

# On the Adaptive Security of the Threshold BLS Signature Scheme

Renas Bacho   Julian Loss

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany  
{renas.bacho, 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 naive* security reduction from  $(t+1)$ -OMDL. 2) *Even in the AGM*, a strong interactive assumption is required in order to prove the scheme secure.

## 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  $\sigma$  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 unbiased coin  $b \in \{0, 1\}$  among  $n$  parties  $P_1, \dots, P_n$  as follows:

- Each party  $P_i$  (non-interactively) creates a share  $\sigma_i$  of some predetermined message  $m$  and sends  $\sigma_i$  to everybody.
- Upon collecting  $t + 1$  shares  $\sigma_i$ , a party locally reconstructs the signature  $\sigma$  for  $m$ .
- All parties can now derive the coin via  $b := \text{LSB}(\text{H}(\sigma))$ , where  $\text{H}$  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*  $\sigma$ , 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,  $\sigma$  (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  $\sigma_i$ . These combined features make (unique and non-interactive) threshold signatures a crucial tool for the design of efficient randomized consensus protocols [16, 3, 5, 47, 34]. 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, 47, 34]. 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*  $\text{Sim}$  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,  $\text{Sim}$  also gets the set of corrupted parties as input. Assuming that  $\text{Sim}$  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:  $\text{Sim}$  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 whether 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  $\sigma_i$  on message  $m$ , party  $P_i$  computes  $H(m)^{sk_i}$ . Here,  $H : \{0, 1\}^* \rightarrow \mathbb{G}$  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  $\sigma$  that verifies relative to  $pk = g^{sk}$  from  $t + 1$  shares  $\sigma_1, \dots, \sigma_{t+1}$  by interpolating  $f$  in the exponent of  $g$ . It can be verified by checking the symmetric pairing equation  $e(\sigma, g) = e(H(m), pk)$ . When dealing with adaptive corruptions, the issue is now that by sending a share  $\sigma_i = H(m)^{sk_i}$  an honest party  $P_i$  *commits* itself to its secret key  $sk_i$ . Hence, the simulator  $\text{Sim}$  must output  $sk_i$  upon  $P_i$  being adaptively corrupted. This, however, is challenging: if  $\text{Sim}$  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  $z_i = H_1^{f(i)}(m) \cdot H_2^{g(i)}(m)$  and  $r_i = H_1^{h(i)}(m) \cdot H_2^{j(i)}(m)$  and  $H_1, H_2$  are independent random oracles.  $pk$  is correspondingly set as  $(g_1, g_2) = (g_z^{f(0)} \cdot g_r^{g(0)}, g_z^{h(0)} \cdot g_r^{j(0)}) \in \hat{\mathbb{G}}^2$ , where  $g_z$  and  $g_r$  are two random generators of  $\hat{\mathbb{G}}$ . Similar as for threshold BLS, one can compute a signature  $\sigma = (z, r)$  from  $t + 1$  shares by interpolation in the exponent and verify it by checking whether  $e(z, g_z) \cdot e(r, g_r) \cdot e(H_1(m), g_1) \cdot e(H_2(m), g_2) = 1$ . 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  $\sigma_i$  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  $\text{DL}_g(\cdot)$  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  $(g^{x_1}, \dots, g^{x_k})$  and gets  $(k - 1)$ -time access to  $\text{DL}_g(\cdot)$ . It is considered successful if it can produce the values of  $x_1, \dots, x_k \in \mathbb{Z}_p$ . 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 [37, 30, 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 2 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 2. As OMDL of degree 2 is assumed to be hard, 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 combine this with a second impossibility result, which states that there is no algebraic, non-rewinding reduction from the  $q$ -discrete logarithm ( $q$ -DL) assumption to the adaptive security of the threshold BLS scheme with corruption threshold  $t$ . Recall that in the  $q$ -DL assumption, the adversary is given an instance  $(g, g^z, \dots, g^{z^q})$ . It is considered successful if it can produce the value of  $z \in \mathbb{Z}_p$ . Now Bauer et al. [9] show that in the AGM, any conceivable static assumption (and even some non-static assumptions) is implied by the  $q$ -DL assumption for a suitable degree  $q \in \mathbb{N}$ . Hence, our results show that even in the AGM, the OMDL assumption of some higher degree is both necessary and sufficient to prove security, unless one uses a rewinding reduction, and static assumptions also do not suffice to prove security. Rewinding, 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 of a certain kind (we call such reductions *naive* and define them in chapter 4.4) 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  $DL_g(\cdot)$  complicates matters significantly, as we now have to simulate  $DL_g(\cdot)$  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  $\sigma_i$ . 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  $DL_g(\cdot)$ . 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  $DL_g(\cdot)$  toward  $Sim$ , the reduction simply forwards any query  $Sim$  directs to  $DL_g(\cdot)$  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 simulatability (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 [32, 14, 42, 44, 31, 17, 24, 39, 8, 6, 23, 40, 18, 43], 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* [10, 13, 27, 52, 51, 23, 7]. 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 [53, 19, 35, 33, 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  $\mathcal{O}(\lambda n^3)$  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 [20, 15] or threshold signatures [16, 3, 5, 47, 34]. Synchronous protocols [48, 3, 2] 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  $\varrho$  that can be used to verify correct generation of  $r$ . To agree on a single bit among  $n$  parties in the  $k$ th round, a party  $P_i$  computes  $r_i = F(sk_i, k)$  along with a proof  $\varrho_i$ . It then samples  $b_i \leftarrow \{0, 1\}$  uniformly and sends  $(b_i, r_i, \varrho_i)$  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  $\varrho_i$  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.<sup>1</sup> 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  $(b_i, r_i, \varrho_i)$  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 [21, 11] 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 the model in which we will work.

### 2.1 General Notation

Let  $\lambda$  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

<sup>1</sup>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.

$g$  and endowed with a symmetric bilinear pairing  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ . 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  $V(x_1, \dots, x_r)$  for the  $r \geq 1$  numbers  $x_1, \dots, x_r \in \mathbb{Z}_p$  as

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

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  $x \leftarrow \mathbf{A}(x_1, \dots, x_n)$  we mean running algorithm  $\mathbf{A}$  on inputs  $(x_1, \dots, x_n)$  and uniformly random coins and then assigning the output to  $x$ . If  $\mathbf{A}$  has oracle access to some algorithm  $\mathbf{B}$  during its execution, we write  $x \leftarrow \mathbf{A}^{\mathbf{B}}(x_1, \dots, x_n)$ . Finally, we write  $\mathbf{G}^{\mathbf{A}}$  to denote the output of the experiment  $\mathbf{G}$  involving algorithm  $\mathbf{A}$ .

## 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  $P_1, \dots, P_n$  (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  $\Delta$ . 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  $\Delta$ . This means that it can observe and deliver messages sent to and from honest parties far quicker than in time  $\Delta$ . 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. 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  $\vec{z} = (z_0, \dots, z_m)$  of integers such that  $\zeta = \prod_i g_i^{z_i}$ , where  $(g_0, \dots, g_m)$  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  $DL_g(\cdot)$  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  $n\text{-OMDL}^A$  as follows:

- **Setup.** For  $i \in [n]$ , sample  $(z_1, \dots, z_n) \leftarrow \mathbb{Z}_p^n$  and set  $\xi_i := g^{z_i} \in \mathbb{G}$ .
- **Online Phase.** Run  $A$  on input  $(par, \xi_1, \dots, \xi_n)$ .  $A$  gets access to oracle  $DL_g(\cdot)$ .
- **Output Determination.** When  $A$  returns  $(z'_1, \dots, z'_n)$ , the experiment returns 1 if the following conditions are satisfied (otherwise, it returns 0):
  - $z'_i = z_i$  for all  $i \in [n]$ ,
  - $DL_g(\cdot)$  was queried at most  $n - 1$  times.

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

**The  $q$ -Discrete Logarithm Assumption.** A non-interactive hardness assumption that is very similar to the OMDL assumption. A special feature of it is the fact that  $q\text{-DL}$  implies every conceivable hardness assumption in the AGM (and even some non-static assumptions) [9].

**Definition 2.3** ( $q$ -Discrete Logarithm Problem). For  $q \in \mathbb{N}$  and an algorithm  $A$ , define experiment  $q\text{-DL}^A$  as follows:

- **Setup.** Sample  $z \leftarrow \mathbb{Z}_p$  and set  $\xi_i := g^{z^i} \in \mathbb{G}$  for all  $i \in [q]$ .
- **Online Phase.** Run  $A$  on input  $(par, \xi_1, \dots, \xi_q)$ .
- **Output Determination.** When  $A$  returns  $z'$ , the experiment returns 1 if  $z' = z$ , otherwise it returns 0.

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

### 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  $\Pi$  be a protocol executed among  $n$  parties  $P_1, \dots, P_n$ , where  $P_i$  outputs a *secret key share*  $sk_i$ , a vector of *public key shares*  $(pk_1, \dots, pk_n)$ , a public key  $pk$ , and parties terminate upon generating output. We define the following security and correctness properties for  $\Pi$ :

- **Consistency:**  $\Pi$  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  $(pk_1, \dots, pk_n)$ .
- **Correctness:**  $\Pi$  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,  $pk = g^{f(0)}$ .
- **Unpredictability:**  $\Pi$  is *t-unpredictable* if for every algorithm  $A$  that corrupts at most  $t$  parties and every algorithm  $A'$  that runs in the view of  $A$  after the execution of  $\Pi$ , we have  $\Pr[A'(par, y = g^x, \text{view}(A)) = x] \leq \text{negl}(\lambda)$  where  $y = g^x$  is the public key output by  $\Pi$ .
- **Oracle-aided Algebraic Simulatability.**  $\Pi$  has  $(t, k, T_A, T_{\text{Sim}})$ -*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  $\text{Sim}$  that runs in time at most  $T_{\text{Sim}}$ , makes  $k - 1$  queries to oracle  $\text{DL}_g(\cdot)$ , and satisfies the following properties:
  - On input  $\xi = g^{z_1}, \dots, g^{z_k} \in \mathbb{G}$ ,  $\text{Sim}$  simulates the role of the honest parties in an execution of  $\Pi$ . At the end of the simulation,  $\text{Sim}$  outputs the public key  $pk = g^x$ . If corruptions are static,  $\text{Sim}$  gets a set of corrupted parties  $\mathcal{C} \subset \{1, \dots, n\}$  of size at most  $t$  as an additional input.
  - On input  $\xi = g^{z_1}, \dots, g^{z_k} \in \mathbb{G}$  and for  $i \in [k - 1]$ , let  $g_i \in \mathbb{G}$  denote the  $i$ th query to  $\text{DL}_g(\cdot)$ . Let  $(\hat{a}_i, a_{i,1}, \dots, a_{i,k})$  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  $(\hat{a}, a_{0,1}, \dots, a_{0,k})$  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  $\text{Sim}$  completes a simulation of an execution of  $\Pi$ , we call  $L$  the *simulatability matrix* of  $\text{Sim}$  (for this particular simulation).



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

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

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

For informal discussions, we sometimes abbreviate our notation by omitting  $T_A$  and  $T_{\text{Sim}}$  from our notation (we simply assume both  $A$  and  $\text{Sim}$  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  $\text{Sim}$  being fully algebraic to  $\text{Sim}$  behaving algebraic only with respect to the elements  $pk, g_1, \dots, g_{k-1}$ . (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  $\text{Sim}$  that on input  $y \in \mathbb{G}$  perfectly simulates an execution of  $\Pi$  in which  $y$  is determined as the public key  $pk$ . Setting  $k = 1$  in our (relaxed) definition, we see that  $\text{Sim}$  queries  $\text{DL}_g(\cdot)$  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  $\Pi$ 's oracle-aided algebraic security as a measure of how far away  $\Pi$  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 4.3 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-inter-active*  $(t, n)$ -threshold signature scheme is a tuple of efficient algorithms  $\Sigma = (\text{DKG}, \text{SSign}, \text{SVer}, \text{Ver}, \text{Comb})$  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  $\sigma_i$ .
- **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  $\sigma_i$ . 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  $(pk_1, \dots, pk_n)$ , a message  $m$ , and a set  $\mathcal{S}$  of  $t+1$  signature shares  $(\sigma_i, i)$  (with corresponding indices). It outputs either a signature  $\sigma$  or  $\perp$ .
- **Ver:** The *signature verification algorithm* is a deterministic algorithm that takes as input a public key  $pk$ , a message  $m$ , and a signature  $\sigma$ . It outputs 1 (accept) or 0 (reject).

Let  $H : \{0, 1\}^* \rightarrow \mathbb{G}$  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  $\Sigma = (\text{DKG}, \text{SSign}, \text{SVer}, \text{Ver}, \text{Comb})$  be a non-interactive  $(t, n)$ -threshold signature scheme. For an algorithm  $A$ , define experiment  $\text{UF-CMA}_{\Sigma, t}^A$  as follows:

- **Setup.** Initialize sets  $\mathcal{H} := \{1, \dots, n\}, \mathcal{C} := \emptyset$ . 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  $\mathcal{H} = \mathcal{H} \setminus \{i\}, \mathcal{C} = \mathcal{C} \cup \{i\}$ . Henceforth,  $A$  controls  $P_i$ .
- **Distributed Key Generation.** Initiate an execution of DKG among parties  $P_1, \dots, P_n$ . Denote by  $(sk_1, \dots, sk_n), pk$ , and  $(pk_1, \dots, pk_n)$  the secret and public key shares determined by DKG. ( $\{sk_i\}_{i \in \mathcal{C}}$  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  $\sigma \leftarrow \text{SSign}(sk_i, m)$ .
  - **Random Oracle Queries.** When  $A$  submits a query  $m$ , check if  $H[m] = \perp$  and if so, set  $H[m] \leftarrow \mathbb{G}$ . Return  $H[m]$ .
- **Output Determination.** When  $A$  outputs a message  $m^*$  and a signature  $\sigma^*$ , let  $\mathcal{S} \subset \{1, \dots, n\}$  denote the subset of parties for which  $A$  made a signing query of the form  $(i, m^*)$ . Output 1 if  $|\mathcal{C} \cup \mathcal{S}| \leq t$  and  $\text{Ver}(pk, m^*, \sigma^*) = 1$ . Otherwise, output 0.

We say that  $\Sigma$  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,  $\Pr[\text{UF-CMA}_{\Sigma, t}^A = 1] \leq \varepsilon$ . Conversely, we say that  $A$   $(\varepsilon, T, q_h, q_s)$ -breaks unforgeability of  $\Sigma$  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  $\Pr[\text{UF-CMA}_{\Sigma, t}^A = 1] > \varepsilon$

### 3.3 Threshold BLS Signature Scheme Th-BLS<sub>DKG</sub>

In this section, we recall Boldyreva’s BLS-based threshold signature scheme. We write Th-BLS<sub>DKG</sub> 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  $\text{Th-BLS}_{\text{DKG}} = (\text{DKG}, \text{SSign}_{\text{BLS}}, \text{SVer}_{\text{BLS}}, \text{Comb}_{\text{BLS}}, \text{Ver}_{\text{BLS}})$  are defined as follows:

- **SSign<sub>BLS</sub>:** On input a secret key share  $sk_i \in \mathbb{Z}_p$  and a message  $m \in \{0, 1\}^*$  return the signature share  $\sigma_i := H(m)^{sk_i} \in \mathbb{G}$ .
- **SVer<sub>BLS</sub>:** On input a public key share  $pk_i \in \mathbb{G}$ , a signature share  $\sigma_i$ , and a message  $m$ , return 1 if  $e(g, \sigma_i) = e(pk_i, H(m))$  and 0 otherwise.

- $\text{Comb}_{\text{BLS}}$ : On input a vector of public key shares  $(pk_1, \dots, pk_n)$ , a set  $\mathcal{S}$  of  $t + 1$  signature shares (and corresponding indices)  $(\sigma_i, i)$ , and a message  $m$ , run  $\text{SVer}_{\text{BLS}}(\sigma_i, pk_i)$  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  $i$ th Lagrange coefficient for the set  $\mathcal{S}_0$ .
- $\text{Ver}_{\text{BLS}}$ : On input a public key  $pk$ , a signature  $\sigma$ , and a message  $m$ , return 1 if  $e(g, \sigma) = e(pk, H(m))$  and 0 otherwise.

## 4 Security Analysis of Th-BLS<sub>DKG</sub>

In this chapter, we analyze the security of Th-BLS<sub>DKG</sub> 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 from the OMDL assumption of degree 2 and from the  $q$ -DL assumption to the security of Th-BLS<sub>DKG</sub> with corruption threshold  $t$ . On the other hand, we show that the trusted dealer key generation algorithm TD-DKG [see Chapter 4.3] has  $(t, k)$ -oracle-aided algebraic security with  $k = t + 1$ . In particular, there is a reduction from the  $(t + 1)$ -OMDL assumption to the security of Th-BLS<sub>TD-DKG</sub> (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.<sup>2</sup> Finally, we show that any algebraic black-box reduction of a certain kind (which will be defined in Chapter 4.4) from the  $(t + 1)$ -OMDL assumption to the security of Th-BLS<sub>DKG</sub> loses a factor of  $q_s$  and can not possibly be improved, in the plain ROM.

### 4.1 Security proof of Th-BLS<sub>DKG</sub> in the AGM

For DKG with  $(t, k)$ -oracle-aided algebraic security, our first theorem asserts the security of Th-BLS<sub>DKG</sub> (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  $\xi$  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  $\xi$  directly from the algebraic equation that comes from the forgery (with its representation).

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

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

*Proof.* We prove the theorem via a sequence of games. Let  $\mathbf{A}$  be an algebraic algorithm that  $(\varepsilon', T', q_h, q_s)$ -breaks unforgeability of Th-BLS<sub>DKG</sub> under chosen message attacks. W.l.o.g. we assume that  $\mathbf{A}$  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  $\mathbf{A}$  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  $\mathbf{A}$  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,  $\mathbf{A}$  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

<sup>2</sup>In fact, the proof for the oracle-aided algebraic security of ADKG easily follows from the security of JF-DKG.

the end of the game, A outputs a message-signature pair  $(m^*, \sigma^*)$ .

**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,  $\Pr[\mathbf{G}_0^A = 1] \leq \Pr[\mathbf{G}_1^A = 1] + q_h^2/p$ .

Let  $\mathcal{V} = \mathcal{C} \cup \mathcal{S}$ , where  $\mathcal{S} \subset \{1, \dots, n\}$  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  $\sigma^*$  on a message  $m^*$  together with a representation

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

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

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

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  $i$ th 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

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

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  $\sum_{i \in Q_s} r_i (\tilde{a}_{i,1} x_1 + \dots + \tilde{a}_{i,n} x_n)$  may include the terms  $r^* \bar{a}^*$  or  $r^* (\tilde{a}_1^* x_1 + \dots + \tilde{a}_n^* x_n)$ , respectively.<sup>3</sup> We define event  $E$  as the event that  $x \neq \bar{a}^* + \tilde{a}_1^* x_1 + \dots + \tilde{a}_n^* x_n$ . 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:*

$$\begin{aligned} \Pr[k\text{-OMDL}^{A_1} = 1] &= \Pr[\mathbf{G}_1^A = 1 \wedge \neg E], \\ \Pr[k\text{-OMDL}^{A_2} = 1] &\geq \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^A = 1 \wedge E]. \end{aligned}$$

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

*Proof.* Let  $\xi = \xi_1, \dots, \xi_k \in \mathbb{G}$  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  $\text{DL}_g(\cdot)$  which they can query at most  $k-1$  times. Both simulate  $\mathbf{G}_1$ , as we now describe.

<sup>3</sup>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  $r^* (\tilde{a}_1^* x_1 + \dots + \tilde{a}_n^* x_n) = r_i (\tilde{a}_{i,1} x_1 + \dots + \tilde{a}_{i,n} x_n)$  when considering concrete values in  $\mathbb{Z}_p$  for  $r^*, r_i$ .

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

$A_1$  simulates the key generation phase by running  $\text{Sim}$  on input  $\xi$ . Whenever  $\text{Sim}$  queries its discrete logarithm oracle,  $A_1$  forwards this query to its own oracle  $\text{DL}_g(\cdot)$ . 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

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

Corruption queries are handled by  $\text{Sim}$ , 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  $\text{Sim}$ 's (at most)  $k-1$  oracle queries.

Suppose that  $A$  wins  $\mathbf{G}_1$  and that event  $\neg E$  happens, i.e.  $x = \bar{a}^* + \tilde{a}_1^* x_1 + \dots + \tilde{a}_n^* x_n$ . 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  $|\mathcal{C}| = t$ . Otherwise,  $A_1$  simulates, for itself,  $t - |\mathcal{C}|$  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  $\text{Sim}$ .) Since  $A_1$  knows all values  $\{x_i\}_{i \in \mathcal{V}}$ , it can compute the secret key  $x$  efficiently via the identity  $x = \bar{a}^* + \tilde{a}_1^* x_1 + \dots + \tilde{a}_n^* x_n$ . Let  $g^{a_1}, \dots, g^{a_{k-1}}$  denote the discrete logarithm oracle queries made by  $\text{Sim}$ . As  $\text{Sim}$  is algebraic, it also outputs a representation  $(\hat{a}_i, a_{i,1}, \dots, a_{i,k})$  for each of these queries,  $i \in [k-1]$ . Similarly, let  $y = g^x$  be the public key output by  $\text{Sim}$  together with its representation  $(\hat{a}_0, a_{0,1}, \dots, a_{0,k})$ . Then  $A_1$  obtains the following system of linear equations in the variables  $z_1, \dots, z_k$ :

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

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  $\text{Sim}$  is invertible and hence  $A_1$  can efficiently compute  $(z_1, \dots, z_k)$  and solve the OMDL instance. Overall, we obtain

$$\Pr[k\text{-OMDL}^{A_1} = 1] = \Pr[\mathbf{G}_1^A = 1 \wedge \neg E].$$

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

**Algorithm  $A_2(\xi, par)$ :** 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  $\tilde{A}_i := \tilde{a}_{i,1}x_1 + \dots + \tilde{a}_{i,n}x_n$  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

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

With the further notations

$$\begin{aligned} B &:= b^* x - \left( \sum_{i \in Q_h} b_i \bar{a}_i + \sum_{i \in Q_s} b_i \tilde{A}_i + b^* \bar{a}^* + b^* \tilde{A}^* \right), \\ D &:= \hat{a} + xa' + \sum_{i=1}^n \check{a}_i x_i + \sum_{i \in Q_h} d_i \bar{a}_i + \sum_{i \in Q_s} d_i \tilde{A}_i + d^* \bar{a} + d^* \tilde{A}^* - d^* x, \end{aligned}$$

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 definition of event  $E$  we get

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

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.<sup>4</sup> 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

$$\Pr[k\text{-OMDL}^{A_2} = 1] \geq \left(1 - \frac{1}{p}\right) \cdot \Pr[\mathbf{G}_1^A = 1 \wedge E].$$

The bound on the running time of  $A_2$  comes from running the simulator  $\text{Sim}$  once, three operations for each random oracle query and one operation for each signing query.  $\square$

Consider algorithm  $B$  playing in  $k\text{-OMDL}$  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  $A_1, A_2$ ). An

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

application of the law of total probability yields for  $p \geq 2$ ,

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

□

*Remark 4.1.* This result, however, does not rule out a tight reduction from the OMDL assumption of some lower degree  $< k$  to the security of  $\text{Th-BLS}_{\text{DKG}}$  (for DKG with  $(t, k)$ -oracle-aided algebraic security). Indeed, using  $\text{Th-BLS}_{\text{TD-DKG}}$  as an example (see Chapter 4.3 for a definition), we find a tight reduction from the OMDL assumption of degree  $t - r$  for any  $r \geq 0$  which is polylogarithmic in  $t$  to the security of this scheme. Note that in Chapter 4.3, Theorem 4.6 we show that TD-DKG has  $(t, k)$ -oracle-aided algebraic security with  $k = t + 1$ . The reason for the existence of a tight reduction from OMDL of some lower degree  $< k$  is based on the possibility that a potential simulator for TD-DKG might fail during the simulation. In our definition of  $(t, k)$ -oracle-aided algebraic security, we assume a perfect (oracle-aided) simulator that never fails. However, there does exist a simulator for the underlying DKG that fails during the simulation with some probability  $> 0$ , and such a "deficient" simulator may just be used by the reduction. This does not only apply to TD-DKG, but possibly also for any other DKG protocol that is employed in the threshold BLS scheme. Indeed, after our following explanation it is not hard to see that this also applies to JF-DKG and New-DKG.

In our proof of the  $(t, k)$ -oracle-aided algebraic security with  $k = t + 1$  of TD-DKG (see proof for Theorem 4.6), on input  $t + 1$  elements  $\xi = \xi_0, \dots, \xi_t \in \mathbb{G}$ , the simulator  $\text{Sim}$  chooses the polynomial  $f = \sum_{i=0}^t a_i X^i \in \mathbb{Z}_p[X]$  of degree  $t$  that determines the secret key shares by embedding  $\xi_i$  into the  $i$ th coefficient of  $f$  for all  $i \in [0, t]$ , that is  $g^{a_i} = \xi_i$ ; corruption queries are answered with the oracle  $\text{DL}_g(\cdot)$ . This simulation is perfect and in line with our security notion of oracle-aided algebraic simulatability. We now describe a "deficient" simulator  $\text{Sim}$  that works for OMDL of some lower degree  $< t + 1$  (that might fail with some probability  $> 0$ ). Take the OMDL assumption of degree  $t - r$  with  $r \geq 0$  and let  $\xi = \xi_{r+1}, \dots, \xi_t \in \mathbb{G}$  with  $\xi_i = g^{z_i}$  for all  $i \in [r + 1, t]$  (where  $z_i \in \mathbb{Z}_p$ ) be the challenge instance to be solved with  $(t - r - 1)$ -times access to an oracle  $\text{DL}_g(\cdot)$ . Firstly,  $\text{Sim}$  picks a random  $(r + 1)$ -element subset  $\tilde{S} \subset [n]$  of  $1, \dots, n$  and  $r + 1$  numbers  $s_1, \dots, s_{r+1} \in \mathbb{Z}_p$  uniformly at random. For convenience we may assume  $\tilde{S} = \{1, \dots, r + 1\}$ . Secondly,  $\text{Sim}$  chooses the polynomial  $f = \sum_{i=0}^t a_i X^i \in \mathbb{Z}_p[X]$  of degree  $t$  that determines the secret key shares such that (i)  $g^{a_i} = \xi_i$  for all  $i \in [r + 1, t]$  (which implicitly sets  $a_i = z_i$ ), and (ii)  $f(i) = s_i$  for all  $i \in [r + 1]$ . This choice determines  $f$  completely and its coefficients solely depend on the  $z_i$ . This is the case because the equations  $f(i) = s_i$  for  $i \in [r + 1]$  give the following system of linear equations in matrix form

$$\begin{pmatrix} s_1 - \sum_{i=r+1}^t a_i 1^i \\ s_2 - \sum_{i=r+1}^t a_i 2^i \\ \vdots \\ s_{r+1} - \sum_{i=r+1}^t a_i (r+1)^i \end{pmatrix} = \begin{pmatrix} 1 & 1^1 & \cdots & 1^r \\ 1 & 2^1 & \cdots & 2^r \\ \vdots & \vdots & & \vdots \\ 1 & (r+1)^1 & \cdots & (r+1)^r \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_r \end{pmatrix}$$

and the Vandermonde matrix on the right-hand side is invertible.<sup>5</sup> As a result, the polynomial  $f$  is completely described (even if implicitly). Now,  $\text{Sim}$  can work properly during a simulation (i.e., it does not fail) if and

<sup>5</sup>Note that  $\text{Sim}$  could likewise embed the instance elements  $\xi_{r+1}, \dots, \xi_t$  into some different (arbitrary)  $t - r$  coefficients of the

only if the adversary chooses a corruption set  $S \subset [n]$  that contains  $\tilde{S}$  as a subset. Otherwise,  $\text{Sim}$  would have to return at least  $t - r$  distinct points on  $f$  which it does not know a priori, so that it would ultimately know  $t - r + r + 1 = t + 1$  distinct points on  $f$ , which conversely means that  $\text{Sim}$  would have to solve the  $(t - r)$ -**OMDL** problem. On the other hand, the probability that  $\tilde{S} \subset S$  is approximately  $1/2^{r+1}$ , and therefore the simulator  $\text{Sim}$  might fail with probability  $1/2^{r+1}$ . In the other cases  $\text{Sim}$  works perfectly and is able to return the internal states of all the corrupted parties properly. Finally, the security reduction simply uses such a "deficient" simulator as described above and fails with the additional probability  $1/2^{r+1}$ . The rest of the reduction, however, proceeds exactly as in the above proof for  $(t + 1)$ -**OMDL**. Thus, for  $(t - r)$ -**OMDL**, the reduction has an additional security loss of  $1/2^{r+1}$  and therefore remains tight as long as  $r \geq 0$  is polylogarithmic in  $t$ .

## 4.2 No reduction from 2-OMDL and $q$ -DL

Our next theorem asserts that the security of  $\text{Th-BLS}_{\text{DKG}}$  (for DKG with  $(t, k)$ -oracle-aided algebraic security) can not be derived from the **OMDL** assumption of degree 2, even in the **AGM+ROM**. Furthermore, we show that such a reduction can not be derived from the  $q$ -**DL** assumption for any  $q$ . In particular, the security of  $\text{Th-BLS}_{\text{DKG}}$  has to rely on some stronger mathematical assumption such as **OMDL** of some higher degree. Our proofs for these impossibility results follow Coron's metareduction technique [22].

The idea behind the proof of the first theorem is the following. Upon receiving an **OMDL** challenge  $\xi$  of degree 2, the metareduction  $\text{M}$  runs the reduction  $\text{R}$  providing it with  $(par, \xi)$ . The discrete logarithm oracle for  $\text{R}$  is simulated by  $\text{M}$ 's own oracle  $\text{DL}_g(\cdot)$ . At the end of the key generation phase,  $\text{R}$  outputs the public key shares and the public key with their respective algebraic coefficients. The coefficients of the public key shares form an  $(n \times 2)$  - matrix  $Q$  over  $\mathbb{Z}_p$  (disregarding the coefficients corresponding to  $g$ ). By a mathematical result, we find that with overwhelming probability (to be precise, with probability at least  $1 - 2^{-t+1}$ ) a set of randomly chosen  $t$  row vectors of the matrix  $Q$  spans the row space of  $Q$ . This eventually allows  $\text{M}$  to compute all the secret key shares (and hence the secret key  $x$ ) after corrupting some  $t$  parties. With the knowledge of  $x$ , it can forge on any message and perfectly simulate an algebraic adversary  $\text{F}$  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,  $\text{R}$  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  $\text{M}$  not being able to compute the secret key and the proof would fail.

We note that the same proof strategy (for the metareduction) is also applicable to the case where the underlying hardness assumption is **OMDL** of some higher degree  $r \leq t$  instead of degree 2. The problem that arises there, however, is that the probability that a set of randomly chosen  $t$  row vectors of the matrix  $Q$  (which is now an  $(n \times r)$  - matrix over  $\mathbb{Z}_p$ ) spans the whole row space of  $Q$  is extremely difficult to determine (especially, in order to tell something about the existence of a tight reduction). We will dive more into this later.

**Theorem 4.3.** *Let DKG have  $(t, k, T_{\text{F}_{alg}}, T_{\text{Sim}})$ -oracle-aided algebraic security. Let  $\text{R}$  be an algebraic reduction such that for every algebraic forger  $\text{F}_{alg}$  that  $(\varepsilon_{\text{F}_{alg}}, T_{\text{F}_{alg}}, q_h, q_s)$ -breaks  $\text{Th-BLS}_{\text{DKG}}$ ,  $\text{R}^{\text{F}_{alg}}$  is an algorithm that  $(\varepsilon_{\text{R}}, T_{\text{R}})$ -breaks **2-OMDL**. Then there exists an algorithm  $\text{M}$  such that  $\text{M}^{\text{R}}$   $(\varepsilon_{\text{M}}, T_{\text{M}})$ -breaks **2-OMDL** with*

$$\varepsilon_{\text{M}} \geq \varepsilon_{\text{R}} - 2^{-t+1}, \quad T_{\text{M}} \leq T_{\text{R}} + T_{\text{F}_{alg}}.$$

*Proof.* Assume that  $\text{R}$  is an algebraic reduction as defined above. We will now build an efficient solver  $\text{M}$  against **2-OMDL**. Let  $\xi = g^{z_1}, g^{z_2} \in \mathbb{G}$  be the **OMDL** instance.  $\text{M}$  gets access to  $\text{DL}_g(\cdot)$  at most one time and his goal is to return  $(z_1, z_2)$ . Algorithm  $\text{M}$  works as follows.

---

polynomial  $f$ , and not necessarily into the last  $t - r$  ones. Eventually, the matrix on the right-hand side of the above equation would be a generalized Vandermonde matrix, which is also known to be invertible for pairwise distinct, positive numbers (which is the case here, since  $\tilde{S} = \{1, \dots, r + 1\}$ ). Thus, this would also result in a complete determination of  $f$ .



1. M runs the reduction R providing it with  $(par, \xi)$ . The discrete logarithm oracle for R is simulated by M's own oracle  $DL_g(\cdot)$ . 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  $(\hat{a}_i, a_{i,1}, a_{i,2})$  for all  $i \in [n]$  such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^2 (g^{z_j})^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^{z_1})^{a_{i,1}} \cdot (g^{z_2})^{a_{i,2}}.$$

2. After termination of the key generation protocol, M chooses a random subset  $S \subset \{1, \dots, n\}$  of parties of order  $t$ . W.l.o.g. we may assume  $S = \{1, \dots, t\}$ . Then M queries R, on behalf of a simulated algebraic forger  $F_{alg}^{sim}$ , for corruptions of the random  $t$  parties given by  $S$ , that is  $P_1, \dots, P_t$ . 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.
3. Afterwards, M forms the  $(n \times 2)$  - matrix over the field  $\mathbb{Z}_p$

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

and computes its rank efficiently via Gaussian elimination. We may assume that the rank of this matrix is 2. The other cases where the rank of  $Q$  is less than 2 work analogously. Consider the  $(t \times 2)$  - matrix  $Q_S$  resulting from the  $t$  rows of  $Q$  corresponding to the set  $S$ , i.e.,

$$Q_S := \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \\ a_{t,1} & a_{t,2} \end{pmatrix}.$$

We consider the following three cases regarding  $Q$ :

- (i) There are  $t + 1$  row vectors in  $Q$  that span a 0-dimensional space.
- (ii) For  $n_0 \leq t$  the number of zero row vectors in  $Q$ , there are  $t + 1 - n_0$  non-zero row vectors in  $Q$  that span a 1-dimensional space.
- (iii) For  $n_0 \leq t$  the number of zero row vectors in  $Q$ , any  $t + 1 - n_0$  non-zero row vectors in  $Q$  span a 2-dimensional space.

In each of these cases we describe how  $M$  proceeds. For all  $i \in [n]$ , we let  $\vec{a}_i := (a_{i,1}, a_{i,2})$ . For each case, we define the two events  $E$  and  $F_i$ ,  $i \in [3]$ , as follows.  $E$  is in each case the event that the reduction  $R$  is successful upon  $M$  corrupting these  $t$  parties. In case (i),  $F_1$  is the event that there are at most  $t$  row vectors in  $Q$  that are zero (obviously, the probability that this happens is zero). In case (ii),  $F_2$  is the event that all vectors  $\vec{a}_1, \dots, \vec{a}_t$  (corresponding to the corrupted parties  $P_1, \dots, P_t$ ) are zero. In case (iii),  $F_3$  is the event that the matrix  $Q_S$  has rank  $\leq 1$ . In case event  $F_i$ ,  $i \in [3]$ , happens,  $M$  fails and just aborts. But in case  $F_i$  does not happen,  $M$  is able to determine the secret key  $x$  in each of the three cases, as we will see now.

Case (i): In that case, there are  $t + 1$  row vectors, say  $\vec{a}_{i_1}, \dots, \vec{a}_{i_{t+1}}$ , such that the corresponding secret key shares are defined as  $x_j = \hat{a}_{i_j}$  for  $j \in [t + 1]$  and  $M$  efficiently computes the secret key  $x$  by Lagrangian interpolation of these  $t + 1$  points.  $M$  proceeds with step 4. Case (ii): In that case, there are  $t + 1$  row vectors in  $Q$  (and  $n_0 \geq 0$  of them being zero), say  $\vec{a}_{i_1}, \dots, \vec{a}_{i_{t+1}}$  (w.l.o.g. with

$\vec{a}_{i_1}, \dots, \vec{a}_{i_{n_0}}$  being zero), that span a 1-dimensional space, so that for all  $j \in [t]$  it is  $\vec{a}_{i_j} = \mu_j \vec{a}_{i_{t+1}}$  for some efficiently computable  $\mu_j \in \mathbb{Z}_p$ . Note that we could replace the role of  $\vec{a}_{i_{t+1}}$  with any other  $\vec{a}_{i_j}$  for  $j \in [n_0 + 1, t + 1]$ . Since  $x_i - \hat{a}_i = \vec{a}_i \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$  for all  $i \in [n]$ , we find that

$$\mu_j(x_{i_{t+1}} - \hat{a}_{i_{t+1}}) = \mu_j \vec{a}_{i_{t+1}} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \vec{a}_{i_j} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = x_{i_j} - \hat{a}_{i_j}$$

for all  $j \in [t]$ . This shows that the polynomial  $f \in \mathbb{Z}_p[X]$  of degree  $t$  that gives the secret key shares is by Lagrangian interpolation determined by only one point, namely  $x_{i_{t+1}}$  corresponding to the row vector  $\vec{a}_{i_{t+1}} \neq 0$ . Note that by the same argumentation,  $f$  is determined by any one point  $x_{i_j}$  for  $j \in [n_0 + 1, t + 1]$  corresponding to a row vector  $\vec{a}_{i_j}$  that is not zero. And in particular, any secret key share is computable via the Lagrangian interpolation formula by knowledge of  $x_{i_{t+1}}$  (or alike secret key share corresponding to a non-zero row vector). We conclude that any secret key share  $x_i = f(i)$  for  $i \in [n]$  has an expression as  $l_i + l'_i \cdot x_{i_{t+1}}$  for some efficiently computable constants  $l_i, l'_i \in \mathbb{Z}_p$ , where  $l'_i \neq 0$  if and only if the  $i$ th row vector of  $Q$  is non-zero. Hence, knowledge of any  $x_i$  corresponding to a non-zero row vector of  $Q$  determines  $f$  completely. So in case event  $F_2$  does not happen,  $\mathbf{M}$  determines  $f$  as described, computes the secret key  $x$ , and proceeds with step 4. Case (iii): In that case, assuming event  $F_3$  does not happen,  $Q_S$  has full rank 2 and therefore the vectors  $\vec{a}_1, \dots, \vec{a}_t$  give a generating set for the row space of  $Q$ . In particular,  $\vec{a}_{t+1}$  is linearly dependent from the other first  $t$  vectors  $\vec{a}_1, \dots, \vec{a}_t$ . This yields a linear system of equations  $\lambda_1 \vec{a}_1 + \dots + \lambda_t \vec{a}_t = \vec{a}_{t+1}$ , where  $\lambda_i \in \mathbb{Z}_p$  for all  $i \in [t]$ . Via Gaussian elimination,  $\mathbf{M}$  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. As a result,  $\mathbf{M}$  obtains the following system of linear equations in the variables  $z_1, z_2$ :

$$\begin{aligned} x_1 &= \hat{a}_1 + a_{1,1}z_1 + a_{1,2}z_2 \\ x_2 &= \hat{a}_2 + a_{2,1}z_1 + a_{2,2}z_2 \\ &\vdots \\ x_t &= \hat{a}_t + a_{t,1}z_1 + a_{t,2}z_2, \end{aligned}$$

which in matrix form is equivalent to

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

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

$$\sum_{i=1}^2 \lambda_i(x_i - \hat{a}_i) = \vec{a}_{t+1} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \sum_{i=1}^2 a_{t+1,i}z_i.$$

Since  $\sum_{i=1}^2 a_{t+1,i}z_i = x_{t+1} - \hat{a}_{t+1}$ ,  $\mathbf{M}$  efficiently computes  $x_{t+1}$  via the identity  $x_{t+1} = \hat{a}_{t+1} + \sum_{i=1}^2 \lambda_i(x_i - \hat{a}_i)$  by knowledge of the secret key shares  $x_1, \dots, x_t$  it got at the beginning of this step through the corruptions. Using Lagrange interpolation,  $\mathbf{M}$  determines the secret key  $x$  and proceeds with step 4.

4.  $\mathbf{M}$  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.  $\mathbf{M}$  queries the signing oracle, with implicit hash queries, on the messages  $m_1, \dots, m_{q_s}$ . Thereafter,  $\mathbf{M}$  makes a hash query on  $m^*$  and  $q_h - q_s - 1$  additional messages from the set  $\mathcal{M}$ . In total,  $\mathbf{M}$  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  $\varepsilon_{F_{alg}}$  and the value 0 with probability  $1 - \varepsilon_{F_{alg}}$ . If  $\zeta = 0$ , then  $M$  sends  $\perp$  to  $R$ . And if  $\zeta = 1$ , then  $M$  computes  $\sigma^* = H[m^*]^x$  and submits  $(m^*, \sigma^*)$  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  $T_{F_{alg}}$  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  $\sigma^*$  is indeed a valid signature on  $m^*$ . Second, consider (as a thought-experiment) an unbounded algebraic forger  $F_{alg} = F_{alg}^{unb}$  that brute-forces the secret key  $x$  from the public key  $g^x$  and outputs a valid forgery  $\sigma^*$  on  $m^*$  with probability  $\varepsilon_{F_{alg}}$  in case event  $F_i$  for an  $i \in [3]$  from step 3 does not happen. 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  $F_{alg}^{unb}$  whenever event  $F_i$  does not happen. Hence,  $\sigma^*$  is a valid signature on  $m^*$ . Additionally, this yields  $\varepsilon_M \geq \varepsilon_R - \Pr[F_i]$  for  $i \in [3]$ .  $R$  will then return  $(z_1, z_2)$ , which  $M$  submits as its solution against **2-OMDL**.

Finally, we bound the probability that  $F_i$  happens for each  $i \in [3]$ . In case (i), we clearly have  $\Pr[F_1] = 0$ . In case (ii), since there are at most  $t$  row vectors in  $Q$  being zero, the probability that  $M$ 's corruption set  $S$  contains exactly these parties (corresponding to zero row vectors in  $Q$ ) is approximately  $2^{-t}$ , i.e.,  $\Pr[F_2] \leq 2^{-t}$ . In case (iii), we have the following assertion.

Claim: The matrix  $Q_S$  has rank 2 with probability at least  $1 - 2^{-t+1}$ .

Proof: For the proof, we consider the projective 1-space  $P^1(\mathbb{F}_p)$  over the finite field  $\mathbb{F}_p = \mathbb{Z}_p$  with  $p$  elements. The space  $P^1(\mathbb{F}_p)$  consists of all 1-dimensional subspaces of  $\mathbb{F}_p^2$  (considered as a 2-dimensional vector space over  $\mathbb{F}_p$ ). By assumption, there are at most  $t - n_0$  non-zero row vectors in  $Q$  that span a 1-dimensional space. Thus, at most  $t - n_0$  non-zero row vectors of  $Q$  correspond to one particular point in  $P^1(\mathbb{F}_p)$ . In order to  $Q_S$  to have rank less than 2, the row vectors corresponding to the (randomly chosen) corruption set  $S = \{1, \dots, t\}$  would precisely have to be the  $n_0$  zero row vectors and remaining  $t - n_0$  (non-zero) vectors that correspond to one particular point in the projective space, that is these  $t - n_0$  vectors would span a 1-dimensional space. Otherwise, there would be at least two row vectors that correspond to different points in the projective space and would therefore span a 2-dimensional space. The probability of this happening is at most  $\frac{1}{2^{n_0}} \cdot \frac{2}{2^{t-n_0}} = \frac{1}{2^{t-1}}$ . Therefore,  $Q_S$  has rank 2 with probability  $\geq 1 - 2^{-t+1}$ .  $\blacksquare$  As a result, we get  $\Pr[F_3] \leq 2^{-t+1}$ . Overall, the bound on  $M$ 's success probability in breaking **2-OMDL** is given by  $\varepsilon_M \geq \varepsilon_R - \max_{i \in [3]} \{\Pr[F_i]\} \geq \varepsilon_R - 2^{-t+1}$ . The bound on  $M$ 's time comes from running the reduction  $R$  once and simulating the forger.  $\square$

In fact, the exact same proof strategy of the metareduction  $M$  can also be applied to the case where the underlying hardness assumption is OMDL of some higher degree  $r \leq t$  instead of degree 2. The main difference to the former case 2-OMDL is that the probability that the matrix  $Q_S$  (which is now a  $(t \times r)$ -matrix over  $\mathbb{Z}_p$ ) has full rank  $r$  is no longer  $1 - 2^{-t+1}$ , but another expression that we believe to depend on  $r$ . Based on that very probability, it is possible to extend our result as follows. For this, we first formulate the underlying mathematical problem and our conjecture on the solution to it.

**Conjecture 4.3.1.** *As usual let  $t < n/2$ . Let  $Q$  be an  $(n \times r)$ -matrix over  $\mathbb{Z}_p$  with  $1 \leq r \leq t$  such that any set of  $t + 1$  row vectors of  $Q$  generates a space of dimension  $r$ . Furthermore, let  $S \subset [n]$  be a  $t$ -element subset of  $\{1, \dots, n\}$  chosen uniformly at random, and let  $Q_S$  be the  $(t \times r)$ -matrix over  $\mathbb{Z}_p$  formed by the row vectors of  $Q$  given by the index set  $S$ . Then the matrix  $Q_S$  has full rank  $r$  with probability at least  $1 - 2^{-t+r-1}$ .*

The case  $r = 1$  is clear, and the case  $r = 2$  was treated in the preceding proof. For our purposes,  $Q$  is the matrix given by the algebraic coefficients of the public key shares as defined in the proof of Theorem

4.3. Furthermore, we note that the condition that any set of  $t + 1$  row vectors of  $Q$  generates a space of dimension  $r$  is satisfied if the reduction  $R$  is generic, assuming the matrix  $Q$  is of full rank (which can be assumed from the beginning w.l.o.g., as the lower rank cases work analogously). We will prove this in the Appendix A.4. Assuming Conjecture 4.3.1 to be true, we extend our result as follows.

**Theorem 4.4.** *Assume Conjecture 4.3.1 to be true. Let DKG have  $(t, k, T_{F_{alg}}, T_{Sim})$ -oracle-aided algebraic security. Let  $R$  be an algebraic reduction such that for every algebraic forger  $F_{alg}$  that  $(\varepsilon_{F_{alg}}, T_{F_{alg}}, q_h, q_s)$ -breaks  $\text{Th-BLS}_{\text{DKG}}$ ,  $R^{F_{alg}}$  is an algorithm that  $(\varepsilon_R, T_R)$ -breaks  $r$ -OMDL (with  $1 \leq r \leq t$  as in Conjecture 4.3.1). Then there exists an algorithm  $M$  such that  $M^R$   $(\varepsilon_M, T_M)$ -breaks  $r$ -OMDL with*

$$\varepsilon_M \geq \varepsilon_R - 2^{-t+r-1}, \quad T_M \leq T_R + T_{F_{alg}}.$$

For our next impossibility result regarding the existence of a security reduction from  $q$ -DL, the metareduction uses a very similar strategy as in the previous proof. The only difference is in how the metareduction  $M$  determines the secret key in step 2 of its strategy. Essentially, after some  $t$  corruptions,  $M$  obtains a non-trivial polynomial equation over  $\mathbb{Z}_p$  in the variable  $z$  (the sought-for solution to the given  $q$ -DL instance  $g^z, \dots, g^{z^q} \in \mathbb{G}$ ), which it can solve efficiently by standard techniques.

**Theorem 4.5.** *Let DKG have  $(t, k, T_{F_{alg}}, T_{Sim})$ -oracle-aided algebraic security. Let  $R$  be an algebraic reduction such that for every algebraic forger  $F_{alg}$  that  $(\varepsilon_{F_{alg}}, T_{F_{alg}}, q_h, q_s)$ -breaks  $\text{Th-BLS}_{\text{DKG}}$ ,  $R^{F_{alg}}$  is an algorithm that  $(\varepsilon_R, T_R)$ -breaks  $q$ -DL. Then there exists an algorithm  $M$  such that  $M^R$   $(\varepsilon_M, T_M)$ -breaks  $q$ -DL with*

$$\varepsilon_M \geq \varepsilon_R - 2^{-t}, \quad T_M \leq T_R + T_{F_{alg}}.$$

*Proof.* Assume that  $R$  is an algebraic reduction as defined above. We will now build an efficient solver  $M$  against  $q$ -DL. Let  $\xi = g^z, \dots, g^{z^q} \in \mathbb{G}$  be the  $q$ -DL instance.  $M$  his goal is to return  $z \in \mathbb{Z}_p$ . Algorithm  $M$  works as follows.

1.  $M$  runs the reduction  $R$  providing it with  $(par, \xi)$ . 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  $(\hat{a}_i, a_{i,1}, \dots, a_{i,q})$  for all  $i \in [n]$  such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^q (g^{z^j})^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^z)^{a_{i,1}} \cdot \dots \cdot (g^{z^q})^{a_{i,q}}.$$

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

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

For all  $i \in [n]$ , we let  $\vec{a}_i := (a_{i,1}, \dots, a_{i,q})$ . We now consider the following three cases:

- (i) There are at least  $t + 1$  row vectors in  $Q$  identical to the zero vector.
- (ii) There are at most  $t - 1$  row vectors in  $Q$  identical to the zero vector.
- (iii) There are exactly  $t$  row vectors in  $Q$  identical to the zero vector.

In each of these cases we describe how  $M$  proceeds. Case (i): In that case, there are  $t + 1$  rows, say  $\vec{a}_1, \dots, \vec{a}_{t+1}$ , such that the corresponding secret key shares are defined as  $x_j = \hat{a}_j$  for  $j \in [t + 1]$  and  $M$  efficiently computes the secret key  $x$  by Lagrangian interpolation of these  $t + 1$  points.  $M$  proceeds with step 3. Case (ii): In that case,  $M$  queries  $R$ , on behalf of a simulated algebraic forger  $F_{alg}^{sim}$ , for corruptions of some random  $t$  parties, say  $P_1, \dots, P_t$ . 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. Since there are at most  $t - 1$  rows in  $Q$  identical to the zero vector, one of the rows  $\vec{a}_1, \dots, \vec{a}_t$  (corresponding to the corrupted parties  $P_1, \dots, P_t$ ) is non-zero (w.l.o.g.  $\vec{a}_1 \neq 0$ ). As a result,  $M$  obtains the following polynomial equation over  $\mathbb{Z}_p$  in the variable  $z$ :  $x_1 = \hat{a}_1 + a_{1,1}z + a_{1,2}z^2 + \dots + a_{1,q}z^q \in \mathbb{Z}_p[z]$ . By standard techniques,  $M$  efficiently computes  $z$  and is done. It aborts the reduction and outputs  $z$  as the solution of the  $q$ -DL instance. Case (iii): In that case,  $M$  queries  $R$ , on behalf of a simulated algebraic forger  $F_{alg}^{sim}$ , for corruptions of some random  $t$  parties, say  $P_1, \dots, P_t$ . Reduction  $R$  returns the internal states of these parties and  $M$  additionally gets full control over them. We define the following two events:  $E$  is the event that the reduction  $R$  is successful upon  $M$  corrupting these  $t$  parties, and  $F$  is the event that not all vectors  $\vec{a}_1, \dots, \vec{a}_t$  (corresponding to the corrupted parties  $P_1, \dots, P_t$ ) are zero. In case event  $F$  happens, proceed as in case (ii). Now, the reduction is successful with probability  $\varepsilon_R$  if the algebraic forger is successful. That means,  $\varepsilon_R = \Pr[E]$  (conditioned on the algebraic forger  $F_{alg}$  being successful). Conditioned on the algebraic forger being successful, we obtain the following identities:

$$\begin{aligned} \varepsilon_R &= \Pr[E] = \Pr[E \wedge F] + \Pr[E \wedge \neg F] \\ &\leq \Pr[E \wedge F] + \Pr[E \mid \neg F] \Pr[\neg F] \\ &\leq \Pr[E \wedge F] + \Pr[\neg F] \leq \Pr[E \wedge F] + 2^{-t}. \end{aligned}$$

It follows that  $\Pr[E \wedge F] \geq \varepsilon_R - 2^{-t}$  and thus  $\varepsilon_M \geq \varepsilon_R - 2^{-t}$ .

3.  $M$  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  $(m_1, \dots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$ .
4.  $M$  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.
5.  $M$  then tosses a biased coin  $\zeta \in \{0, 1\}$  that takes the value 1 with probability  $\varepsilon_{F_{alg}}$  and the value 0 with probability  $1 - \varepsilon_{F_{alg}}$ . If  $\zeta = 0$ , then  $M$  sends  $\perp$  to  $R$ . And if  $\zeta = 1$ , then  $M$  computes  $\sigma^* = H[m^*]^x$  and submits  $(m^*, \sigma^*)$  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  $T_{F_{alg}}$  in order to correctly simulate an algebraic forger.
6. We see that this constitutes a valid forgery as follows. First,  $m^*$  was not queried to the signing oracle and  $\sigma^*$  is indeed a valid signature on  $m^*$ . Second, consider (as a thought-experiment) an unbounded algebraic forger  $F_{alg} = F_{alg}^{unb}$  that brute-forces the secret key  $x$  from the public key  $g^x$  and outputs a valid forgery  $\sigma^*$  on  $m^*$  with probability  $\varepsilon_{F_{alg}}$ . 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,  $\sigma^*$  is a valid signature on  $m^*$ .  $R$  will then return  $z$  with probability  $\varepsilon_R$ , which  $M$  submits as its solution against  $q$ -DL.

Since  $F_{alg}^{sim}$  perfectly simulates a real algebraic forger for  $R$ , the bound on  $A$ 's success probability in breaking  $q$ -DL is clear. The bound on  $M$ 's time comes from running the reduction  $R$  once and simulating the forger.  $\square$

*Remark 4.2.* An important implication of this theorem is that the security of Th-BLS<sub>DKG</sub> can not be derived from a static mathematical assumption in the AGM (and in particular not in the plain ROM). This follows from the fact that  $q$ -DL implies every conceivable hardness assumption in the AGM (and even some non-static assumptions) [9]. If there was a reduction from such a static assumption, this would mean that a successful forgery would lead to the breaking of this underlying assumption. And this would in turn lead to the breaking of  $q$ -DL for some  $q$ . Thus, there would ultimately exist a reduction from  $q$ -DL for some  $q$  to the security of Th-BLS<sub>DKG</sub>, which stands in direct contradiction to Theorem 4.5.

### 4.3 Security of some DKG protocols

Before we proceed with our impossibility result on the tightness of a security proof for  $\text{Th-BLS}_{\text{DKG}}$  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  $\text{Th-BLS}_{\text{DKG}}$ . 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  $\text{Sim}$  with  $k = t + 1$  that simulates the role of the trusted dealer TD in an execution of TD-DKG. On input  $t + 1$  elements  $\xi = \xi_0, \dots, \xi_t \in \mathbb{G}$ ,  $\text{Sim}$  defines the polynomial  $f \in \mathbb{Z}_p[X]$  of degree  $t$  by embedding  $\xi_i$  into the  $i$ th coefficient of  $f$  for all  $i \in [0, t]$ . Corruption queries are answered with the oracle  $\text{DL}_g(\cdot)$ .

**Theorem 4.6.** *Protocol TD-DKG has  $(t, k, T_A, T_{\text{Sim}})$ -oracle-aided algebraic security with  $k = t + 1$  and  $T_{\text{Sim}} \leq T_A + 2n(t + 1)$ .*

*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  $(t, k, T_A, T_{\text{Sim}})$ -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  $\text{Sim}$  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  $\text{DL}_g(\cdot)$ ,  $\text{Sim}$  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  $i \in [n]$ ,  $\text{Sim}$  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  $P_j$ ,  $\text{Sim}$  queries  $\text{DL}_g(g^{f(j)})$  and returns  $sk_j = f(j)$ . Since  $A$  makes at most  $t$  corruption queries,  $\text{Sim}$  accesses the oracle  $\text{DL}_g(\cdot)$  at most  $t$  times and hence is a well-defined simulator. Let  $\mathcal{C} \subset \{1, \dots, n\}$  denote the subset of corrupted parties at the end of an execution of  $\text{Sim}$ . W.l.o.g. we may assume that  $|\mathcal{C}| = t$ . By construction, the simulatability matrix of  $\text{Sim}$  is the square Vandermonde matrix  $V(\dots)$  for the  $t + 1$  distinct numbers in  $\mathcal{C} \cup \{0\}$ , which is invertible. Finally,  $f$  is indistinguishable from a random polynomial over  $\mathbb{Z}_p$  of degree  $t$  and  $\text{Sim}$ 's simulation of TD-DKG is perfect. The claim on the running time is easy to verify.  $\square$

Now we turn to Pedersen's JF-DKG protocol. 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  $\text{Sim}$  with  $k = n(t + 1)$  that simulates the role of the honest parties in an execution of JF-DKG. On input  $n(t + 1)$  elements  $\xi = \xi_0, \dots, \xi_{k-1} \in \mathbb{G}$ ,  $\text{Sim}$  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,  $\text{Sim}$  just evenly distributes  $\xi$  among all the polynomials  $f_i$ . And corruption queries are answered through the oracle  $\text{DL}_g(\cdot)$ . By the end of the simulation, the simulatability matrix of  $\text{Sim}$  is just a block diagonal matrix with the  $i$ th block being the square Vandermonde matrix for some  $t + 1$  distinct numbers. This is due to the fact that the  $i$ th 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,  $\text{Sim}$  is a well-defined oracle-aided algebraic simulator.

**Theorem 4.7.** *Protocol JF-DKG has  $(t, k, T_A, T_{\text{Sim}})$ -oracle-aided algebraic security with  $k \leq n(t + 1)$  and  $T_{\text{Sim}} \leq T_A + 2n^2(t + 1) + n$ .*

*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  $t$ -consistent and  $t$ -correct. It remains to show  $(t, k, T_A, T_{\text{Sim}})$ -oracle-aided algebraic simulatability for  $k = n(t + 1)$ . For this, we build an appropriate algebraic simulator  $\text{Sim}$  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  $P_i$ ,  $\text{Sim}$  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  $\xi_0 = g^{z_0}, \dots, \xi_{k-1} = g^{z_{k-1}}$  with  $(k - 1)$ -time access to an oracle  $\text{DL}_g(\cdot)$ , for all  $i \in [n]$ ,  $\text{Sim}$  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  $z_0, \dots, z_t$ , those of  $f_2$  are  $z_{t+1}, \dots, z_{2t+1}$ , etc.  $f_i$  is the polynomial of party  $P_i$ .

For all  $i, j \in [n]$ ,  $\text{Sim}$  computes  $g^{f_i(j)}$  as usual. We note that as long as  $\text{Sim}$  works properly, the only elements broadcast by it are those in Step 1. Here, for all  $i \in \mathcal{H}$ ,  $\text{Sim}$  broadcasts  $A_{ik} = g^{a_{ik}}$  for  $k \in [0, t]$ . Whenever  $A$  decides to corrupt  $P_i$ ,  $\text{Sim}$  samples a subset  $\mathcal{G}_i \leftarrow \{1, \dots, n\} \setminus \mathcal{C}$  of order  $t + 1 - |\mathcal{C}|$  and queries  $\text{DL}_g(g^{f_i(k)})$  for all  $k \in \mathcal{G}_i$ . In addition,  $\text{Sim}$  queries  $\text{DL}_g(g^{f_j(i)})$  for all  $j \in \mathcal{H} \setminus \{i\}$ . (The idea here is that for each new corruption query  $j$ ,  $\text{Sim}$  already knows  $|\mathcal{C}|$  points on the polynomial  $f_j$  from previous corruption queries. Thus, in order to return  $f_j$  to the adversary,  $\text{Sim}$  has to query  $\text{DL}_g(\cdot)$  only on some other (random)  $t + 1 - |\mathcal{C}|$  points. Additionally, in order to return  $P_j$ 's internal state properly,  $\text{Sim}$  also has to query  $\text{DL}_g(\cdot)$  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,  $\text{Sim}$  determines the polynomial  $f_i$  and returns the data  $f_i$  and  $f_j(i)$  for  $j \in \mathcal{H}$  before updating  $\mathcal{C}$  and  $\mathcal{H}$ . We may w.l.o.g. assume that  $A$  corrupts exactly  $t$  parties. At the end of the simulation,  $\text{Sim}$  samples  $i^* \leftarrow \mathcal{H}$  and queries  $\text{DL}_g(g^{a_{i^*0}})$  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  $pk = \prod_{i \in \mathcal{Q}} g^{a_{i^*0}}$  with representation  $(\sum_{i \in \mathcal{Q} \setminus \{i^*\}} a_{i^*0}, 0, \dots, 0, 1, 0, \dots, 0)$  such that it is  $y = g^{\sum_{i \in \mathcal{Q} \setminus \{i^*\}} a_{i^*0}} \cdot (g^{a_{i^*0}})^1$ .

First, we verify that  $\text{Sim}$  is well-defined by counting the total number of queries to  $\text{DL}_g(\cdot)$  it has made. The  $j$ th corruption query (on party  $P_i$ ) yields  $|\mathcal{G}_i| + |\mathcal{H} \setminus \{i\}| = t + 1 - |\mathcal{C}| + |\mathcal{H}| - 1 = (t + 2 - j) + (n - j)$  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,  $\text{Sim}$  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,  $\text{Sim}$  is a well-defined algebraic simulator.

We consider the simulatability matrix  $L$  of  $\text{Sim}$ . By construction,  $\text{Sim}$  has eventually queried  $\text{DL}_g(\cdot)$  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  $f_1, \dots, f_n$ . Since the invertibility of a square matrix is invariant under row switching transformations (corresponding to the chronological order of queries made to  $\text{DL}_g(\cdot)$  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  $z_0, \dots, z_t$  and thus  $V(f_1)$  occupies the first  $t + 1$  rows/columns,  $f_2$  has coefficients  $z_{t+1}, \dots, z_{2t+1}$ , 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  $\text{Sim}$ 's simulation of the protocol is perfect. The claim on the running time is easy to verify.  $\square$

The proof for Gennaro et al.'s New-DKG protocol 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,  $\text{Sim}$  simply honestly samples  $g_i \leftarrow \mathbb{Z}_p[X]$  at random and proceeds otherwise as in the proof for JF-DKG.

**Theorem 4.8.** *Protocol New-DKG has  $(t, k, T_A, T_{\text{Sim}})$ -oracle-aided algebraic security with  $k \leq n(t + 1)$  and  $T_{\text{Sim}} \in T_A + \mathcal{O}(n^3)$ .*

*Proof.* This is done in the same fashion as the previous proof. Again, New-DKG is known to be  $t$ -consistent and  $t$ -correct. The  $(t, k, T_A, T_{\text{Sim}})$ -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.  $\square$

#### 4.4 Non-tightness of naive reduction from $(t + 1)$ -OMDL in ROM

We close this work with an impossibility result on the tightness of a security proof for  $\text{Th-BLS}_{\text{DKG}}$  under the  $(t + 1)$ -OMDL in the plain random oracle model, assuming the reduction is *naive*. We give a proper definition of what this means in our context.

**Definition 4.1** (Naive Security Reduction). Let  $R$  be an algebraic reduction from the hardness of  $(t + 1)$ -OMDL to the security of  $\text{Th-BLS}_{\text{DKG}}$ . Then we say that  $R$  is **naive** if the  $n \times (t + 1)$  - matrix  $Q$  formed by the algebraic coefficients of the public key shares satisfies the following condition: any set of  $t + 1$  row vectors of  $Q$  generates a space of dimension  $t + 1$ .

As we now argue, naive reductions are an interesting class of reductions that appear difficult to bypass. As we show in the Appendix A.4, any generic reduction that embeds the  $(t + 1)$ -element OMDL instance „fully“ into the polynomial  $f \in \mathbb{Z}_p[X]$  of degree  $t$  that determines the secret key shares is indeed naive. By „fully“, we simply mean that the matrix  $Q$  has full rank  $t + 1$ . Furthermore, we emphasize that our tight reduction from Theorem 4.1 is an example of a naive reduction and that we do not know of a tighter reduction strategy. An example of a non-naive reduction would be one that embeds only  $t$  or fewer elements of the  $(t + 1)$ -element OMDL instance into the polynomial  $f$ . However, it is unclear in what way this would be helpful.

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  $\text{DL}_g(\cdot)$  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  $\text{DL}_g(\cdot)$  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  $\mathcal{I}_R$  of  $R$  in which  $R$  necessarily must have queried the  $\text{DL}_g(\cdot)$  oracle  $t$  times. This state  $\mathcal{I}_R$  will then be the state to which we rewind  $R$  later. In fact,  $\mathcal{I}_R$  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 4.9.** *Let DKG have  $(t, k, T_F, T_{\text{Sim}})$ -oracle-aided algebraic security with  $k = t + 1$ . Let  $R$  be a naive reduction as defined above such that for every forger  $F$  that  $(\varepsilon_F, T_F, q_h, q_s)$ -breaks  $\text{Th-BLS}_{\text{DKG}}$ ,  $R^F$  is an algorithm that  $(\varepsilon_R, T_R)$ -breaks  $(t + 1)$ -OMDL. Then there exists an algorithm  $M$  such that  $M^R$   $(\varepsilon_M, T_M)$ -breaks  $(t + 1)$ -OMDL with*

$$\varepsilon_M \geq \varepsilon_R - \varepsilon_F \cdot \frac{2}{eq_s}, \quad T_M \leq 2(T_R + T_F).$$

*In particular, a naive reduction from  $\text{Th-BLS}_{\text{DKG}}$  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  $\varepsilon_M$ .*

*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  $\text{DL}_g(\cdot)$  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  $(par, \xi)$  and access to  $DL_g(\cdot)$ . 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  $(\hat{a}_i, a_{i,1}, \dots, a_{i,t+1})$  for all  $i \in [n]$  such that

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

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

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

Since  $R$  is a naive reduction, any set of  $t+1$  row vectors of  $Q$  generates a space of dimension  $t+1$ . It follows that any set of  $t$  row vectors of  $Q$  generates a space of dimension  $t$ , i.e., any  $t$  row vectors of  $Q$  are linearly independent. Now,  $M$  samples a random subset  $\mathcal{C} \subset \{1, \dots, n\}$  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  $P_i \in \mathcal{C}$  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  $\mathcal{I}_R$ . Suppose that  $R$  has queried  $DL_g(\cdot)$  less than  $t$  times up to this point. In that case,  $M$  aborts the reduction  $R$  and queries  $DL_g(g^{x_{j^*}})$  for an arbitrary  $j^* \leftarrow \{1, \dots, n\} \setminus \mathcal{C}$ . Since any  $t+1$  row vectors of  $Q$  span the whole row space of  $Q$ ,  $M$  efficiently computes  $(z_1, \dots, z_{t+1})$  as usual. Therefore, unless  $(t+1)$ -**OMDL** is easy, we may assume that  $R$  has queried  $DL_g(\cdot)$  at least  $t$  times up to this point.

3. 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  $(m_1, \dots, m_{q_s}) \leftarrow (\mathcal{M} \setminus \{m^*\})^{q_s}$ , which defines the two sequences of messages

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

4.  $M$  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  $\mathcal{S}_1 = (\sigma_1, \dots, \sigma_{i-1}, \sigma^*)$ . If any of these signatures is invalid, which can be checked using the public key shares, then  $M$  aborts.
5. We rewind  $R$  back to its previous state  $\mathcal{I}_R$ . 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  $\mathcal{S}_2 = (\sigma_1, \dots, \sigma_{q_s})$ .<sup>6</sup> If any of these signatures is invalid, then  $M$  aborts.
6.  $M$  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.
7.  $M$  then tosses a biased coin  $\zeta \in \{0, 1\}$  that takes the value 1 with probability  $\varepsilon_F$  and the value 0 with probability  $1 - \varepsilon_F$ . If  $\zeta = 0$ , then  $M$  sends  $\perp$  to  $R$ . And if  $\zeta = 1$ , then  $M$  submits  $(m^*, \sigma^*)$  as a forgery. This is done in time  $T_F$  in order to correctly simulate a forger.
8. Obviously, this constitutes a valid forgery.  $R$  will then return  $(z_1, \dots, z_t)$  with probability  $\varepsilon_R$ , which  $M$  submits as its solution against  $t$ -**OMDL**.

---

<sup>6</sup>Here we use that  $\text{Th-BLSDKG}$  is a signature scheme with unique signatures.

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

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

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

**Lemma 4.10.** *Let  $\mathcal{V}$  be a set of sequences of at most  $q_s$  integers from  $\mathcal{M}$  with the property that for any  $(m_1, \dots, m_j) \in \mathcal{V}$ , it is also  $(m_1, \dots, m_{j-1}) \in \mathcal{V}$ . Then the following identity holds*

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

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

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

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

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  $\mathsf{M}$ 's time comes from the rewind, which is essentially the same as running both  $\mathsf{R}$  and  $\mathsf{F}_{\text{sim}}$  twice.  $\square$

## Acknowledgement

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507237585.

## References

- [1] 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. Communication complexity of byzantine agreement, revisited. In Peter Robinson and Faith Ellen, editors, *38th ACM Symposium Annual on Principles of Distributed Computing*, pages 317–326, Toronto, ON, Canada, July 29 – August 2, 2019. Association for Computing Machinery.
- [3] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected  $O(1)$  rounds, expected  $O(n^2)$  communication, and optimal resilience. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 320–334, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany.
- [4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *40th ACM Symposium Annual on Principles of Distributed Computing*, pages 363–373. Association for Computing Machinery, Portland, OR, USA, 2021.
- [5] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *38th ACM Symposium Annual on Principles of Distributed Computing*, pages 337–346, Toronto, ON, Canada, July 29 – August 2, 2019. Association for Computing Machinery.
- [6] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. Cryptology ePrint Archive, Report 2021/1587, 2021. <https://eprint.iacr.org/2021/1587>.
- [7] Handan Kiliç Alper and Jeffrey Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 157–188, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.
- [8] Diego F. Aranha, Anders P. K. Dalskov, Daniel Escudero, and Claudio Orlandi. Improved threshold signatures, proactive secret sharing, and input certification from LSS isomorphisms. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology - LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America*, volume 12912 of *Lecture Notes in Computer Science*, pages 382–404, Bogotá, Colombia, October 6–8, 2021. Springer, Heidelberg, Germany.
- [9] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 121–151, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.
- [10] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.
- [11] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 353–380, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

- [12] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany.
- [13] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.
- [14] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In Tanja Lange and Orr Dunkelman, editors, *Progress in Cryptology - LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America*, volume 11368 of *Lecture Notes in Computer Science*, pages 352–377, Havana, Cuba, September 20–22, 2017. Springer, Heidelberg, Germany.
- [15] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strohli. Asynchronous verifiable secret sharing and proactive cryptosystems. In Vijayalakshmi Atluri, editor, *ACM CCS 2002: 9th Conference on Computer and Communications Security*, pages 88–97, Washington, DC, USA, November 18–22, 2002. ACM Press.
- [16] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *19th ACM Symposium Annual on Principles of Distributed Computing*, pages 123–132, Portland, OR, USA, July 16–19, 2000. Association for Computing Machinery.
- [17] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1769–1787, Virtual Event, USA, November 9–13, 2020. ACM Press.
- [18] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. *Cryptology ePrint Archive*, Report 2021/060, 2021. <https://eprint.iacr.org/2021/060>.
- [19] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 98–115, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.
- [20] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th Annual ACM Symposium on Theory of Computing*, pages 42–51, San Diego, CA, USA, May 16–18, 1993. ACM Press.
- [21] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In *34th International Symposium on Distributed Computing*, volume 25 of *LIPICS*, pages 1–25. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2020.
- [22] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

- [23] Elizabeth Crites, Chelsea Komlo, and Mary Maller. 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. Securing DNSSEC keys via threshold ECDSA from generic MPC. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II*, volume 12309 of *Lecture Notes in Computer Science*, pages 654–673, Guildford, UK, September 14–18, 2020. Springer, Heidelberg, Germany.
- [25] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *43rd IEEE Symposium on Security and Privacy (to appear)*. IEEE Computer Society Press, San Francisco, CA, USA, 2022.
- [26] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO’87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.
- [27] Manu Drijvers, Kasma Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.
- [28] Michael Fischer, Nancy A. Lynch, and Robert Patterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
- [29] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.
- [30] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 63–95, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.
- [31] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1179–1194, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
- [32] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 156–174, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany.
- [33] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.
- [34] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 803–818, Virtual Event, USA, November 9–13, 2020. ACM Press.

- [35] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 221–242, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.
- [36] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. *Journal of Cryptology*, 31(1):276–306, January 2018.
- [37] Julia Kastner, Julian Loss, and Jiayu Xu. On pairing-free blind signature schemes in the algebraic group model. In *International Conference on Public-Key Cryptography-PKC*, volume 2 of *LNCS*, pages 468–497. Springer, 2022.
- [38] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1751–1767, Virtual Event, USA, November 9–13, 2020. ACM Press.
- [39] Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. In *Selected Areas in Cryptography-SAC*, pages 34–65, 2020.
- [40] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. In *2021 IEEE Symposium on Security and Privacy*, pages 608–625, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
- [41] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM Symposium Annual on Principles of Distributed Computing*, pages 303–312, Paris, France, July 15–18, 2014. Association for Computing Machinery.
- [42] Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.
- [43] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, March 2022.
- [44] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1837–1854, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
- [45] Julian Loss. *New techniques for the modular analysis of digital signature schemes*. PhD thesis, Ruhr University Bochum, Germany, 2019.
- [46] Julian Loss and Tal Moran. 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. Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In Yuval Emek and Christian Cachin, editors, *39th ACM Symposium Annual on Principles of Distributed Computing*, pages 129–138, Virtual Event, Italy, August 3–7, 2020. Association for Computing Machinery.
- [48] Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017: 8th Innovations in Theoretical Computer Science Conference*, volume 4266, pages 6:1–6:1, Berkeley, CA, USA, January 9–11, 2017. LIPIcs.

- [49] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press.
- [50] Fabrice Mouhartem. 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. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.
- [52] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1717–1731, Virtual Event, USA, November 9–13, 2020. ACM Press.
- [53] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO’91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.
- [54] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.
- [55] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 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. Short pairing-free blind signatures with exponential security. In *Advances in Cryptology - EUROCRYPT (to appear)*, 2022.

## A DKG Protocols

We describe the DKG protocols that we refer to directly in the main body of this paper.

### A.1 TD-DKG Protocol

This is the usual key generation protocol where a trusted dealer shares a secret among  $n$  parties  $P_1, \dots, P_n$  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  $\mathbb{Z}_p$ :

$$f = a_0 + a_1X + \dots + a_tX^t$$

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  $(pk_1, \dots, pk_n)$ , 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  $(pk_1, \dots, pk_n)$  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  $P_1, \dots, P_n$  via  $n$  parallel executions of Feldman's VSS scheme. This protocol can be found in [53] and [33].

**Protocol JF-DKG:**

1. Each party  $P_i$  chooses a random polynomial  $f_i$  of degree  $t$  with coefficients in  $\mathbb{Z}_p$ :

$$f_i = a_{i0} + a_{i1}X + \dots + a_{it}X^t.$$

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

$$g^{s_{ij}} = \prod_{k=0}^t (A_{ik})^{j^k} \tag{1}$$

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  $\mathcal{Q} \subset \{1, \dots, n\}$  as the set of non-disqualified parties.
4. The public key  $y$  is computed as  $y = \prod_{i \in \mathcal{Q}} A_{i0}$ . The public verification values are computed as  $A_k = \prod_{i \in \mathcal{Q}} A_{ik}$  for  $k \in [t]$ . Each party  $P_j$  sets his secret key share as  $x_j = \sum_{i \in \mathcal{Q}} s_{ij}$ . The secret key  $x$  itself is not computed by any party, but is equal to  $x = \sum_{i \in \mathcal{Q}} a_{i0}$ .

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

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

Let  $H' : \{0, 1\}^* \rightarrow \mathbb{G}^*$  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  $\mathbb{Z}_p$ :

$$\begin{aligned} f_i(X) &= a_{i0} + a_{i1}X + \dots + a_{it}X^t, \\ g_i(X) &= b_{i0} + b_{i1}X + \dots + b_{it}X^t. \end{aligned}$$

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

$$g^{s_{ij}} h^{u_{ij}} = \prod_{k=0}^t (C_{ik})^{j^k} \quad (1)$$

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  $\mathcal{Q} \subset \{1, \dots, n\}$ .
3. Each party  $P_j$  sets his share of the secret key as  $x_j = \sum_{i \in \mathcal{Q}} s_{ij}$  and the value  $x'_j = \sum_{i \in \mathcal{Q}} u_{ij}$ . The distributed secret key  $x$  itself is not computed by any party, but it is equal to  $x = \sum_{i \in \mathcal{Q}} z_i$ .
4. Each party  $P_i, i \in \mathcal{Q}$ , exposes  $y_i = g^{z_i}$  via Feldman-VSS:
  - (a) Each party  $P_i, i \in \mathcal{Q}$ , 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  $\mathcal{Q}$  by checking the identity

$$g^{s_{ij}} = \prod_{k=0}^t (A_{ik})^{j^k} \quad (2)$$

for all  $i \in \mathcal{Q}$ . 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  $i \in \mathcal{Q}$ . The public value  $y$  is computed as  $y = \prod_{i \in \mathcal{Q}} y_i$ .

*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 Generic Reduction

**Theorem A.1.** *Let  $1 \leq r \leq t + 1$ , and let DKG have  $(t, k)$ -oracle-aided algebraic security. And let  $R$  be a generic reduction such that for every algebraic forger  $F_{alg}$  that breaks  $\text{Th-BLS}_{\text{DKG}}$ ,  $R^{F_{alg}}$  is an algorithm that breaks  $r$ -OMDL. Furthermore, let the  $(n \times r)$  - matrix  $Q$  formed by the algebraic coefficients of the public key shares have rank  $r$ . Then any set of  $t + 1$  row vectors of  $Q$  generates a space of dimension  $r$ .*

*Proof.* Let  $\xi = g^{z_1}, \dots, g^{z_r} \in \mathbb{G}$  be the OMDL instance of degree  $r$ . And let  $f = a_0 + a_1X + \dots + a_tX^t \in \mathbb{Z}_p[X]$  be the degree  $t$  polynomial that determines the secret key shares. 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  $(\hat{a}_i, a_{i,1}, \dots, a_{i,r})$  for all  $i \in [n]$  such that

$$g^{x_i} = g^{\hat{a}_i} \cdot \prod_{j=1}^r (g^{z_j})^{a_{i,j}} = g^{\hat{a}_i} \cdot (g^{z_1})^{a_{i,1}} \cdot \dots \cdot (g^{z_r})^{a_{i,r}}.$$

Let  $Q$  be the  $(n \times r)$  - matrix over the field  $\mathbb{Z}_p$  formed by these algebraic coefficients

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

By Lagrange interpolation in the exponent though the public key shares, the values  $g^{a_i}$  also come together with a representation  $(\hat{d}_i, d_{i,1}, \dots, d_{i,r})$  for all  $i \in [0, t]$  such that

$$g^{a_i} = g^{\hat{d}_i} \cdot \prod_{j=1}^r (g^{z_j})^{d_{i,j}} = g^{\hat{d}_i} \cdot (g^{z_1})^{d_{i,1}} \cdot \dots \cdot (g^{z_r})^{d_{i,r}}.$$

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

$$D := \begin{pmatrix} d_{0,1} & d_{0,2} & \cdots & d_{0,r} \\ d_{1,1} & d_{1,2} & \cdots & d_{1,r} \\ \vdots & \vdots & & \vdots \\ d_{t,1} & d_{t,2} & \cdots & d_{t,r} \end{pmatrix},$$

and let  $V = V(1, \dots, n)$  be the  $n \times (t + 1)$  Vandermonde matrix for the numbers  $1, \dots, n$ . By construction of  $Q, V$  and  $D$ , we have the identity  $Qz = VDz$  and thus  $(Q - VD) \cdot z = 0$ . Since  $R$  is a generic reduction, there is at least one component of the vector  $z = (z_1, \dots, z_r)$  that looks uniformly random for  $R$ . As a consequence of the Schwartz–Zippel lemma, it follows  $Q - VD = 0$  with overwhelming probability (to be precise, with probability at least  $1 - 1/p$ ). Therefore, we may assume that  $Q = VD$ . Since  $\text{rank}(Q) = \text{rank}(VD) \leq \min\{\text{rank}(V), \text{rank}(D)\}$  and  $V$  has full rank  $t + 1$ , it follows  $r = \text{rank}(Q) \leq \text{rank}(D) \leq r$  so that  $\text{rank}(D) = r$ . Now let  $S \subset [n]$  be some  $(t + 1)$ -element subset of  $\{1, \dots, n\}$ . We let  $Q_S$  (and  $V_S$  likewise) be

the  $(t+1) \times r$  - matrix formed by the row vectors of  $Q$  specified by  $S$ . The equation  $Q = VD$  translates into  $Q_S = V_S D$ , where  $V_S$  is a  $(t+1) \times (t+1)$  - matrix of rank  $t+1$ . Finally, Sylvester's rank inequality yields

$$\begin{aligned} & (t+1) + r - (t+1) \\ &= \text{rank}(V_S) + \text{rank}(D) - (t+1) \\ &\leq \text{rank}(Q_S) \leq \min\{t+1, r\} = r, \end{aligned}$$

that is  $r \leq \text{rank}(Q_S)$  and thus  $r = \text{rank}(Q_S)$ . This means that any set  $S$  of  $t+1$  row vectors of  $Q$  generates a space of dimension  $r$ .  $\square$