigm for enhancing the security and the availability of cryptographic schemes. It achieves this by dividing secret keys into n shares distributed across a network of parties (or servers). In (t, n)-threshold cryptosystems, secret key operations require the cooperation of at least $t + 1$ out of n parties. In this way the system remains secure against adversaries that corrupt up to t parties.

3.1 Distributed Key Generation

Distributed key generation (DKG) protocols are an essential component of threshold cryptosystems. The purpose of a DKG protocol is to distribute the shared keys of parties securely without relying on a trusted dealer. At the end of the protocol, the public key is output in the clear, whereas the secret key is kept as a virtual secret shared among all parties. The secret key is never explicitly computed, reconstructed or stored in any single location. This shared secret key can then be used later for threshold cryptosystems, such as threshold signatures or threshold encryption, without ever being explicitly reconstructed.

Definition 3.1 (Distributed Key Generation Protocol). Let Π be a protocol executed among n parties $P_1, \ldots, P_n$, where P_i outputs a *secret key share* sk_i, a vector of *public key shares* $(pk_1, \ldots, pk_n)$, a public key pk, and parties terminate upon generating output. We define the following security and correctness properties for Π:

- **Consistency:** Π is *t-consistent* if the following holds whenever at most t parties are corrupted: all honest parties output the same public key $y = g^x$ and the same vector of public key shares $(pk_1, \ldots, pk_n)$.
- **Correctness:** Π is *t-correct* if the following holds whenever at most t parties are corrupted: there exists a polynomial $f \in \mathbb{Z}_p[X]$ of degree t such that, for all $i \in [n]$, $sk_i = f(i)$ and $pk_i = g^{sk_i}$. Moreover, $pk = g^{f(0)}$.
- **Oracle-aided Algebraic Simulatability.** Π has $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-*oracle-aided algebraic simulatability* if for every algorithm A that runs in time at most T_{A} and corrupts at most t parties, there exists an algebraic simulator Sim that runs in time at most T_{Sim}, makes $k - 1$ queries to oracle $\mathsf{DL}_g(\cdot)$, and satisfies the following properties:
  - On input $\xi = g^{z_1}, \ldots, g^{z_k} \in \mathbb{G}$, Sim simulates the role of the honest parties in an execution of Π. At the end of the simulation, Sim outputs the public key $pk = g^x$. If corruptions are static, Sim gets a set of corrupted parties $C \subset \{1, \ldots, n\}$ of size at most t as an additional input.
  - On input $\xi = g^{z_1}, \ldots, g^{z_k} \in \mathbb{G}$ and for $i \in [k - 1]$, let $g_i \in \mathbb{G}$ denote the ith query to $\mathsf{DL}_g(\cdot)$. Let $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ denote the corresponding algebraic coefficients, i.e., $g_i = g^{\hat{a}_i} \cdot \prod_{j=1}^{k} (g^{z_j})^{a_{i,j}}$ and set $(\hat{a}, a_{0,1}, \ldots, a_{0,k})$ as the algebraic coefficients corresponding to pk. Then the following matrix over $\mathbb{Z}_p$ is invertible
$$L := \begin{pmatrix} a_{0,1} & a_{0,2} & \cdots & a_{0,k} \\ a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k} \end{pmatrix}.$$
    Whenever Sim completes a simulation of an execution of Π, we call L the *simulatability matrix* of Sim (for this particular simulation).
  - Denote by $\mathsf{view}_{\mathsf{A},y,\Pi}$ the view of A in an execution of Π conditioned on all honest parties outputting $pk =$

y. Similarly, denote by $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}}$ the view of A when interacting with Sim on input ξ, conditioned on Sim outputting $pk = y$. (For convenience, Sim's final output pk is omitted from $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}}$). Then, for all y and all ξ, $\mathsf{view}_{\mathsf{A},\xi,y,\mathsf{Sim}}$ and $\mathsf{view}_{\mathsf{A},y,\Pi}$ are identically distributed.

Let $k \in \mathbb{N}$ be the minimum k such that Π has $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulatability. Then we call k the *$(t, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-simulatability factor* of Π.

We say that Π has *$(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic security* if it is t-consistent, t-correct, and has $(t, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-simulatability factor k.

For informal discussions, we sometimes abbreviate our notation by ommitting T_{A} and T_{Sim} from our notation (we simply assume both A and Sim to be some PPT algorithms).

Discussion. We give a brief discussion of our security properties for distributed key generation protocols. Consistency and correctness notions are in line with what is achieved by most conventional DKG protocols.

We note that we could easily weaken the requirement of Sim being fully algebraic to Sim behaving algebraic only with respect to the elements $pk, g_1, \ldots, g_{k-1}$. (In other words, only these elements come with an algebraic representation.) All our results remain true for this weaker notion of oracle-aided algebraic security. We stress that this weaker notion is a direct generalization of the usual notion of secrecy which requires a (not necessarily algebraic) simulator Sim that on input $y \in \mathbb{G}$ perfectly simulates an execution of Π in which y is determined as the public key pk. Setting $k = 1$ in our (relaxed) definition, we see that Sim queries $\mathsf{DL}_g(\cdot)$ exactly $k-1 = 0$ times, is trivially algebraic towards the output element pk (since the input is just pk itself) and the simulatability matrix is $L = (1)$, which is trivially invertible. Therefore, one can view the degree k of Π's oracle-aided algebraic security as a measure of how far away Π is from being fully secret.

As already observed in [41], *full secrecy is not inherently required* in the context of threshold signing. This is not particularly surprising, as one might expect any reasonable signature scheme to remain unforgeable even when some information about the secret key is leaked. This observation is also the motivation behind our notion of oracle-aided algebraic simulatability. While this notion might look somewhat artificial at first glance, we will show that it is sufficient to provide unforgeability for the threshold BLS signature scheme against an adaptive adversary (in the AGM). Moreover, we prove in chapter 4 that several well-known DKG protocols including JF-DKG (proposed by Pedersen [53]) and New-DKG (proposed by Gennaro et al. [33]) have oracle-aided algebraic simulatability, also against adaptive adversaries. We stress that neither of these protocols satisfies secrecy against an adaptive adversary. In fact, it has been noted many times in the literature that JF-DKG does not even achieve full secrecy against a static adversary that corrupts a mere two parties!

Finally, we remark that we require perfect simulatability only for convenience; it is straight forward to adjust our definition so as to allow for statistical or even computationally indistinguishable simulations.

3.2 Threshold Signature Scheme

In this section, we introduce the syntax and security notions for threshold signature schemes. We remark that we focus on *non-interactive* schemes for this work.

Definition 3.2 (Non-Interactive Threshold Signature). A *non-interactive (t, n)-threshold signature scheme* is a tuple of efficient algorithms $\Sigma = (\mathsf{DKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Ver}, \mathsf{Comb})$ with the following properties:

- DKG: This is a distributed key generation protocol in the sense of Definition 3.1.
- SSign: The *share signing algorithm* is a possibly randomized algorithm that takes as input a message m and a secret key share sk_i. It outputs a signature share σ_i.
- SVer: The *signature share verification algorithm* is a deterministic algorithm that takes as input a message m, a public key share pk_i, and a signature share σ_i. It outputs 1 (accept) or 0 (reject).
- Comb: The *signature share combining algorithm* is a deterministic algorithm that takes as input the public key pk, a vector of public key shares $(pk_1, \ldots, pk_n)$, a message m, and a set $\mathcal{S}$ of $t+1$ signature shares (σ_i, i) (with corresponding indices). It outputs either a signature σ or $\bot$.
- Ver: The *signature verification algorithm* is a deterministic algorithm that takes as input a public key pk, a message m, and a signature σ. It outputs 1 (accept) or 0 (reject).

Let $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$ be a cryptographic hash function (modelled as a random oracle). We define the security of a non-interactive threshold signature scheme in the adaptive corruption setting as follows:

Definition 3.3 (Unforgeability Under Chosen Message Attack). Let $\Sigma = (\mathsf{DKG}, \mathsf{SSign}, \mathsf{SVer}, \mathsf{Ver}, \mathsf{Comb})$ be a non-interactive (t, n)-threshold signature scheme. For an algorithm A, define experiment $\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Sigma,t}$ as follows:

- **Setup.** Initialize sets $\mathcal{H} := \{1, \ldots, n\}, C := \emptyset$. Run A on input par.
- **Corruption Queries.** At any point of the experiment, A may corrupt a party P_i by submitting an index i. In this case, return the internal state of P_i and set $\mathcal{H} = \mathcal{H} \setminus \{i\}$, $C = C \cup \{i\}$. Henceforth, A controls P_i.
- **Distributed Key Generation.** Initiate an execution of DKG among parties $P_1, ..., P_n$. Denote by $(sk_1, \ldots, sk_n)$, pk, and $(pk_1, \ldots, pk_n)$ the secret and public key shares determined by DKG. ($\{sk_i\}_{i \in C}$ are known to A.)
- **Online Phase.** During this phase, A gets additional access to oracles that answer queries of the following types:
  - **Signing Queries.** When A submits a pair (i, m) for $i \in \mathcal{H}$, return $\sigma \leftarrow \mathsf{SSign}(sk_i, m)$.
  - **Random Oracle Queries.** When A submits a query m, check if $H[m] = \bot$ and if so, set $H[m] \leftarrow \mathbb{G}$. Return $H[m]$.
- **Output Determination.** When A outputs a message m^* and a signature σ^*, let $\mathcal{S} \subset \{1, \ldots, n\}$ denote the subset of parties for which A made a signing query of the form (i, m^*). Output 1 if $|C \cup \mathcal{S}| \leq t+1$ and $\mathsf{Ver}(pk, m^*, \sigma^*) = 1$. Otherwise, output 0.

We say that Σ is $(\varepsilon, T, q_h, q_s)$*-unforgeable under chosen message attacks (UF-CMA)* if for all algorithms A running in time at most T, making at most q_h random oracle queries, and making at most q_s signing queries, $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Sigma,t} = 1] \leq \varepsilon$. Conversely, we say that A $(\varepsilon, T, q_h, q_s)$-breaks unforgeability of Σ under chosen message attacks if it runs in time at most T, makes at most q_h to the random oracle, makes at most q_s queries to the signing oracle, and $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Sigma,t} = 1] > \varepsilon$

3.3 Threshold BLS Signature Scheme $\mathsf{Th\text{-}BLS_{DKG}}$

In this section, we recall Boldyreva's BLS-based threshold signature scheme. We write $\mathsf{Th\text{-}BLS_{DKG}}$ to denote the scheme when setup is done using the distributed key generation algorithm DKG.

Definition 3.4 (Threshold BLS Signature Scheme [12].). Let DKG be a distributed key generation protocol. The algorithms of the (t, n)-threshold signature scheme $\mathsf{Th\text{-}BLS_{DKG}} = (\mathsf{DKG}, \mathsf{SSign_{BLS}}, \mathsf{SVer_{BLS}}, \mathsf{Comb_{BLS}}, \mathsf{Ver_{BLS}})$ are defined as follows:

- $\mathsf{SSign_{BLS}}$: On input a secret key share $sk_i \in \mathbb{Z}_p$ and a message $m \in \{0,1\}^*$ return the signature share $\sigma_i := \mathsf{H}(m)^{sk_i} \in \mathbb{G}$.
- $\mathsf{SVer_{BLS}}$: On input a public key share $pk_i \in \mathbb{G}$, a signature share σ_i, and a message m, return 1 if $e(g, \sigma_i) = e(pk_i, \mathsf{H}(m))$ and 0 otherwise.
- $\mathsf{Comb_{BLS}}$: On input a vector of public key shares $(pk_1, ..., pk_n)$, a set $\mathcal{S}$ of $t+1$ signature shares (and corresponding indices) (σ_i, i), and a message m, run $\mathsf{SVer_{BLS}}(\sigma_i, pk_i)$ for all $i \in \mathcal{S}_0 := \{i \in [n] \mid (\sigma_i, i) \in \mathcal{S}\}$. If any of these calls returns 0, return $\perp$. Otherwise, return $\sigma = \prod_{i \in \mathcal{S}_0} \sigma_i^{L_i}$, where $L_i = \prod_{j \in \mathcal{S}_0 \setminus \{i\}} \left(\frac{j}{j-i}\right)$ denotes the ith Lagrange coefficient for the set $\mathcal{S}_0$.
- $\mathsf{Ver_{BLS}}$: On input a public key pk, a signature σ, and a message m, return 1 if $e(g, \sigma) = e(pk, \mathsf{H}(m))$ and 0 otherwise.

4 SECURITY ANALYSIS OF $\mathsf{Th\text{-}BLS_{DKG}}$

In this chapter, we analyze the security of $\mathsf{Th\text{-}BLS_{DKG}}$ in the case where DKG has (t, k)-oracle-aided algebraic security. First, we show a tight reduction to the OMDL assumption of degree k. Second, we show that there is no algebraic black-box reduction to the OMDL assumption of degree t (or lower) to the security of $\mathsf{Th\text{-}BLS_{DKG}}$ with corruption threshold t. On the other hand, we show that the trusted dealer key generation algorithm TD-DKG [see Appendix A.1] has (t, k)-oracle-aided algebraic security with $k = t + 1$. In particular, there is a reduction to the $(t + 1)$-OMDL assumption to the security of $\mathsf{Th\text{-}BLS_{TD\text{-}DKG}}$ (with corruption threshold t). Apart from that, we show oracle-aided algebraic security for JF-DKG and New-DKG, but stress that several other DKG protocols such as ADKG [25] also have this security.[2] Finally, we show that any algebraic black-box reduction from the security of $\mathsf{Th\text{-}BLS_{DKG}}$ to the $(t + 1)$-OMDL assumption loses a factor of q_s and can not possibly be improved, in the plain ROM.

For DKG with (t, k)-oracle-aided algebraic security, our first theorem asserts the security of $\mathsf{Th\text{-}BLS_{DKG}}$ (in the AGM+ROM) under the assumption that the k-OMDL problem is hard. Our proof follows the tight security proof for the (standard) BLS scheme [29, 45]. The key idea is to embed an OMDL challenge ξ in either the secret key shares or inside the random oracle queries, a choice that remains hidden from the adversary. In the former case, we simulate by using the oracle-aided algebraic simulator coming from DKG. In the latter case, we solve ξ directly from the algebraic equation that comes from the forgery (with its representation).

Theorem 4.1. *If* k-**OMDL** *is* (ε, T)*-hard in the AGM and* DKG *has* $(t, k, T', T_{\mathsf{Sim}})$*-oracle-aided algebraic security, then* $\mathsf{Th\text{-}BLS_{DKG}}$ *is* $(\varepsilon', T', q_h, q_s)$*-secure in the AGM+ROM, where*

$$\varepsilon \geq \frac{\varepsilon'}{4} - \frac{q_h^2}{4p}, \quad T \leq T' + T_{\mathsf{Sim}} + 3q_h + q_s.$$

Proof. We prove the theorem via a sequence of games. Let A be an algebraic algorithm that $(\varepsilon', T', q_h, q_s)$-breaks unforgeability of $\mathsf{Th\text{-}BLS_{DKG}}$ under chosen message attacks. W.l.o.g. we assume that A always queries the random oracle H before any signing query for the same message m. (Note that the challenger can always enforce this by making appropriate random oracle queries for itself.) Similarly, we may assume that queries to the random oracle are distinct and that A queries the random oracle H on m^* (the forgery message) before producing its forgery.

Game $\mathbf{G}_0$: This is the real game. The challenger runs DKG on behalf of the honest parties. Whenever A decides to corrupt a party P_i, the challenger faithfully returns the internal state of that party and sets $\mathcal{C} = \mathcal{C} \cup \{i\}, \mathcal{H} = \mathcal{H} \setminus \{i\}$. In addition, A gets full control over P_i. For all $i \in [n]$, let $y_i \in \mathbb{G}$ denote the public key share assigned to P_i by DKG and let x_i denote P_i's secret key share. Moreover, let $x \in \mathbb{Z}_p$ and $y = g^x$ denote the secret key and public key, respectively. Random oracle queries are answered by sampling $r_i \leftarrow \mathbb{Z}_p^*$ and returning $h_i = g^{r_i} \in \mathbb{G}$. Partial signing queries (j, m) are answered by returning $H[m]^{x_j}$. At the end of the game, A outputs a message-signature pair (m^*, σ^*).

Game $\mathbf{G}_1$: This game is identical to the game before, except that the game aborts and the adversary loses when there is a collision $H[m_1] = H[m_2]$ among distinct random oracle queries $m_1 \neq m_2$ from A. By a standard argument, $\Pr[\mathbf{G}_0^{\mathsf{A}} = 1] \leq \Pr[\mathbf{G}_1^{\mathsf{A}} = 1] + q_h^2/p$.

Let $\mathcal{V} = \mathcal{C} \cup \mathcal{S}$, where $\mathcal{S} \subset \{1, \ldots, n\}$ is the subset of parties for which A made a signing query of the form (i, m^*). As A is an algebraic adversary, at the end of $\mathbf{G}_1$ it returns a forgery σ^* on a message m^* together with a representation

$$a = (\hat{a}, a', \breve{a}_1, \ldots, \breve{a}_n, \bar{a}_1, \ldots, \bar{a}_{q_h}, \tilde{a}_{1,1}, \ldots, \tilde{a}_{1,n}, \ldots, \tilde{a}_{q_s,1}, \ldots, \tilde{a}_{q_s,n})$$

of elements in $\mathbb{Z}_p$ such that

$$\sigma^* = H[m^*] = g^{\hat{a}} \cdot y^{a'} \cdot y_1^{\breve{a}_1} \cdot \ldots \cdot y_n^{\breve{a}_n} \cdot \prod_{i=1}^{q_h} h_i^{\bar{a}_i} \cdot \prod_{i=1}^{q_s} \sigma_{i,1}^{\tilde{a}_{i,1}} \cdot \ldots \cdot \sigma_{i,n}^{\tilde{a}_{i,n}}.$$

Here, the representation is split (from left to right) into powers of the generator g, the public key $y = g^x$, the public key shares $y_j = g^{x_j}$, $j \in [n]$, all of the answers to hash queries h_i, $i \in [q_h]$, and the partial signatures $\sigma_{i,j}$, $i \in [q_s]$ and $j \in [n]$, returned by the random oracle and the partial signing oracle, respectively. Note that we have tacitly combined the other elements that are publicly communicated during the key generation phase into the term $g^{\hat{a}}$. We will clarify later why this is possible. In the following, let m_i denote the ith

[2] In fact, the proof for the oracle-aided algebraic security of ADKG easily follows from the security of JF-DKG.

query to H and let $i^* \in [q_h]$ denote the index corresponding to the forgery message m^*. Recall that we write r^* and r_i for $i \in [q_h]$ to denote the values such that $H[m^*] = g^{r^*}$ and $H[m_i] = g^{r_i}$. Having said that, the above equation is equivalent to

$$\begin{aligned} r^*x &= \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i=1}^{q_h} r_i \bar{a}_i + \sum_{i=1}^{q_s} r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n) \\ &= \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i \in Q_h} r_i \bar{a}_i + \sum_{i \in Q_s} r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n) \\ &\quad + r^*\bar{a}^* + r^*(\tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n), \qquad (\spadesuit) \end{aligned}$$

where in the last equation we split the answers to the (hash and partial signing) queries for m^* from those of the other messages, with appropriate sets Q_h and Q_s (to be precise, $Q_h = [q_h] \setminus \{i^*\}$ and $Q_s = [q_s] \setminus \{i^*\}$) and the notation $\tilde{a}^*_j = \tilde{a}^*_{i^*,j}$ for all $j \in [n]$. Since A wins $\mathbf{G}_1$, we remark that neither$\sum_{i \in Q_h} r_i \bar{a}_i$ nor $\sum_{i \in Q_s} r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n)$ may include the terms $r^*\bar{a}^*$ or $r^*(\tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n)$, respectively. [3] We define event E as the event that $x \neq \bar{a}^* + \tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n$. We have the following lemma.

Lemma 4.2. *Let* $\mathbf{G}_1$ *and* E *be as defined above. Then there exist (algebraic) algorithms* A_1 *and* A_2 *playing in game* k-**OMDL** *that run in time at most* T *such that:*

$$\Pr[k\text{-}\mathbf{OMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E],$$

$$\Pr[k\text{-}\mathbf{OMDL}^{\mathsf{A}_2} = 1] \geq \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E].$$

Moreover, $T \leq T' + T_{\mathsf{Sim}} + 3q_h + q_s$.

Proof. Let $\xi = \xi_1, \ldots, \xi_k \in \mathbb{G}$ with $\xi_i = g^{z_i}$, $i \in [k]$, be the OMDL instance. A_1 and A_2 both have access to a discrete logarithm oracle $\mathsf{DL}_g(\cdot)$ which they can query at most $k - 1$ times. Both simulate $\mathbf{G}_1$, as we now describe.

Algorithm $\mathsf{A}_1(\xi, par)$: Algorithm A_1 works as follows. Since DKG has $(t, k, T', T_{\mathsf{Sim}})$-oracle-aided algebraic security, there exists an algebraic simulator Sim that runs in time at most T_{Sim} with $(k-1)$-time access to a discrete logarithm oracle. Sim takes as input the k-OMDL instance $\xi_1 = g^{z_1}, \ldots, \xi_k = g^{z_k} \in \mathbb{G}$ and perfectly simulates an execution of DKG, where at most t parties can be corrupted.

A_1 simulates the key generation phase by running Sim on input ξ. Whenever Sim queries its discrete logarithm oracle, A_1 forwards this query to its own oracle $\mathsf{DL}_g(\cdot)$. Random oracle queries are answered by sampling $r_i \leftarrow \mathbb{Z}_p^*$ and returning $h_i = g^{r_i} \in \mathbb{G}$. A_1 aborts when there is a collision $H[m_1] = H[m_2]$ among different random oracle queries $m_1 \neq m_2$ from A. Signing queries (j, m_i) are answered by returning $h_i^{x_j}$ via the algebraic identity

$$H[m_i]^{x_j} = \left(g^{r_i}\right)^{x_j} = \left(g^{x_j}\right)^{r_i} = y_j^{r_i}.$$

Corruption queries are handled by Sim, which allows A_1 to return the internal state of up to t parties correctly. It is not hard to see that A_1's simulation of $\mathbf{G}_1$ is perfect and that A_1 can correctly answer Sim's (at most) $k-1$ oracle queries.

Suppose that A wins $\mathbf{G}_1$ and that event $\neg E$ happens, i.e. $x = \bar{a}^* + \tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n$. We note that the partial signing queries of the form (i, m^*) do not reveal any more information than if the challenger were simply handing over the corresponding private key share x_i. We thus treat these partial signing queries for m^* as corruption queries. We may also assume w.l.o.g. that $|C| = t$. Otherwise, A_1 simulates, for itself, $t - |C|$ corruption queries for random parties from the set of uncorrupted parties $\mathcal{H}$ after receiving the forgery, as if these were regular queries from A. (It does so by querying Sim.) Since A_1 knows all values $\{x_i\}_{i \in \mathcal{V}}$, it can compute the secret key x efficiently via the identity $x = \bar{a}^* + \tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n$. Let $g^{a_1}, \ldots, g^{a_{k-1}}$ denote the discrete logarithm oracle queries made by Sim. As Sim is algebraic, it also outputs a representation $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ for each of these queries, $i \in [k-1]$. Similarly, let $y = g^x$ be the public key output by Sim together with its representation $(\hat{a}_0, a_{0,1}, \ldots, a_{0,k})$. Then A_1 obtains the following system of linear equations in the variables $z_1, \ldots, z_k$:

$$\begin{aligned} x &= \hat{a}_0 + a_{0,1}z_1 + \ldots + a_{0,k}z_k \\ a_1 &= \hat{a}_1 + a_{1,1}z_1 + \ldots + a_{1,k}z_k \\ &\vdots \\ a_{k-1} &= \hat{a}_{k-1} + a_{k-1,1}z_1 + \ldots + a_{k-1,k}z_k, \end{aligned}$$

which in matrix form is equivalent to

$$\begin{pmatrix} x - \hat{a}_0 \\ a_1 - \hat{a}_1 \\ \vdots \\ a_{k-1} - \hat{a}_{k-1} \end{pmatrix} = \begin{pmatrix} a_{0,1} & a_{0,2} & \cdots & a_{0,k} \\ a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{pmatrix}.$$

By definition, the simulatability matrix of Sim is invertible and hence A_1 can efficiently compute $(z_1, \ldots, z_k)$ and solve the OMDL instance. Overall, we obtain

$$\Pr[k\text{-}\mathbf{OMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E].$$

The bound on the running time of A_1 (number of group operations and exponentiations) comes from running the simulator Sim once, one exponentiation for each random oracle query and one exponentiation for each signing query.

Algorithm $\mathsf{A}_2(\xi, par)$: Algorithm A_2 works as follows. It runs DKG correctly on behalf of the honest parties. In particular, it knows all the secret key shares x_j. Whenever the adversary A decides to corrupt a party, A_2 faithfully reveals the internal state of that party. Random oracle queries are answered by sampling $b_i, d_i \leftarrow \mathbb{Z}_p^*$ and returning $h_i = g^{r_i} = \xi_1^{b_i} g^{d_i}$, which implicitly sets $r_i = z_1 b_i + d_i$. A_2 aborts in case it detects a collision among answers in the list H. Partial signing queries (j, m_i) are answered by returning $h_i^{x_j}$. Again, it is not hard to see that A_2's simulation of $\mathbf{G}_1$ is perfect.

In case A_2 does not abort, let $\tilde{A}_i := \tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n$ for all i, where we write $\tilde{A}^*$ for $\tilde{A}_{i^*}$. Suppose that A wins $\mathbf{G}_1$ and that event E happens, i.e. $x \neq \bar{a}^* + \tilde{A}^*$. With our notation, equation $(\spadesuit)$ is equivalent to

$$\begin{aligned} z_1 b^* x + d^* x = \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i \in Q_h} d_i \bar{a}_i + \sum_{i \in Q_s} d_i \tilde{A}_i + d^*\bar{a} + d^*\tilde{A}^* \\ + z_1 \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^*\bar{a}^* + b^*\tilde{A}^* \right). \end{aligned}$$

[3] Here, we consider r^* and r_i, $i \in [n]$ as formal variables over $\mathbb{Z}_p$ rather than the concrete value that they may take. Note that it is indeed possible that for some i, $r_i \bar{a}_i = r^* \bar{a}^*$ or $r^*(\tilde{a}^*_1 x_1 + \ldots + \tilde{a}^*_n x_n) = r_i(\tilde{a}_{i,1}x_1 + \ldots + \tilde{a}_{i,n}x_n)$ when considering concrete values in $\mathbb{Z}_p$ for r^*, r_i.

With the further notations

$$B := b^*x - \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^* \bar{a}^* + b^* \tilde{A}^* \right),$$

$$D := \hat{a} + xa' + \sum_{i=1}^{n} \breve{a}_i x_i + \sum_{i \in Q_h} d_i \bar{a}_i + \sum_{i \in Q_s} d_i \tilde{A}_i + d^* \bar{a} + d^* \tilde{A}^* - d^* x,$$

this reduces to $Bz_1 = D$. Recall that we have tacitly combined the other group elements that were publicly communicated during the key generation phase into the term $g^{\hat{a}}$. This is possible because A_2 faithfully runs DKG on behalf of the honest parties and therefore has knowledge of the exponents of those elements relative to the base g and can combine them into the value $\hat{a}$. Let us now consider the case where $B = 0$. With the E defining inequality this is equivalent to

$$\begin{aligned} 0 &= b^*x - \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^* \bar{a}^* + b^* \tilde{A}^* \right) \\ \iff \quad b^*(x - \bar{a}^* - \tilde{A}^*) &= \sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i \\ \iff \quad b^* &= \left(\sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i \right) \cdot \left(x - \bar{a}^* - \tilde{A}^* \right)^{-1}. \end{aligned}$$

As already noted, $\sum_{i \in Q_h} r_i \bar{a}_i$ and $\sum_{i \in Q_s} r_i \tilde{A}_i$ do not include the terms $r^* \bar{a}^*$ and $r^* \tilde{A}^*$, respectively. Hence, $\sum_{i \in Q_h} b_i \bar{a}_i$ and $\sum_{i \in Q_s} b_i \tilde{A}_i$ do not include $b^* \bar{a}^*$ and $b^* \tilde{A}^*$, respectively.[4] As defined by the identity $H[m^*] = g^{z_1 b^* + d^*}$, b^* remains information-theoretically hidden from A. This implies that the right-hand side of the equation is statistically independent of the uniform value b^*. Therefore, with probability $1 - 1/p$ we have $B \neq 0$, and thus A_2 can compute z_1 efficiently as $z_1 := B^{-1}D$. Overall, we obtain

$$\Pr[k\text{-}\mathbf{OMDL}^{\mathsf{A}_2} = 1] \geq \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E].$$

The bound on the running time of A_2 comes from running the simulator Sim once, three operations for each random oracle query and one operation for each signing query. □

Consider algorithm B playing in k-**OMDL** as follows: B samples $i^* \leftarrow [2]$ and then internally emulates A_{i^*}. Clearly, B is an algebraic algorithm running in time at most T (the running time of $\mathsf{A}_1, \mathsf{A}_2$). An application of the law of total probability yields for $p \geq 2$,

$$\begin{aligned} \Pr[k\text{-}\mathbf{OMDL}^{\mathsf{B}} = 1] &= \sum_{i=1}^{2} \Pr[k\text{-}\mathbf{OMDL}^{\mathsf{B}} = 1 \mid i^* = 1] \cdot \Pr[i^* = 1] \\ &= \frac{1}{2} \sum_{i=1}^{2} \Pr[k\text{-}\mathbf{OMDL}^{\mathsf{A}_i} = 1] \\ &\geq \frac{1}{2} \left(1 - \frac{1}{p}\right) \left(\Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge E] + \Pr[\mathbf{G}_1^{\mathsf{A}} = 1 \wedge \neg E] \right) \\ &= \frac{1}{2} \left(1 - \frac{1}{p}\right) \Pr[\mathbf{G}_1^{\mathsf{A}} = 1] \geq \frac{1}{4} \Pr[\mathbf{G}_1^{\mathsf{A}} = 1] \\ &\geq \frac{1}{4} \cdot \Pr[\mathbf{G}_0^{\mathsf{A}} = 1] - q_h^2/p. \end{aligned}$$

□

Our next theorem asserts that the security of $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ (for DKG with (t, k)-oracle-aided algebraic security) can not be derived from the OMDL assumption of degree t, even in the AGM+ROM. In particular, the security of $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ has to rely on some stronger mathematical assumption such as OMDL of degree $t + 1$ or higher. Our proof of this impossibility result follows Coron's metareduction technique [22]. The idea behind the proof is the following. Upon receiving an OMDL challenge ξ of degree t, the metareduction M runs the reduction R providing it with (par, ξ). The discrete logarithm oracle for R is simulated by M's own oracle $\mathsf{DL}_g(\cdot)$. At the end of the key generation phase, R outputs the public key shares and the public key with their respective algebraic coefficients. The coefficient of the public key shares form a $(n \times t)$ - matrix Q over $\mathbb{Z}_p$ (disregarding the coefficients corresponding to g). Since Q has rank at most t, the metareduction can find t rows $\vec{r}_1, \ldots, \vec{r}_t$ that span the row space of Q. This eventually allows M to compute all the secret key shares (and hence the secret key x) after corrupting certain t parties. With the knowledge of x, it can forge on any message and perfectly simulate an algebraic adversary F for the reduction. We point out that in this step, it is crucial for the adversary to be allowed to corrupt parties adaptively, i.e., even after the termination of the DKG protocol. If the adversary were a static one, R would get the set of corrupted parties as an input and could take the algebraic coefficients of these parties in such a way that the corresponding vectors do not span the row space of Q. This results in M not being able to compute the secret key and the proof would fail.

Theorem 4.3. *Let* DKG *have* $(t, k, T_{\mathsf{F}_{alg}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security. Let* R *be an algebraic reduction such that for every algebraic forger* F_{alg} *that* $(\varepsilon_{\mathsf{F}_{alg}}, T_{\mathsf{F}_{alg}}, q_h, q_s)$*-breaks* $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$, $\mathsf{R}^{\mathsf{F}_{alg}}$ *is an algorithm that* $(\varepsilon_{\mathsf{R}}, T_{\mathsf{R}})$*-breaks* t-**OMDL**. *Then there exists an algorithm* M *such that* M^{R} $(\varepsilon_{\mathsf{M}}, T_{\mathsf{M}})$*-breaks* t-**OMDL** *with* $\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}}$, $T_{\mathsf{M}} \leq T_{\mathsf{R}} + T_{\mathsf{F}_{alg}}$. *In particular,* DKG *has simulatability factor* $k \geq t+1$.

Proof. Assume that R is an algebraic reduction as defined above. We will now build an efficient solver M against t-**OMDL**. Let $\xi = g^{z_1}, \ldots, g^{z_t} \in \mathbb{G}$ be the OMDL instance. M gets access to $\mathsf{DL}_g(\cdot)$ at most $t - 1$ times and his goal is to return $(z_1, \ldots, z_t)$. Algorithm M works as follows.

1. M runs the reduction R providing it with (par, ξ). The discrete logarithm oracle for R is simulated by M's own oracle $\mathsf{DL}_g(\cdot)$. As R is an algebraic reduction, at the end of the key generation phase, it returns a vector of public key shares $(g^{x_1}, \ldots, g^{x_n})$ together with a representation $(\hat{a}_i, a_{i,1}, \ldots, a_{i,t})$ for all $i \in [n]$ such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^{t} \left(g^{z_j}\right)^{a_{i,j}} = g^{\hat{a}_i} \cdot \left(g^{z_1}\right)^{a_{i,1}} \cdot \ldots \cdot \left(g^{z_t}\right)^{a_{i,t}}.$$

2. After termination of the key generation protocol, M chooses a random subset $S \subset \{1, \ldots, n\}$ of parties of order $t + 1$. W.l.o.g. we may assume $S = \{1, \ldots, t + 1\}$. Then M forms

[4] Here, we consider b^* as a formal variable over $\mathbb{Z}_p$ rather than its concrete value.

the $(t+1) \times t$ - matrix over the field $\mathbb{Z}_p$

$$Q := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,t} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,t} \\ \vdots & \vdots & & \vdots \\ a_{t+1,1} & a_{t+1,2} & \cdots & a_{t+1,t} \end{pmatrix}$$

and computes its rank efficiently via Gaussian elimination. We may assume that the rank of this matrix is t. The other cases where the rank of Q is less than t work analogously. For all $i \in [t+1]$, we let $\vec{a}_i := (a_{i,1}, \dots, a_{i,t})$ and denote by U_i the linear subspace of $\mathbb{Z}_p^t$ generated by the vectors $\vec{a}_1, \dots, \vec{a}_i$. By repeatedly computing the rank of $U_1, U_2, \dots$, M is able to identify a vector $\vec{a}_j \in \{\vec{a}_1, \dots, \vec{a}_{t+1}\}$ that is linearly dependent from the other t vectors of that set. W.l.o.g. we may assume $j = t+1$. This yields a linear system of equations $\lambda_1 \vec{a}_1 + \ldots + \lambda_t \vec{a}_t = \vec{a}_{t+1}$, where $\lambda_i \in \mathbb{Z}_p$ for all $i \in [t]$. Again via Gaussian elimination, M determines the linear coefficients $\lambda_1, \dots, \lambda_t$ of this system of equations. Note that this approach would not work in the static corruption model, as previously explained.

3. Having done this, M queries R, on behalf of a simulated algebraic forger F_{alg}^{sim}, for corruptions of the parties $P_1, \dots, P_t$ corresponding to the linearly independent vectors $\vec{a}_1, \dots, \vec{a}_t$. Reduction R returns the internal states of these parties and M additionally gets full control over them. We stress that the secret key shares of parties $P_1, \dots, P_t$ returned by R are all correct, which can be checked using the public key shares. As a result, M obtains the following system of linear equations in the variables $z_1, \dots, z_t$:

$$\begin{aligned} x_1 &= \hat{a}_1 + a_{1,1} z_1 + \ldots + a_{1,t} z_t \\ x_2 &= \hat{a}_2 + a_{2,1} z_1 + \ldots + a_{2,t} z_t \\ &\vdots \\ x_t &= \hat{a}_t + a_{t,1} z_1 + \ldots + a_{t,t} z_t, \end{aligned}$$

which in matrix form is equivalent to

$$\begin{pmatrix} x_1 - \hat{a}_1 \\ x_2 - \hat{a}_2 \\ \vdots \\ x_t - \hat{a}_t \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,t} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,t} \\ \vdots & \vdots & & \vdots \\ a_{t,1} & a_{t,2} & \cdots & a_{t,t} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_t \end{pmatrix}.$$

Multiplying the i-th row of the matrix with λ_i for all $i \in [t]$ and adding up the equations yields

$$\sum_{i=1}^{t} \lambda_i (x_i - \hat{a}_i) = \vec{a}_{t+1} \cdot \begin{pmatrix} z_1 \\ \vdots \\ z_t \end{pmatrix} = \sum_{i=1}^{t} a_{t+1,i} z_i.$$

Since $\sum_{i=1}^{t} a_{t+1,i} z_i = x_{t+1} - \hat{a}_{t+1}$, M efficiently computes x_{t+1} via the identity $x_{t+1} = \hat{a}_{t+1} + \sum_{i=1}^{t} \lambda_i (x_i - \hat{a}_i)$. Using Lagrange interpolation, M determines the secret key x.[5]

4. M picks a set $\mathcal{M} \subset \{0,1\}^*$ of q_h arbitrary messages (e.g., at random or the lexicographically first). Then it samples $m^* \leftarrow \mathcal{M}$ and $(m_1, \dots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$.
5. M queries the signing oracle, with implicit hash queries, on the messages $m_1, \dots, m_{q_s}$. Thereafter, M makes a hash query on m^* and $q_h - q_s - 1$ additional messages from the set $\mathcal{M}$. In total, M has made exactly q_h hash queries, including the implicit hash queries from signing, and exactly q_s signing queries, so that it corresponds to what the reduction expects.
6. M then tosses a biased coin $\zeta \in \{0,1\}$ that takes the value 1 with probability $\varepsilon_{\mathsf{F}_{alg}}$ and the value 0 with probability $1 - \varepsilon_{\mathsf{F}_{alg}}$. If $\zeta = 0$, then M sends $\perp$ to R. And if $\zeta = 1$, then M computes $\sigma^* = H[m^*]^x$ and submits (m^*, σ^*) as a forgery with algebraic representation $(x, 0, \dots, 0)$, so that $\sigma^* = g_0^x \cdot \prod_{i \geq 1} g_i^0$ where $(g_0, g_1, \dots, g_r)$ is the list of all group elements M has received during the execution of R and we assume w.l.o.g. $g_0 = H[m^*]$. This is done in time $T_{\mathsf{F}_{alg}}$ in order to correctly simulate an algebraic forger.
7. We see that this constitutes a valid forgery as follows. First, m^* was not queried to the signing oracle and σ^* is indeed a valid signature on m^*. Second, consider (as a thought-experiment) an unbounded algebraic forger $\mathsf{F}_{alg} = \mathsf{F}_{alg}^{unb}$ that brute-forces the secret key x from the public key g^x and outputs a valid forgery σ^* on m^* with probability $\varepsilon_{\mathsf{F}_{alg}}$. By assertion of the theorem, R has to work even against such an unbounded forger. Clearly, the view of R when interacting with F_{alg}^{sim} is indistinguishable from its view when interacting with F_{alg}^{unb}. Hence, σ^* is a valid signature on m^*. R will then return $(z_1, \dots, z_t)$ with probability ε_{R}, which M submits as its solution against t-**OMDL**.

Since F_{alg}^{sim} perfectly simulates a real algebraic forger for R, the bound on A's success probability in breaking t-**OMDL** is clear. The bound on M's time comes from running the reduction R once and simulating the forger. The bound on DKG's simulatability factor follows in combination with Theorem 4.1. □

Remark 4.1. An important implication of this theorem is that the security of $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ can not be derived from a static mathematical assumption in the AGM (and in particular not in the plain ROM). This follows from the separation results in [9].

Before we proceed with our impossibility result on the tightness of a security proof for $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ under the $(t+1)$-OMDL assumption, we focus on some concrete DKG protocols. Concretely, we show the (t,k)-oracle-aided algebraic security of several well-known DKG protocols, so that these can be safely employed into $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$. We begin our excursion with TD-DKG, which serves as a DKG protocol that represents the traditional trusted dealer scheme. In this protocol, a trusted dealer TD chooses a random polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Then, for all $i \in [n]$, it secretly sends the secret key share $sk_i = f(i)$, the vector of public key shares $(pk_1, \dots, pk_n) = (g^{f(1)}, \dots, g^{f(n)})$, and the public key $pk = g^{f(0)}$ to party P_i.

Our strategy for the proof is by building an (t,k)-oracle-aided algebraic simulator Sim with $k = t+1$ that simulates the role

[5] Note that we could also simply invert the matrix with row vectors $\vec{a}_1, \dots, \vec{a}_t$ in order to determine the values $z_1, \dots, z_t$. However, this matrix inversion approach does not work for the cases where the rank of Q is less than t. But the approach taken here simply carries over to lower-ranked matrices Q.

of the trusted dealer TD in an execution of TD-DKG. On input $t+1$ elements $\xi = \xi_0, \dots, \xi_t \in \mathbb{G}$, Sim defines the polynomial $f \in \mathbb{Z}_p[X]$ of degree t by embedding ξ_i into the ith coefficient of f for all $i \in [0, t]$. Corruption queries are answered with the oracle $\mathsf{DL}_g(\cdot)$.

Theorem 4.4. *Protocol* TD-DKG *has* $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k = t + 1$ *and* $T_{\mathsf{Sim}} \leq T_{\mathsf{A}} + 2n(t+1)$.

Proof. Let A be an adversary that runs in time at most T_{A} and corrupts at most t parties during an execution of the protocol. Clearly, TD-DKG is t-consistent and t-correct. It remains to show $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulatability for $k = t + 1$. Theorem 4.3 then implies the simulatability factor $t + 1$. For this, we build an $(t, t+1, T)$-oracle-aided algebraic simulator Sim as follows. On input $t + 1$ elements $\xi_0 = g^{z_0}, \dots, \xi_t = g^{z_t} \in \mathbb{G}$ with t-time access to an oracle $\mathsf{DL}_g(\cdot)$, Sim lets the polynomial $f = \sum_{i=0}^{t} a_i X^i \in \mathbb{Z}_p[X]$ of degree t be such that $g^{a_i} = \xi_i$ for all $i \in [0, t]$, which implicitly sets $a_i = z_i$. Then, for all $i \in [n]$, Sim computes $g^{f(i)}$ as

$$g^{f(i)} = g^{\sum_{j=0}^{t} a_j i^j} = \prod_{j=0}^{t} (g^{a_j})^{i^j} = \prod_{j=0}^{t} \xi_j^{i^j}$$

and sends the public key shares $(pk_1, \dots, pk_n) = (g^{f(1)}, \dots, g^{f(n)})$ along with the public key $pk = g^{f(0)} = \xi_0$ to party P_i. Whenever A decides to corrupt a party P_j, Sim queries $\mathsf{DL}_g(g^{f(j)})$ and returns $sk_j = f(j)$. Since A makes at most t corruption queries, Sim accesses the oracle $\mathsf{DL}_g(\cdot)$ at most t times and hence is a well-defined simulator. Let $C \subset \{1, \dots, n\}$ denote the subset of corrupted parties at the end of an execution of Sim. W.l.o.g. we may assume that $|C| = t$. By construction, the simulatability matrix of Sim is the square Vandermonde matrix $V(\dots)$ for the $t + 1$ distinct numbers in $C \cup \{0\}$, which is invertible. Finally, f is indistinguishable from a random polynomial over $\mathbb{Z}_p$ of degree t and Sim's simulation of TD-DKG is perfect. The claim on the running time is easy to verify. □

Now we turn to Pedersen's JF-DKG protocol [see Appendix A.2]. We note that the public key shares $(g^{x_1}, \dots, g^{x_n})$ are not output explicitly by JF-DKG, but can be computed from publicly available information [33]. Therefore, we may assume that these values are publicly known. The proof of the following theorem is essentially just an adaption of the preceding proof to the setting where each party P_i acts as a dealer with its own polynomial $f_i \in \mathbb{Z}_p[X]$. Concretely, we build an (t, k)-oracle-aided algebraic simulator Sim with $k = n(t + 1)$ that simulates the role of the honest parties in an execurtion of JF-DKG. On input $n(t+1)$ elements $\xi = \xi_0, \dots, \xi_{k-1} \in \mathbb{G}$, Sim defines the polynomials $f_i \in \mathbb{Z}_p[X]$ of degree t by embedding different $t + 1$ $\xi_{i_0}, \dots, \xi_{i_t}$ into the coefficients of f_i for all $i \in [n]$ such that $\bigcup_{i \in [n]} \{i_0, \dots, i_t\} = [n(t + 1) - 1]$. This means, Sim just evenly distributes ξ among all the polynomials f_i. And corruption queries are answered through the oracle $\mathsf{DL}_g(\cdot)$. By the end of the simulation, the simulatability matrix of Sim is just a block diagonal matrix with the ith block being the square Vandermonde matrix for some $t + 1$ distinct numbers. This is due to the fact that the ith block corresponds to the polynomial f_i and these f_i each hide different $\xi_I = \xi_{i_0}, \dots, \xi_{i_t}$ elements. Since this block diagonal matrix is invertible, Sim is a well-defined oracle-aided algebraic simulator. For the proof see Appendix A.4.

Theorem 4.5. *Protocol* JF-DKG *has* $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k \leq n(t + 1)$ *and* $T_{\mathsf{Sim}} \leq T_{\mathsf{A}} + 2n^2(t + 1) + n$.

The proof for Gennaro et al.'s New-DKG protocol [see Appendix A.3] is essentially the same as the preceding one for JF-DKG, since the „masking“ polynomials g_i appearing in New-DKG do not contribute to the secret key shares. For these, Sim simply honestly samples $g_i \leftarrow \mathbb{Z}_p[X]$ at random and proceeds otherwise as in the proof for JF-DKG. For the proof see Appendix A.5.

Theorem 4.6. *Protocol* New-DKG *has* $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k \leq n(t + 1)$ *and* $T_{\mathsf{Sim}} \in T_{\mathsf{A}} + O(n^3)$.

We close this work with an impossibility result on the tightness of a security proof for $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ under the $(t + 1)$-OMDL. As before, our proof follows the metareduction technique. In a typical scenario, the metareduction M rewinds the reduction R back to a previous state, with the consequence that M gains some new information from the second run of R which eventually allows M to simulate a forger to R successfully. In our case, however, the reduction has access to the $\mathsf{DL}_g(\cdot)$ oracle which M also has to simulate to R. This comes with a subtle but severe problem: after rewinding R back to a previous state, M additionally has to answer the same number of $\mathsf{DL}_g(\cdot)$ queries that R has made in its first run. This is a non-trivial or even impossible task for M, unless R has made none oracle queries in its first run. But a priori, M can not predict or control R's behaviour at all. We resolve this issue by finding a state $\mathcal{I}_{\mathsf{R}}$ of R in which R necessarily must have queried the $\mathsf{DL}_g(\cdot)$ oracle t times. This state $\mathcal{I}_{\mathsf{R}}$ will then be the state to which we rewind R later. In fact, $\mathcal{I}_{\mathsf{R}}$ will be shortly after the termination of DKG. The remainder of the proof proceeds as in the spirit of [22, 36] which equally leads to a security loss linear in the number of signing queries q_s.

Theorem 4.7. *Let* DKG *have* $(t, k, T_{\mathsf{F}}, T_{\mathsf{Sim}})$*-oracle-aided algebraic security with* $k = t + 1$. *Let* R *be an algebraic reduction such that for every forger* F *that* $(\varepsilon_{\mathsf{F}}, T_{\mathsf{F}}, q_h, q_s)$*-breaks* $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$, R^{F} *is an algorithm that* $(\varepsilon_{\mathsf{R}}, T_{\mathsf{R}})$*-breaks* $(t + 1)$-**OMDL**. *Then there exists an algorithm* M *such that* M^{R} $(\varepsilon_{\mathsf{M}}, T_{\mathsf{M}})$*-breaks* $(t + 1)$-**OMDL** *with*

$$\varepsilon_{\mathsf{M}} \geq \varepsilon_{\mathsf{R}} - \varepsilon_{\mathsf{F}} \cdot \frac{2}{e q_s}, \quad T_{\mathsf{M}} \leq 2(T_{\mathsf{R}} + T_{\mathsf{F}}).$$

In particular, a reduction from $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ *to* $(t + 1)$-**OMDL** *with a security loss less than* q_s *yields a solver* M *for* $(t + 1)$-**OMDL** *with non-negative success probability* ε_{M}.

Proof. Assume that R is an algebraic reduction as defined above. We will now build an efficient solver M against $(t + 1)$-**OMDL**. Let $\xi = g^{z_1}, \dots, g^{z_{t+1}} \in \mathbb{G}$ be the OMDL instance. M gets access to $\mathsf{DL}_g(\cdot)$ at most t times and his goal is to return $(z_1, \dots, z_{t+1})$. Algorithm M works as follows.

1. M runs the reduction R providing it with (par, ξ) and access to $\mathsf{DL}_g(\cdot)$. As R is an algebraic reduction, at the end of the key generation phase, it returns a vector of public key shares $(g^{x_1}, \dots, g^{x_n})$ together with a representation

$(\hat{a}_i, a_{i,1}, \ldots, a_{i,t+1})$ for all $i \in [n]$ such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^{t+1} \left(g^{z_j}\right)^{a_{i,j}} = g^{\hat{a}_i} \cdot \left(g^{z_1}\right)^{a_{i,1}} \cdot \ldots \cdot \left(g^{z_{t+1}}\right)^{a_{i,t+1}} .$$

2. After termination of the key generation phase, M forms the $n \times (t+1)$ - matrix over the field $\mathbb{Z}_p$

$$Q := \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,t+1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,t+1} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,t+1} \end{pmatrix}$$

and computes its rank efficiently via Gaussian elimination. Since any $t+1$ key shares determine the remaining key shares via Lagrange interpolation, this matrix has rank at most $t+1$. If the rank of Q is t or less, then M proceeds in the same way as the solver in the proof of Theorem 4.3. But this is impossible unless $(t+1)$-**OMDL** is easy. Therefore, we may assume that the rank of Q is $t+1$.

3. M samples a random subset $C \subset \{1, \ldots, n\}$ of parties of order t and queries R, on behalf of a simulated forger F_{sim}, for corruptions of these parties. Reduction R returns the internal states of these parties and M additionally gets full control over them. We stress that the secret key shares of parties $P_i \in C$ returned by R are all correct, which can be checked using the public key shares. We denote R's state up to this point by $\mathcal{I}_\mathsf{R}$. Suppose that R has queried $\mathsf{DL}_g(\cdot)$ less than t times up to this point. In that case, M aborts the reduction R and queries $\mathsf{DL}_g(g^{x_{j^*}})$ for an arbitrary $j^* \leftarrow \{1, \ldots, n\} \setminus C$. Since the matrix Q has full rank, M efficiently computes $(z_1, \ldots, z_{t+1})$ as usual. Therefore, unless $(t+1)$-**OMDL** is easy, we may assume that R has queried $\mathsf{DL}_g(\cdot)$ at least t times up to this point.

4. Let $q = \max\{q_h, 2q_s\}$. M chooses a set $\mathcal{M} \subset \{0,1\}^*$ of q_h arbitrary messages. Then it samples $i \leftarrow [q_s]$, $m^* \leftarrow \mathcal{M}$ and $(m_1, \ldots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$, which defines the two sequences of messages

$$\mathcal{M}_1 = (m_1, \ldots, m_{i-1}, m^*), \quad \mathcal{M}_2 = (m_1, \ldots, m_{q_s}).$$

5. M queries the signing oracle, with implicit hash queries, on the messages in $\mathcal{M}_1$. If R does not abort, M receives the signature list $\mathcal{S}_1 = (\sigma_1, \ldots, \sigma_{i-1}, \sigma^*)$. If any of these signatures is invalid, which can be checked using the public key shares, then M aborts.

6. We rewind R back to its previous state $\mathcal{I}_\mathsf{R}$. Now, M queries the signing oracle, with implicit hash queries, on the messages in $\mathcal{M}_2$. If R does not abort, M receives the signature list $\mathcal{S}_2 = (\sigma_1, \ldots, \sigma_{q_s})$.[6] If any of these signatures is invalid, then M aborts.

7. M makes a hash query on m^* and $q_h - q_s - 1$ additional messages from the set $\mathcal{M}$. In total, M has made exactly q_h hash queries, including the implicit hash queries from signing, and exactly q_s signing queries, so that it correspond to what the reduction expects.

8. M then tosses a biased coin $\zeta \in \{0,1\}$ that takes the value 1 with probability ε_F and the value 0 with probability $1 - \varepsilon_\mathsf{F}$. If $\zeta = 0$, then M sends $\perp$ to R. And if $\zeta = 1$, then M submits (m^*, σ^*) as a forgery. This is done in time T_F in order to correctly simulate a forger.

9. Obviously, this constitutes a valid forgery. R will then return $(z_1, \ldots, z_t)$ with probability ε_R, which M submits as its solution against t-**OMDL**.

In the following, we analyze M's success probability in breaking $(t+1)$-**OMDL** as in the spirit of [36]. We denote by $\mathcal{V}$ the set of all sequences of indices such that the corresponding signature queries are correctly answered by R in time $\leq T_\mathsf{R}$. Obviously, if $(m_1, \ldots, m_j) \in \mathcal{V}$, then also $(m_1, \ldots, m_{j-1}) \in \mathcal{V}$. Consider (as a thought-experiment) an unbounded forger $\mathsf{F} = \mathsf{F}_{unb}$ that makes hash queries to messages as F_{sim}, signature queries to the messages from $\mathcal{M}_2$, and outputs a valid forgery σ^* on m^* with probability ε_F. By assertion of the theorem, reduction R has to work even against such an unbounded forger and outputs $(z_1, \ldots, z_{t+1})$ with probability at least ε_R. After the rewind, the view of R when interacting with F_{sim} is indistinguishable from its view when interacting with F_{unb} unless $\mathcal{M}_1 \not\subseteq \mathcal{V}$ and $\mathcal{M}_2 \subseteq \mathcal{V}$. The first condition means that R does not answer all the signing queries before the rewind correctly, while the second condition means that R does answer all the signing queries after the rewind correctly. In this case, F_{sim} would not output a valid signature, while F_{unb} would output a valid signature. This argument relies on the fact that our threshold signature scheme has unique signatures. Let $\vec{z} = (z_1, \ldots, z_{t+1})$ be the solution to the OMDL instance. This discussion yields

$$\begin{aligned} &|\Pr[R^{\mathsf{F}_{sim}}(par, \xi) = \vec{z}] - \Pr[R^{\mathsf{F}_{unb}}(par, \xi) = \vec{z}]| \\ &\leq \varepsilon_\mathsf{F} \cdot \Pr[\mathcal{M}_1 \not\subseteq \mathcal{V} \wedge \mathcal{M}_2 \subseteq \mathcal{V}]. \end{aligned}$$

We have the following lemma by Coron [22], Appendix D.

Lemma 4.8. *Let $\mathcal{V}$ be a set of sequences of at most q_s integers from $\mathcal{M}$ with the property that for any $(m_1, \ldots, m_j) \in \mathcal{V}$, it is also $(m_1, \ldots, m_{j-1}) \in \mathcal{V}$. Then the following identity holds*

$$\Pr[(m_1, \ldots, m_{q_s}) \in \mathcal{V} \wedge (m_1, \ldots, m_{i-1}, m^*) \notin \mathcal{V}] \leq \frac{1}{eq_s}$$

where the probability is taken over the randomness of $i \leftarrow [q_s]$ and $(m_1, \ldots, m_{q_s}, m^*) \leftarrow \mathcal{M}^{q_s+1}$.

The probability that $m_i \neq m^*$ for all $i \in [q_s]$ is bounded by $(1 - q_s/|\mathcal{M}|)$. This allows us to use the above lemma and get

$$\Pr[\mathcal{M}_1 \not\subseteq \mathcal{V} \wedge \mathcal{M}_2 \subseteq \mathcal{V}] \leq \frac{1}{eq_s}\left(1 - \frac{q_s}{|\mathcal{M}|}\right)^{-1}.$$

Overall, we get

$$\begin{aligned} \varepsilon_\mathsf{M} &= \Pr[R^{\mathsf{F}_{sim}}(par, \xi) = \vec{z}] \\ &\geq \Pr[R^{\mathsf{F}_{unb}}(par, \xi) = \vec{z}] - \varepsilon_\mathsf{F} \cdot \frac{1}{eq_s}\left(1 - \frac{q_s}{|\mathcal{M}|}\right)^{-1} \\ &\geq \varepsilon_\mathsf{R} - \varepsilon_\mathsf{F} \cdot \frac{2}{eq_s} \end{aligned}$$

where the last inequality comes from $|\mathcal{M}| \geq 2q_s$, and therefore $\left(1 - \frac{q_s}{|\mathcal{M}|}\right)^{-1} \leq 2$. Finally, the bound on M's time comes from the

[6] Here we use that $\mathsf{Th\text{-}BLS}_{\mathsf{DKG}}$ is a signature scheme with unique signatures.

rewind, which is essentially the same as running both R and F_{sim} twice. □

REFERENCES

[1] 2022. Chia Network FAQ. (2022). https://www.chia.net/faq/.

[2] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. Communication Complexity of Byzantine Agreement, Revisited. In *38th ACM Symposium Annual on Principles of Distributed Computing*, Peter Robinson and Faith Ellen (Eds.). Association for Computing Machinery, Toronto, ON, Canada, 317–326. https://doi.org/10.1145/3293611.3331629

[3] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *FC 2019: 23rd International Conference on Financial Cryptography and Data Security (Lecture Notes in Computer Science)*, Ian Goldberg and Tyler Moore (Eds.), Vol. 11598. Springer, Heidelberg, Germany, Frigate Bay, St. Kitts and Nevis, 320–334. https://doi.org/10.1007/978-3-030-32101-7_20

[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching Consensus for Asynchronous Distributed Key Generation. In *40th ACM Symposium Annual on Principles of Distributed Computing*. Association for Computing Machinery, Portland, OR, USA, 363–373.

[5] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *38th ACM Symposium Annual on Principles of Distributed Computing*, Peter Robinson and Faith Ellen (Eds.). Association for Computing Machinery, Toronto, ON, Canada, 337–346. https://doi.org/10.1145/3293611.3331612

[6] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. 2021. Low-Bandwidth Threshold ECDSA via Pseudorandom Correlation Generators. Cryptology ePrint Archive, Report 2021/1587. (2021). https://eprint.iacr.org/2021/1587.

[7] Handan Kilinç Alper and Jeffrey Burdges. 2021. Two-Round Trip Schnorr Multi-signatures via Delinearized Witnesses. In *Advances in Cryptology – CRYPTO 2021, Part I (Lecture Notes in Computer Science)*, Tal Malkin and Chris Peikert (Eds.), Vol. 12825. Springer, Heidelberg, Germany, Virtual Event, 157–188. https://doi.org/10.1007/978-3-030-84242-0_7

[8] Diego F. Aranha, Anders P. K. Dalskov, Daniel Escudero, and Claudio Orlandi. 2021. Improved Threshold Signatures, Proactive Secret Sharing, and Input Certification from LSS Isomorphisms. In *Progress in Cryptology - LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America (Lecture Notes in Computer Science)*, Patrick Longa and Carla Ràfols (Eds.), Vol. 12912. Springer, Heidelberg, Germany, Bogotá, Colombia, 382–404. https://doi.org/10.1007/978-3-030-88238-9_19

[9] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. 2020. A Classification of Computational Assumptions in the Algebraic Group Model. In *Advances in Cryptology – CRYPTO 2020, Part II (Lecture Notes in Computer Science)*, Daniele Micciancio and Thomas Ristenpart (Eds.), Vol. 12171. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 121–151. https://doi.org/10.1007/978-3-030-56880-1_5

[10] Mihir Bellare and Gregory Neven. 2006. Multi-signatures in the plain public-Key model and a general forking lemma. In *ACM CCS 2006: 13th Conference on Computer and Communications Security*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, Alexandria, Virginia, USA, 390–399. https://doi.org/10.1145/1180405.1180453

[11] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. 2020. Asynchronous Byzantine Agreement with Subquadratic Communication. In *TCC 2020: 18th Theory of Cryptography Conference, Part I (Lecture Notes in Computer Science)*, Rafael Pass and Krzysztof Pietrzak (Eds.), Vol. 12550. Springer, Heidelberg, Germany, Durham, NC, USA, 353–380. https://doi.org/10.1007/978-3-030-64375-1_13

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

[13] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *Advances in Cryptology – ASIACRYPT 2018, Part II (Lecture Notes in Computer Science)*, Thomas Peyrin and Steven Galbraith (Eds.), Vol. 11273. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia, 435–464. https://doi.org/10.1007/978-3-030-03329-3_15

[14] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. 2017. Using Level-1 Homomorphic Encryption to Improve Threshold DSA Signatures for Bitcoin Wallet Security. In *Progress in Cryptology - LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America (Lecture Notes in Computer Science)*, Tanja Lange and Orr Dunkelman (Eds.), Vol. 11368. Springer, Heidelberg, Germany, Havana, Cuba, 352–377. https://doi.org/10.1007/978-3-030-25283-0_19

[15] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. 2002. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In *ACM CCS 2002: 9th Conference on Computer and Communications Security*, Vijayalakshmi Atluri (Ed.). ACM Press, Washington, DC, USA, 88–97. https://doi.org/10.1145/586110.586124

[16] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random oracles in constantipole: practical asynchronous Byzantine agreement using cryptography (extended abstract). In *19th ACM Symposium Annual on Principles of Distributed Computing*, Gil Neiger (Ed.). Association for Computing Machinery, Portland, OR, USA, 123–132. https://doi.org/10.1145/343477.343531

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

[18] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2021. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. Cryptology ePrint Archive, Report 2021/060. (2021). https://eprint.iacr.org/2021/060.

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

[20] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *25th Annual ACM Symposium on Theory of Computing*. ACM Press, San Diego, CA, USA, 42–51. https://doi.org/10.1145/167088.167105

[21] Shir Cohen, Idit Keidar, and Alexander Spiegelman. 2020. Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP. In *34th International Symposium on Distributed Computing (LIPICS)*, Vol. 25. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 1–25.

[22] Jean-Sébastien Coron. 2002. Optimal Security Proofs for PSS and Other Signature Schemes. In *Advances in Cryptology – EUROCRYPT 2002 (Lecture Notes in Computer Science)*, Lars R. Knudsen (Ed.), Vol. 2332. Springer, Heidelberg, Germany, Amsterdam, The Netherlands, 272–287. https://doi.org/10.1007/3-540-46035-7_18

[23] Elizabeth Crites, Chelsea Komlo, and Mary Maller. 2021. How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures. Cryptology ePrint Archive, Report 2021/1375. (2021). https://eprint.iacr.org/2021/1375.

[24] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. 2020. Securing DNSSEC Keys via Threshold ECDSA from Generic MPC. In *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II (Lecture Notes in Computer Science)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.), Vol. 12309. Springer, Heidelberg, Germany, Guildford, UK, 654–673. https://doi.org/10.1007/978-3-030-59013-0_32

[25] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *43rd IEEE Symposium on Security and Privacy (to appear)*. IEEE Computer Society Press, San Francisco, CA, USA.

[26] Yvo Desmedt. 1988. Society and Group Oriented Cryptography: A New Concept. In *Advances in Cryptology – CRYPTO'87 (Lecture Notes in Computer Science)*, Carl Pomerance (Ed.), Vol. 293. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 120–127. https://doi.org/10.1007/3-540-48184-2_8

[27] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. 2019. On the Security of Two-Round Multi-Signatures. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 1084–1101. https://doi.org/10.1109/SP.2019.00050

[28] Michael Fischer, Nancy A. Lynch, and Robert Patterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382.

[29] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. 2018. The Algebraic Group Model and its Applications. In *Advances in Cryptology – CRYPTO 2018, Part II (Lecture Notes in Computer Science)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10992. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 33–62. https://doi.org/10.1007/978-3-319-96881-0_2

[30] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. 2020. Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model. In *Advances in Cryptology – EUROCRYPT 2020, Part II (Lecture Notes in Computer Science)*, Anne Canteaut and Yuval Ishai (Eds.), Vol. 12106. Springer, Heidelberg, Germany, Zagreb, Croatia, 63–95. https://doi.org/10.1007/978-3-030-45724-2_3

[31] Rosario Gennaro and Steven Goldfeder. 2018. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 1179–1194. https://doi.org/10.1145/3243734.3243859

[32] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *ACNS 16: 14th International Conference on Applied Cryptography and Network Security (Lecture Notes in Computer Science)*, Mark Manulis, Ahmad-Reza Sadeghi,

and Steve Schneider (Eds.), Vol. 9696. Springer, Heidelberg, Germany, Guildford, UK, 156–174. https://doi.org/10.1007/978-3-319-39555-5_9

[33] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology* 20, 1 (Jan. 2007), 51–83. https://doi.org/10.1007/s00145-006-0347-3

[34] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *ACM CCS 2020: 27th Conference on Computer and Communications Security*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 803–818. https://doi.org/10.1145/3372297.3417262

[35] Stanislaw Jarecki and Anna Lysyanskaya. 2000. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. In *Advances in Cryptology – EUROCRYPT 2000 (Lecture Notes in Computer Science)*, Bart Preneel (Ed.), Vol. 1807. Springer, Heidelberg, Germany, Bruges, Belgium, 221–242. https://doi.org/10.1007/3-540-45539-6_16

[36] Saqib A. Kakvi and Eike Kiltz. 2018. Optimal Security Proofs for Full Domain Hash, Revisited. *Journal of Cryptology* 31, 1 (Jan. 2018), 276–306. https://doi.org/10.1007/s00145-017-9257-9

[37] Julia Kastner, Julian Loss, and Jiayu Xu. 2022. On pairing-free blind signature schemes in the algebraic group model. In *International Conference on Public-Key Cryptography-PKC (LNCS)*, Vol. 2. Springer, 468–497.

[38] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *ACM CCS 2020: 27th Conference on Computer and Communications Security*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 1751–1767. https://doi.org/10.1145/3372297.3423364

[39] Chelsea Komlo and Ian Goldberg. 2020. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *Selected Areas in Cryptography-SAC*. 34–65.

[40] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. 2021. Refresh When You Wake Up: Proactive Threshold Wallets with Offline Devices. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 608–625. https://doi.org/10.1109/SP40001.2021.00067

[41] Benoît Libert, Marc Joye, and Moti Yung. 2014. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *33rd ACM Symposium Annual on Principles of Distributed Computing*, Magnús M. Halldórsson and Shlomi Dolev (Eds.). Association for Computing Machinery, Paris, France, 303–312. https://doi.org/10.1145/2611462.2611498

[42] Yehuda Lindell. 2017. Fast Secure Two-Party ECDSA Signing. In *Advances in Cryptology – CRYPTO 2017, Part II (Lecture Notes in Computer Science)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10402. Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 613–644. https://doi.org/10.1007/978-3-319-63715-0_21

[43] Yehuda Lindell. 2022. Simple Three-Round Multiparty Schnorr Signing with Full Simulatability. Cryptology ePrint Archive, Report 2022/374. (March 2022).

[44] Yehuda Lindell and Ariel Nof. 2018. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 1837–1854. https://doi.org/10.1145/3243734.3243788

[45] Julian Loss. 2019. *New techniques for the modular analysis of digital signature schemes.* Ph.D. Dissertation. Ruhr University Bochum, Germany.

[46] Julian Loss and Tal Moran. 2018. Combining Asynchronous and Synchronous Byzantine Agreement: The Best of Both Worlds. Cryptology ePrint Archive, Report 2018/235. (2018). https://eprint.iacr.org/2018/235.

[47] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *39th ACM Symposium Annual on Principles of Distributed Computing*, Yuval Emek and Christian Cachin (Eds.). Association for Computing Machinery, Virtual Event, Italy, 129–138. https://doi.org/10.1145/3382734.3405707

[48] Silvio Micali. 2017. Very Simple and Efficient Byzantine Agreement. In *ITCS 2017: 8th Innovations in Theoretical Computer Science Conference*, Christos H. Papadimitriou (Ed.), Vol. 4266. LIPIcs, Berkeley, CA, USA, 6:1–6:1. https://doi.org/10.4230/LIPIcs.ITCS.2017.6

[49] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. 1999. Verifiable Random Functions. In *40th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, New York, NY, USA, 120–130. https://doi.org/10.1109/SFFCS.1999.814584

[50] Fabrice Mouhartem. 2018. Implementation of Libert et al.'s Threshold BLS Signature. (2018). https://gitlab.inria.fr/fmouhart/threshold-signature.

[51] Jonas Nick, Tim Ruffing, and Yannick Seurin. 2021. MuSig2: Simple Two-Round Schnorr Multi-signatures. In *Advances in Cryptology – CRYPTO 2021, Part I (Lecture Notes in Computer Science)*, Tal Malkin and Chris Peikert (Eds.), Vol. 12825. Springer, Heidelberg, Germany, Virtual Event, 189–221. https://doi.org/10.1007/978-3-030-84242-0_8

[52] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. 2020. MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces. In *ACM CCS 2020: 27th Conference on Computer and Communications Security*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 1717–1731. https://doi.org/10.1145/3372297.3417236

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

[54] Victor Shoup. 2000. Practical Threshold Signatures. In *Advances in Cryptology – EUROCRYPT 2000 (Lecture Notes in Computer Science)*, Bart Preneel (Ed.), Vol. 1807. Springer, Heidelberg, Germany, Bruges, Belgium, 207–220. https://doi.org/10.1007/3-540-45539-6_15

[55] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 2021. Synchronous Distributed Key Generation without Broadcasts. Cryptology ePrint Archive, Report 2021/1635. (2021). https://eprint.iacr.org/2021/1635.

[56] Stefano Tessaro and Chenzhi Zhu. 2022. Short Pairing-Free Blind Signatures with Exponential Security. In *Advances in Cryptology - EUROCRYPT (to appear)*.

A DKG PROTOCOLS

We describe the DKG protocols that we refer to directly in this work and give the proofs for the (t, k)-oracle-aided algebraic security of JF-DKG and New-DKG for $k \leq n(t+1)$ (Theorem 4.5).

A.1 TD-DKG Protocol

This is the usual key generation protocol where a trusted dealer shares a secret among n parties $P_1, \ldots, P_n$ via some secret sharing scheme. We treat this protocol as an instance of a DKG for the purpose of our results being consistent.

Protocol TD-DKG:

1. The trusted dealer TD chooses a random polynomial f of degree t with coefficients in $\mathbb{Z}_p$:
$$f = a_0 + a_1 X + \ldots + a_t X^t$$
For all $k \in [0, t]$, TD broadcasts the elements $A_i = g^{a_i}$.
2. For all $i \in [n]$, TD computes the public key share $pk_i = g^{f(i)}$. It then secretly sends $sk_i = f(i)$, the vector $(pk_1, \ldots, pk_n)$, and $y = g^{a_0}$ to party P_i.
3. Each party P_i verifies the share he received from the dealer by checking the identity
$$g^{sk_i} = \prod_{k=0}^{t} A_i^{i^k}. \tag{1}$$
If the check fails, P_i broadcasts a complaint against TD.
4. For every complaint from party P_i, the dealer reveals the share $sk_i = f(i)$ matching (1).
5. The dealer is disqualified if either
   - he received more than t complaints in Step 3, or
   - he answered a complaint in Step 4 with values that do not match (1).

   Upon TD being disqualified the protocol terminates and the DKG was unsuccessful.
6. If TD is non-disqualified, every party sets his share of the secret key as $x_i = sk_i$. The vector of public key shares is set as $(pk_1, \ldots, pk_n)$ and the public key is set as $pk = y$. The secret key x itself is not computed by any party or sent by TD, but it is equal to $x = f(0)$.

A.2 JF-DKG Protocol

This is Pedersen's traditional DKG in which it shares a secret among n parties $P_1 \ldots, P_n$ via n parallel executions of Feldman's VSS scheme. This protocol can be seen in [53] and [33].

Protocol JF-DKG:

1. Each party P_i chooses a random polynomial f_i of degree t with coefficients in $\mathbb{Z}_p$:
$$f_i = a_{i0} + a_{i1}X + \ldots + a_{it}X^t.$$
For all $k \in [0, t]$, P_i broadcasts $A_{ik} = g^{a_{ik}}$. Each P_i computes the polynomial shares $s_{ij} = f_i(j)$ for $j \in [n]$ and sends s_{ij} secretly to party P_j.
2. Each party P_j verifies the shares he received from the other parties by checking
$$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \qquad (1)$$
for all $i \in [n]$. If the check fails for an index i, P_j broadcasts a complaint against P_i.
3. Each party P_i who received a complaint from party P_j reveals the share s_{ij} matching (1). If any of the revealed shares fails this equation, P_i is disqualified. We define $Q \subset \{1, \ldots, n\}$ as the set of non-disqualified parties.
4. The public key y is computed as $y = \prod_{i \in Q} A_{i0}$. The public verification values are computed as $A_k = \prod_{i \in Q} A_{ik}$ for $k \in [t]$. Each party P_j sets his secret key share as $x_j = \sum_{i \in Q} s_{ij}$. The secret key x itself is not computed by any party, but is equal to $x = \sum_{i \in Q} a_{i0}$.

A.3 Gennaro et al.'s New-DKG Protocol

This is Gennaro et al.'s well-known DKG protocol in which the n parties $P_1, \ldots, P_n$ take part to share a secret. This protocol can be seen in [33].

Let $\mathsf{H}' : \{0, 1\}^* \to \mathbb{G}^*$ be a cryptographic hash function (modelled as a random oracle).

Protocol New-DKG:

0. This step is done only once. Set $h = H'[1]$.
1. Each party P_i performs a Pedersen-VSS of a random value z_i as a dealer:
   (a) P_i chooses two random polynomials f_i, g_i of degree t with coefficients in $\mathbb{Z}_p$:
   $$f_i(X) = a_{i0} + a_{i1}X + \ldots + a_{it}X^t,$$
   $$g_i(X) = b_{i0} + b_{i1}X + \ldots + b_{it}X^t.$$
   For all $k \in [0, t]$, P_i broadcasts the elements $C_{ik} = g^{a_{ik}} h^{b_{ik}}$. Let $z_i = a_{i0}$. Each P_i computes the polynomial shares $s_{ij} = f_i(j)$, $u_{ij} = g_i(j)$ for $j \in [n]$ and sends (s_{ij}, u_{ij}) secretly to party P_j.
   (b) Each party P_j verifies the shares he received from the other parties by checking the identity
   $$g^{s_{ij}} h^{u_{ij}} = \prod_{k=0}^{t} (C_{ik})^{j^k} \qquad (1)$$
   for all $i \in [n]$. If the check fails for an index i, P_j broadcasts a complaint against P_i.
   (c) Each party P_i who received a complaint from party P_j reveals the shares s_{ij}, u_{ij} matching (1).
   (d) Each party marks as disqualified any party that either
      - received more than t complaints in Step 1(b), or
      - answered a complaint in Step 1(c) with values that do not meet (1).
2. Each party then builds the set of non-disqualified parties $Q \subset \{1, \ldots, n\}$.
3. Each party P_j sets his share of the secret key as $x_j = \sum_{i \in Q} s_{ij}$ and the value $x'_j = \sum_{i \in Q} u_{ij}$. The distributed secret key x itself is not computed by any party, but it is equal to $x = \sum_{i \in Q} z_i$.
4. Each party P_i, $i \in Q$, exposes $y_i = g^{z_i}$ via Feldman-VSS:
   (a) Each party P_i, $i \in Q$, broadcasts the elements $A_{ik} = g^{a_{ik}}$ for all $k \in [0, t]$.
   (b) Each party P_j verifies the values broadcast by the other parties in Q by checking the identity
   $$g^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \qquad (2)$$
   for all $i \in Q$. If the check fails for an index i, P_j complains against P_i by broadcasting the values s_{ij}, u_{ij} that satisfy (1) but not (2).
   (c) For parties P_i who receive at least one valid complaint, the other parties run the reconstruction phase of Pedersen-VSS to compute z_i, f_i and A_{ik} for $k \in [0, t]$ in the clear. Let $y_i = g^{z_i}$ for $i \in Q$. The public value y is computed as $y = \prod_{i \in Q} y_i$.

Remark A.1. Note that the New-DKG protocol requires the additional element $h \in \mathbb{G}^*$ in order to run Pedersen's verifiable secret sharing (VSS). One possibility is to assume that $h = H'[1]$ is made public as part of the global parameters. Another possibility is to have h generated jointly by the parties in a preliminary phase of the protocol, e.g. by using JF-DKG.

A.4 Proof of Theorem 4.5

Proof. Let A be an adversary that runs in time at most T_{A} and corrupts at most t parties during an execution of the protocol. JF-DKG is known to be t-consist and t-correct. It remains to show $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulatability for $k = n(t + 1)$. For this, we build an appropriate algebraic simulator Sim as follows. Let $\mathcal{C}, \mathcal{H}$ denote the dynamically evolving subsets of corrupted and honest parties, respectively. Whenever A decides to corrupt a party P_i, Sim returns the internal state of P_i before setting $\mathcal{C} = \mathcal{C} \cup \{i\}$ and $\mathcal{H} = \mathcal{H} \setminus \{i\}$. Initially, we set $\mathcal{C} = \emptyset$. On input k elements $\xi_0 = g^{z_0}, \ldots, \xi_{k-1} = g^{z_{k-1}}$ with $(k-1)$-time access to an oracle $\mathsf{DL}_g(\cdot)$, for all $i \in [n]$, Sim lets the polynomial $f_i = \sum_{k=0}^{t} a_{ik}X^k \in \mathbb{Z}_p[X]$ of degree t be such that $g^{a_{ij}} = \xi_{(i-1)(t+1)+j}$ for all $j \in [0, t]$. In particular, the coefficients of f_1 are $z_0, \ldots, z_t$, those of f_2 are $z_{t+1}, \ldots, z_{2t+1}$, etc. f_i is the polynomial of party P_i.

For all $i, j \in [n]$, Sim computes $g^{f_i(j)}$ as usual. We note that as long as Sim works properly, the only elements broadcast by it are those in Step 1. Here, for all $i \in \mathcal{H}$, Sim broadcasts $A_{ik} = g^{a_{ik}}$ for $k \in [0, t]$. Whenever A decides to corrupt P_i, Sim samples a subset $\mathcal{G}_i \leftarrow \{1, \ldots, n\} \setminus \mathcal{C}$ of order $t + 1 - |\mathcal{C}|$ and queries $\mathsf{DL}_g(g^{f_i(k)})$ for all $k \in \mathcal{G}_i$. In addition, Sim queries $\mathsf{DL}_g(g^{f_j(i)})$ for all $j \in \mathcal{H} \setminus \{i\}$. (The idea here is that for each new corruption query j, Sim already knows $|\mathcal{C}|$ points on the polynomial f_j from previous corruption queries. Thus, in order to return f_j to the adversary, Sim has to query $\mathsf{DL}_g(\cdot)$ only on some other (random)

$t + 1 - |C|$ points. Additionally, in order to return P_j's internal state properly, Sim also has to query $\mathsf{DL}_g(\cdot)$ on the polynomial shares $f_j(i)$ this party P_j is supposed to get from the other (honest) parties). By drawing those values from previous corruption queries, Sim determines the polynomial f_i and returns the data f_i and $f_j(i)$ for $j \in \mathcal{H}$ before updating C and $\mathcal{H}$. We may w.l.o.g. assume that A corrupts exactly t parties. At the end of the simulation, Sim samples $i^* \leftarrow \mathcal{H}$ and queries $\mathsf{DL}_g(g^{a_{i0}})$ for all $i \in \mathcal{H} \setminus \{i^*\}$. (This is done in order to determine an algebraic representation for the public key). Subsequently, it outputs the public key $pk = \prod_{i \in Q} g^{a_{i0}}$ with representation $(\sum_{i \in Q \setminus \{i^*\}} a_{i0}, 0, \ldots, 0, 1, 0, \ldots 0)$ such that $y = g^{\sum_{i \in Q \setminus \{i^*\}} a_{i0}} \cdot (g^{a_{i^*0}})^1$.

First, we verify that Sim is well-defined by counting the total number of queries to $\mathsf{DL}_g(\cdot)$ it has made. The jth corruption query (on party P_i) yields $|\mathcal{G}_i| + |\mathcal{H} \setminus \{i\}| = t + 1 - |C| + |\mathcal{H}| - 1 = (t + 2 - j) + (n - j)$ oracle queries. Summing up these values from $j = 1$ to $j = t$ yields $\sum_{j=1}^{t}(t + 2 - j) + (n - j) = t(n + 1)$. At the end of the simulation, Sim makes $|\mathcal{H}| - 1 = n - t - 1$ additional queries, which gives a total number of $t(n + 1) + (n - t - 1) = n(t + 1) - 1$ oracle queries. Thus, Sim is a well-defined algebraic simulator.

We consider the simulatability matrix L of Sim. By construction, Sim has eventually queried $\mathsf{DL}_g(\cdot)$ on $g^{f_i(\cdot)}$ for all $i \in [n] \setminus \{i^*\}$ on some $t + 1$ distinct arguments (input values) and $g^{f_{i^*}(\cdot)}$ on some t distinct arguments $\neq 0$ with the public key corresponding to $a_{i^*0} = f_{i^*}(0)$. Thus, the queried elements along with the public key essentially correspond to $t + 1$ distinct arguments of the polynomials $f_1, \ldots, f_n$. Since the invertibility of a square matrix is invariant under row switching transformations (corresponding to the chronological order of queries made to $\mathsf{DL}_g(\cdot)$ during the execution of the DKG protocol), we may assume that we make the queries corresponding to each polynomial f_i at once. Individually, the matrix corresponding to the queries $g^{f_i(\cdot)}$ is the square Vandermonde matrix of the $t+1$ distinct input arguments, which is invertible. Overall, we find that L is a block diagonal matrix

$$\begin{pmatrix} V(f_1) & & & \\ & V(f_2) & & \\ & & \ddots & \\ & & & V(f_n) \end{pmatrix},$$

since every polynomial f_i has different coefficients: f_1 has coefficients $z_0, \ldots, z_t$ and thus $V(f_1)$ occupies the first $t+1$ rows/columns, f_2 has coefficients $z_{t+1}, \ldots, z_{2t+1}$, etc. Each block matrix is invertible and therefore L is also invertible. Finally, each f_i is indistinguishable from a random polynomial over $\mathbb{Z}_p$ of degree t and Sim's simulation of the protocol is perfect. The claim on the running time is easy to verify. □

A.5 Proof of Theorem 4.6

Proof. This is done in the same fashion as the previous proof. Again, New-DKG is known to be t-consist and t-correct. The $(t, k, T_{\mathsf{A}}, T_{\mathsf{Sim}})$-oracle-aided algebraic simulator for $k = n(t + 1)$ is described as follows. Sim works in the same way as the algebraic simulator in the previous proof, whereby honestly sampling the „masking“ polynomials $g_i \leftarrow \mathbb{Z}_p[X]$ at random, so that these are known to Sim explicitly. The further analysis works analogously. The claim on the running time is easy to verify. □