

# Securing Group Key Exchange against Strong Corruptions

## [Full version]

Emmanuel Bresson<sup>1</sup> and Mark Manulis<sup>2</sup>

<sup>1</sup> DCSSI Crypto Lab, Paris, France  
emmanuel@bresson.org

<sup>2</sup> UCL Crypto Group, Louvain-la-Neuve, Belgium  
mark.manulis@uclouvain.be

**Abstract.** When users run a group key exchange (GKE) protocol, they usually extract the key from some auxiliary (ephemeral) secret information generated during the execution. *Strong corruptions* are attacks by which an adversary can reveal these ephemeral secrets, in addition to the possibly used long-lived keys. Undoubtedly, security impact of strong corruptions is serious, and thus specifying appropriate security requirements and designing secure GKE protocols appears an interesting yet challenging task — the aim of our paper.

We start by investigating the current setting of strong corruptions and derive some further refinements such as *opening attacks* that allow to reveal ephemeral secrets of users without their long-lived keys. This allows to consider even stronger attacks against honest, but “opened” users. Further, we define strong security goals for GKE protocols in the presence of such powerful adversaries and propose a Tree Diffie-Hellman protocol immune to their attacks. Our security definitions in particular include the case of malicious insiders, for appropriate security goals such as mutual authentication, key confirmation, contributiveness and key-replication resilience. The proposed protocol proceeds in three rounds and is provably secure in the standard model.

**Key words.** Authenticated group key exchange, insider attacks, strong corruptions, mutual authentication, contributiveness, Tree Diffie-Hellman

## 1 Introduction

A group key exchange (GKE) protocol provides participants with a common secret group key. The main (semantic) security requirement called *Authenticated Key Exchange (AKE)* [8,9] aims to ensure that the established key is indistinguishable from a random one by any outsider adversary. The second requirement called *Mutual Authentication (MA)* [8] aims to ensure that all legitimate protocol participants and only them have actually computed identical session group keys. These security requirements have been extensively studied in the literature (see the recent survey in [27]). However in the most basic scenarios, all users are somehow protected, that is, the adversary has no control over them, and is restricted to attacks carried out through the network (which nevertheless include impersonation attacks where the adversary talks on the network by *pretending* being a legitimate user).

In order to take into account real-life threats on users, the notion of *forward secrecy* is usually considered. Forward secrecy means that the established session key remains secure “in the future”, that is, remains indistinguishable from random even if the adversary corrupts a long-lived key in the future. The notion is motivated by the fact that, by nature, long-lived keys get more chance to be leaked to an attacker than ephemeral secrets.

The next known kind of corruptions, referred to as *strong corruptions* in [31,33,9] provides the adversary with even more information. Namely, the adversary gets the user’s ephemeral secrets in addition to the long-lived keys. But, he is not allowed to get the established session group key. In [31], Shoup explains why such a separation makes sense: session keys are typically controlled by higher-level

applications that will use them, while internal, ephemeral secrets are specific to the group key exchange protocol execution and could be erased once this protocol is finished.

Actually, it seems impossible to obtain security when ephemeral secrets are revealed: if the adversary (even “passively”) can learn all intermediate key material, then he will likely be able to compute the final key. However, such scenario becomes much more interesting for dynamic groups: in many cases, ephemeral secrets of a particular session are subsequently re-used (in addition to some refreshed data) when the group key is updated. Then, it is important to ask how the knowledge of ephemeral secrets in a corrupted session impacts the security of other sessions (past and future). This is precisely where the notion of *strong forward/backward secrecy* raises up.

At this point, we emphasize that the kind of corruptions considered in this paper is slightly different than those encountered in general multi-party cryptographic protocols. The corruptions we are studying here are *adaptive* (opposed to static) in the sense that the adversary chooses which users to corrupt based on the information he gained so far and in any stage of the protocol execution. On the other hand, the corruption mode is *passive* rather than *active*, because the adversary can only “read” secrets held by the attacked user (whatever these secrets are ephemeral or long-lived): in no case he gets control on the user’s behavior. However, through the knowledge of the long-lived key he can (typically) inject signed messages on behalf of the user while preventing the original user’s messages from being delivered. In fact, this allows an active participation of the adversary during the protocol execution, and thus we say the adversary is *active*; but this refers to his ability to control the network, not the user’s behavior.

In order to further refine the security definitions, our intention is to separate the long-lived key from the ephemeral secrets and to specify *when* the adversary can learn them. Through this separation we explicitly allow the adversary to reveal ephemeral secrets without revealing the long-lived key; we call this *opening attacks*. They are the balanced complement of weak corruption attacks, where long-lived keys are revealed, but ephemeral secrets are not. We note that under opening attacks, there is hope to prevent the adversary from actively participating in the protocol on behalf of the opened parties since he does not receive the long-lived keys. Finally, we notice that the strong corruption model in its current form is the best (or worst) of two worlds: if the adversary corrupts then it obtains the long-lived keys and the ephemeral data, if it does not corrupt then it obtains nothing. But separating the attacks in two distinct modes allows to refine and opt for stronger security definitions.

The AKE requirement is usually defined from the perspective of some (fresh) session, and thus makes sense only if the adversary is restricted to neither participate on behalf of an user nor to obtain any ephemeral secret in that session. On the other hand, the MA requirement remains meaningful even without such limitations. Even if achieving MA without AKE is of low interest for key exchange protocols, it is still legitimate to ask whether achieving MA under strong corruptions during the attacked session is possible. This especially, since the MA requirement still makes sense in the presence of *malicious participants/insiders* (users whose long-lived keys are known to the adversary).

We emphasize that this setting of strong corruptions differs from the one considered in general multi-party protocols. In multi-party computation scenario, malicious participants *are* the adversary: their behavior is by definition under adversary’s control, and the adversary’s goal is to gain information about honest users’ internal state. While our setting is complementary: the adversary has (*via* queries) information about the long-lived keys of some users who it can impersonate; additionally, it can have information about the internal states of some other users without being able to impersonate them (such users are called *honest*); the goal of the adversary is then to influence the behavior of at least one honest user whom it cannot impersonate.

Furthermore, consideration of malicious insiders in multiple sessions raises attacks related to *key control* and *contributiveness*: for instance, think of a participant who can force the same key to be obtained twice (*key-replication* [25]) and used in two distinct applications. Here we recall that the question on who controls the value of the group key states the important difference between group key exchange and group key transport protocols (in which a trusted party chooses the key on behalf of participants) [12]. In GKE protocols it is essential that the key is computed from inputs (contributions)

of all participants such that even a strict subset of (malicious) insiders cannot influence the resulting value of the group key. Especially, when considering malicious insiders that can choose own contributions arbitrarily and may additionally reveal internal states of honest participants at any stage of the protocol execution through opening attacks, preventing key control and ensuring contributiveness for the honest users appears to be a challenging task.

## 1.1 Related Work

**Original Definitions** The definitions of AKE- and MA-security (without strong corruptions and only for honest users) were originally given by Bresson *et al.* [11] (see [21, 22, 19] for further variants, and [13] for some flaws). In [9], Bresson *et al.* modeled strong corruptions, but for AKE-security only, following the ideas of Shoup [31], and Canetti and Krawczyk [15] for two-party protocols, for which such strong AKE-security has been recently modeled in LaMacchia *et al.* [26].

Katz and Shin in 2005 extended the definition of MA-security [20] by assuming misbehaving (malicious) protocol participants; and they provided a concrete generic solution (compiler) to prevent these attacks, however, without considering opening attacks against ephemeral secrets as well as key control and contributiveness. The significance of security against malicious participants was also recognized by Choo *et al.* [16] through *unknown-key share attacks*, by which an active adversary tries to make an honest protocol participant believe that the group key is shared with one party when it is in fact shared with another party.

**On Key Control and Contributiveness** Mitchel *et al.* [29] (see also Boyd and Mathuria [6]) gave informal definition of *key control*, to describe attacks where participants try to influence the resulting value of the key. Yet informally, Ateniese *et al.* [2] proposed the notion of *contributiveness* meaning that all participants must equally contribute to the computation of the key and guarantee its freshness (see [33]); these definitions emphasize the difference between key distribution and key exchange [28]. Following these requirements Bresson and Catalano [7] have considered the (weaker) case where participants are honest but have biased source of randomness so that an adversary can possibly gain extra information about the key. Deepening this, Bohli *et al.* [4] gave definitions of key control and contributiveness considering a (stronger) case where participants *deliberately* wish to influence the resulting value of the group key. Still, their definitions are based on the model from [11], and, thus, do not consider strong corruptions. Finally, Krawczyk mentioned in [25] that a key exchange protocol should prevent *key-replication attacks* whose goal is to influence the acceptance of the same key in different protocol sessions.

**Other Work Close to Ours** Independent of our work, Desmedt *et al.* [18] considered a property of *non-malleability* for GKE protocols, which is close to key control and contributiveness. Their security goal, called *shielded-insider privacy*, aims to prevent attacks where an *outsider adversary* upon communication with some malicious participants *prior* to the protocol execution, obtains information about the later computed group key. In order to ensure shielded-insider privacy they use Pedersen’s commitments [30]; however in case of strong corruptions committed secrets can still be revealed to the adversary (due to opening attacks), so that malicious participants would still be able to bias the computation. In our model we do not consider this scenario explicitly, but focus on the (in)ability of the adversary representing malicious participants to predict the resulting value of the later established group key.

Recently, Manulis [27] analyzed several existing models for GKE protocols with respect to considering strong corruptions: he pointed out that security against strong corruptions is currently considered in a rather restrictive way: only for *strong forward secrecy* of AKE-security. Moreover, none of the available game-based security models is complete enough to unify the most important definitions of AKE-, MA-security, and key control and contributiveness.

## 1.2 Contributions and Organization

In this paper we solve most of the problems put in light above by revisiting the security model for GKE protocols from the perspective of strong corruptions and designing a provably secure GKE protocol that resists these attacks.

**Security Model and Stronger Definitions** As our first contribution in Section 2 we provide the following:

- we model a powerful adversary who is given access to strong corruptions, by describing an appropriate game-based security model for GKE protocols, thus extending the ideas from Bresson *et al.* [9], in particular towards strong AKE-security;
- in our definition of strong MA-security we take into account the adversary that participates on behalf of corrupted users during the attacked session *and* reveals the ephemeral secrets of all other (honest) users; due to opening attacks our definition is stronger than the related one given in [20];
- we formalize strong contributiveness as security against attacks aiming to enforce a value chosen by a powerful adversary as a group key (e.g. key-replication attacks [25]); since the adversary can corrupt and, thus, impersonate users, and additionally open all other honest users our requirement is stronger than the one in [4].

**GKE Protocol with Strong Security** As a second contribution in Section 3, we describe a 3-round GKE protocol, named TDH1, and prove that it provides strong versions of AKE-, MA-security and contributiveness, while the deployed techniques can be seen as general for many GKE protocols. TDH1 tolerates the following numbers of malicious insiders (out of  $n$  participants in total): for MA-security up to  $n - 2$ , for contributiveness up to  $n - 1$ , whereby all remaining honest users might be opened! Our security proofs do not rely on the Random Oracle Model [3]. The AKE-security of TDH1 is based on the Tree Decisional Diffie-Hellman (TDDH) assumption, introduced by Kim *et al.* in [23, 24]. We give a formal definition of the underlying TDDH problem and show its polynomial equivalence to the standard Decisional Diffie-Hellman (DDH) problem [5] by a proof which addresses *arbitrary* full binary trees, i.e., trees where each node has exactly zero or two leaves (note, Kim *et al.* addressed only a subset, i.e., linear and complete trees).

## 2 Execution Model and Strong Security Definitions for GKE

We start by (re)stating existing definitions and classical notations using the game-based approach. Note that another way (which we do not consider here) to define security requirements is to use the simulation-based approach, e.g. [20] (but see Remark 1).

### 2.1 Protocol Execution and Participants

**Users, Instance Oracles, Long-Lived Keys** Let  $\mathcal{U}$  be a set of  $N$  users. Each  $U_i \in \mathcal{U}$  holds a long-lived key  $LL_i$  and has several instances called *oracles*, denoted  $\Pi_i^s$  for  $s \in \mathbb{N}$ , participating in distinct concurrent executions. (When we do not refer to a specific user  $U_i$  we use the index  $U$ , e.g.  $\Pi_U^s$ .)

**Internal States** Every  $\Pi_U^s$  maintains an *internal state information*  $\mathbf{state}_U^s$  which is composed of all ephemeral secret information used during the protocol execution. The long-lived key  $LL_U$  is, in nature, excluded from it (moreover the long-lived key is specific to the user, not to the oracle). An oracle  $\Pi_U^s$  is *unused* until initialization (by which it is given the long-lived key  $LL_U$ ). It then becomes a group member, associated to a particular session, and turns into the *stand-by* state where it waits for an

invocation to execute the protocol. When the protocol starts, the oracle learns its partner id  $\text{pid}_U^s$  (and possibly  $\text{sid}_U^s$ ) and turns into a *processing* state where it sends, receives and processes messages. During that stage, the internal state information  $\text{state}_U^s$  is maintained. After having computed  $k_U^s$  oracle  $\Pi_U^s$  *accepts* and *terminates* the execution of the protocol operation (possibly after some additional auxiliary steps) meaning that it would not send or receive further messages. If the protocol fails,  $\Pi_U^s$  terminates without accepting, and  $k_U^s$  is set to an **undefined** value.

**Session Group Key, Session and Partner IDs, Group Members** Every session is identified by a unique, publicly-known *session id*  $\text{sid}_U^s$ . In each session each oracle  $\Pi_U^s$  gets a value  $\text{pid}_U^s$  that contains the identities of participating users (including  $U$ ) and computes the *session group key*  $k_U^s \in \{0, 1\}^\kappa$ , where  $\kappa$  is a security parameter.

By  $\mathcal{G}(\Pi_i^s) = \{\Pi_j^t, \text{ where } U_j \in \text{pid}_{U_i}^s \text{ and } \text{sid}_i^s = \text{sid}_j^t\}$  we denote the *group* of oracle  $\Pi_i^s$  and say that  $\Pi_i^s$  and  $\Pi_j^t$  are *partnered* if  $\Pi_j^t \in \mathcal{G}(\Pi_i^s)$  and  $\Pi_i^s \in \mathcal{G}(\Pi_j^t)$ . Sometimes we simply write  $\mathcal{G}$  to denote the *group of oracles* participating in the same protocol session. Then each oracle in  $\mathcal{G}$  is called a *group member*. Note that oracles in  $\mathcal{G}$  may be ordered, e.g., lexicographically based on the user identities.

**Definition 1.** A GKE protocol  $\mathsf{P}$  consists of a key generation algorithm **KeyGen**, and a protocol **Setup**:

- $\mathsf{P}.\text{KeyGen}(1^\kappa)$ : On input a security parameter  $1^\kappa$  each user in  $\mathcal{U}$  is provided with a long-lived key  $LL_U$ .
- $\mathsf{P}.\text{Setup}(\mathcal{S})$ : On input a set  $\mathcal{S}$  of  $n$  unused oracles a new group  $\mathcal{G}$  is created and set to be  $\mathcal{S}$ . A probabilistic interactive protocol is executed between the oracles in  $\mathcal{G}$  such that all oracles accept with the session group key and terminate.

A protocol is said to be *correct* if, when no adversary is present, all participants compute the same key. Note that our definition is independent of the communication channel, e.g., (asymmetric) broadcast, multicast, or unicast.

## 2.2 Strong Adversarial Model

Now we consider an adversary  $\mathcal{A}$  which is a Probabilistic Polynomial-Time (PPT) algorithm having complete control over the network.  $\mathcal{A}$  can interact with protocol participants via queries to their oracles. Note that our security model (similar to [9, 20, 4]) does not deal with the issues of denial-of-service and fault-tolerance; our security definitions aim to prevent honest participants from accepting the group key biased by malicious insiders.

### Queries to the Instance Oracles

- $\text{Execute}(\mathcal{S})$ :  $\mathcal{A}$  eavesdrops an honest execution of  $\mathsf{P}.\text{Setup}$  between a chosen set of oracles and is given the resulting transcript of  $\mathsf{P}.\text{Setup}(\mathcal{S})$ .
- $\text{Send}(\Pi_U^s, m)$ :  $\mathcal{A}$  sends message  $m$  to oracle  $\Pi_U^s$  and receives the response  $\Pi_U^s$  would have generated after having (honestly) processed message  $m$ . The response may be empty if  $m$  is incorrect. The adversary can have  $\Pi_U^s$  invoking  $\mathsf{P}.\text{Setup}$  with the oracles in  $\mathcal{S}$  via a query of the form  $\text{Send}(\text{'start'}, \Pi_U^s, \mathcal{S})$ :  $\mathcal{A}$  gets the first message that  $\Pi_U^s$  would generate in this case.
- $\text{RevealKey}(\Pi_U^s)$ :  $\mathcal{A}$  is given the session group key  $k_U^s$ , provided  $\Pi_U^s$  has accepted.
- $\text{RevealState}(\Pi_U^s)$ :  $\mathcal{A}$  is given the internal state information  $\text{state}_U^s$  (consisting of ephemeral secrets).
- $\text{Corrupt}(U)$ :  $\mathcal{A}$  is given the long-lived key  $LL_U$ .
- $\text{Test}(\Pi_U^s)$ :  $\mathcal{A}$  tests the semantic security of  $k_U^s$ . Formally, a bit  $b$  is privately flipped and  $\mathcal{A}$  is given  $k_U^s$  if  $b = 1$  and a random string if  $b = 0$ .

*Remark 1.* Note that natural separation of the queries *RevealState* and *Corrupt* explicitly provides the possibility for the opening attacks mentioned in the introduction. By asking the *RevealState* query to an instance oracle  $\Pi_U^s$  the adversary reads out its ephemeral secrets but cannot impersonate  $U$  in the protocol execution, unless a *Corrupt*( $U$ ) query is asked (in which case all instance oracles  $\Pi_U^s$  become malicious insiders through possible impersonation actions of  $\mathcal{A}$ ). Thus, just opening a user does not make him malicious. In contrast, simulation-based security models (e.g. Universal Composability / Reactive Simulatability) handle strong corruptions typically as follows: upon corrupting a user the adversary learns *all* information known to that user and controls him thereafter. Obviously, in the simulation-based models opening attacks (which strengthen the adversary) are currently *not* modeled. From this perspective, our adversarial setting appears to be stronger than the simulation-based one focusing on similar security goals.

### 2.3 Strong AKE-Security

In case of strong AKE-security one must also consider the knowledge of the adversary about long-lived keys *and* ephemeral secrets of session participants. If the adversary obtains a long-lived key before the session is started then it can impersonate a user, and thus, learn the session key. And if the adversary is allowed to obtain long-lived keys before the session is finished then it should be restricted from actively using these keys during that time [21].

On the other hand, the adversary should be allowed to reveal ephemeral secrets of participants before the session starts<sup>1</sup> *and* after the session is finished. Note that, if one allows long-lived key corruptions in later sessions, revealing ephemeral secrets during the attacked session wouldn't make sense. In order to model the described requirements for the adversarial knowledge we define the notion of oracle freshness, extending those given in [9, 21] by the condition that allows opening attacks in the previous and later sessions.

**Definition 2.** (Oracle Freshness) *In the execution of  $\mathsf{P}$  the oracle  $\Pi_U^s$  is fresh if **all** of the following holds:*

- (a) *no  $U_i \in \text{pid}_U^s$  is asked for a query *Corrupt* prior to a query of the form *Send*( $\Pi_j^t, m$ ) with  $U_j \in \text{pid}_U^s$  until  $\Pi_U^s$  and its partners accept,*
- (b) *neither  $\Pi_U^s$  nor any of its partners is asked for a query *RevealState* until  $\Pi_U^s$  and its partners accept,*
- (c) *neither  $\Pi_U^s$  nor any of its partners is asked for a query *RevealKey* after having accepted.*

*We say that a session is fresh if all participating oracles are fresh.*

We note that the above definition ensures that if at least one oracle participating in a session is fresh then the whole session is fresh too because freshness of one oracle requires freshness of all its partners. This notion of oracle freshness simplifies the following definition of strong AKE-security for GKE protocols.

**Definition 3.** (Strong AKE-Security) *Let  $\mathsf{P}$  be a correct GKE protocol and  $b$  a uniformly chosen bit. Consider an adversary  $\mathcal{A}$  against the AKE-security of  $\mathsf{P}$ . We define the adversarial game  $\text{Game}_{\mathcal{A}, \mathsf{P}}^{\text{ake}-b}(\kappa)$  as follows:*

- *after initialization,  $\mathcal{A}$  interacts with instance oracles via queries;*
- *at some point  $\mathcal{A}$  asks a **Test** query to a **fresh** oracle  $\Pi_U^s$  which has accepted;*
- *$\mathcal{A}$  continues interacting with instance oracles;*
- *when  $\mathcal{A}$  terminates, it outputs a bit, which we define to be the output of the game.*

<sup>1</sup> A GKE protocol may use auxiliary secrets pre-computed offline in order to achieve better performance during the communication phase. Such protocols are not strong AKE-secure since the adversary can break into the internal states prior to the protocol execution.

We define:  $\text{Adv}_{\mathcal{A},\mathcal{P}}^{\text{ake}}(\kappa) := \left| 2 \Pr[\text{Game}_{\mathcal{A},\mathcal{P}}^{\text{ake}-b}(\kappa) = b] - 1 \right|$

and denote with  $\text{Adv}_{\mathcal{P}}^{\text{ake}}(\kappa)$  the maximum advantage over all PPT adversaries  $\mathcal{A}$ . We say that a GKE protocol  $\mathcal{P}$  provides strong AKE-security if this advantage is negligible.

We again stress that (strong) AKE-security makes sense for adversaries that are not able to corrupt users and act on their behalf during the attacked session or reveal any ephemeral secrets used in that session — this is guaranteed by the *freshness* property.

## 2.4 Strong MA-Security

We say that  $\Pi_U^s$  is a *malicious participant/insider* if the adversary has previously asked  $\text{Corrupt}(U)$ . In all other cases  $\Pi_U^s$  is *honest*. The following definition of MA-security unifies the requirements of mutual authentication, key confirmation, and unknown-key share resilience w.r.t. malicious insiders and internal state corruptions revealing ephemeral secrets of honest participants at any protocol stage.

**Definition 4.** (Strong MA-Security) *Let  $\mathcal{P}$  be a correct GKE protocol and  $\mathcal{A}$  an adversary who is allowed to query  $\text{Send}$ ,  $\text{Execute}$ ,  $\text{RevealKey}$ ,  $\text{RevealState}$ , and  $\text{Corrupt}$ . We denote this interaction as  $\text{Game}_{\mathcal{A},\mathcal{P}}^{\text{ma}}(\kappa)$ . We say that  $\mathcal{A}$  wins if at some point, there exists an honest user  $U_i$  whose instance oracle  $\Pi_i^s$  has accepted with  $k_i^s$  and another user  $U_j \in \text{pid}_i^s$  that is uncorrupted at the time  $\Pi_i^s$  accepts such that*

- (a) *There is **no** instance oracle  $\Pi_j^t$  with  $(\text{pid}_j^t, \text{sid}_j^t) = (\text{pid}_i^s, \text{sid}_i^s)$ , **or***
- (b) *There is **an** instance oracle  $\Pi_j^t$  with  $(\text{pid}_j^t, \text{sid}_j^t) = (\text{pid}_i^s, \text{sid}_i^s)$  that has accepted with  $k_j^t \neq k_i^s$ .*

The maximum probability of this event is denoted  $\text{Succ}_{\mathcal{P}}^{\text{ma}}(\kappa)$ ; we say that a GKE protocol  $\mathcal{P}$  provides strong MA-security if this probability is negligible.

## 2.5 Strong Contributiveness

In the following we propose a definition which unifies the requirements of key control, contributiveness and unpredictability of group keys, taking into account malicious insiders. Informally, we consider an active adversary  $\mathcal{A}$  which can both *corrupt* and *open* participants at any stage of the protocol execution in such a way that there exists at least one honest oracle (which may nevertheless be opened!) that accepts the session group key chosen by the adversary. In particular, our definition prevents malicious insiders from enforcing honest participants to accept a group key used in some previous session (key-replication attacks [25], see also Remark 2).

**Definition 5.** (Strong Contributiveness) *Let  $\mathcal{P}$  be a correct GKE protocol and  $\mathcal{A}$  an adversary operating in two stages (prepare and attack) and having access to the queries  $\text{Send}$ ,  $\text{Execute}$ ,  $\text{RevealKey}$ ,  $\text{RevealState}$ , and  $\text{Corrupt}$ . We define the following game  $\text{Game}_{\mathcal{A},\mathcal{P}}^{\text{con}}(\kappa)$ :*

- $\mathcal{A}(\text{prepare})$  interacts with instance oracles via queries;
- $\mathcal{A}(\text{prepare})$  outputs  $\tilde{k} \in \{0,1\}^\kappa$ , and some state information  $\zeta$ ;
- the following sets are built:  $\mathcal{G}_{\text{us}}$  consisting of all honest used oracles,  $\mathcal{G}_{\text{std}}$  consisting of all honest oracles that are in the stand-by state<sup>2</sup>, and  $\Psi$  consisting of session ids  $\text{sid}_i^t$  for every  $\Pi_i^t \in \mathcal{G}_{\text{us}}$ ;
- $\mathcal{A}(\text{attack}, \zeta)$  interacts with instance oracles via queries;
- at the end of this stage  $\mathcal{A}$  outputs  $(s, U)$ .

The adversary  $\mathcal{A}$  wins in  $\text{Game}_{\mathcal{A},\mathcal{P}}^{\text{con}}(\kappa)$  if **all** of the following holds:

<sup>2</sup> Note that  $\mathcal{G}_{\text{std}} \subseteq \mathcal{G}_{\text{us}}$ .

- (a)  $\Pi_U^s$  is honest, has terminated accepting  $\tilde{k}$ ,  $\Pi_U^s \notin \mathcal{G}_{\text{us}} \setminus \mathcal{G}_{\text{std}}$  and  $\text{sid}_U^s \notin \Psi$ .
- (b) There are at most  $n - 1$  corrupted users  $U_i$  having oracles  $\Pi_i^t$  partnered with  $\Pi_U^s$ .

We define:  $\text{Succ}_{\mathcal{A}, \mathcal{P}}^{\text{con}}(\kappa) := \Pr[\mathcal{A} \text{ wins in } \text{Game}_{\mathcal{A}, \mathcal{P}}^{\text{con}}(\kappa)]$

and denote with  $\text{Succ}_{\mathcal{P}}^{\text{con}}(\kappa)$  the maximum probability of this event over all PPT adversaries  $\mathcal{A}$ ; we say  $\mathcal{P}$  provides strong contributiveness if this probability is negligible in  $\kappa$ .

The first requirement ensures that  $\Pi_U^s$  belongs to an honest user. The set  $\mathcal{G}_{\text{us}} \setminus \mathcal{G}_{\text{std}}$  consists of all oracles that at the end of the **prepare** stage have already terminated or remain in the processing state. Thus, requiring  $\Pi_U^s \notin \mathcal{G}_{\text{us}} \setminus \mathcal{G}_{\text{std}}$  prevents the case where  $\mathcal{A}$  while being a session participant outputs  $\tilde{k}$  for the still running protocol execution which is then accepted by  $\Pi_U^s$  that participates in the same execution (this is not an attack since participants do not compute group keys synchronously). Similarly, the condition  $\text{sid}_U^s \notin \Psi$  prevents that  $\mathcal{A}$  while being in the **attack** stage outputs  $(s, U)$  such that  $\Pi_U^s$  has accepted with  $\tilde{k}$  already in the **prepare** stage. Finally, since in every session id is unique,  $\text{sid}_U^s \notin \Psi$  holds if at least one new session has been executed with  $\Pi_U^s$  in the **attack** stage. The second requirement allows  $\mathcal{A}$  to corrupt at most  $n - 1$  (out of totally  $n$ ) participants in the session where  $\Pi_U^s$  accepts with  $\tilde{k}$ .

Note also that  $U$  must be honest but  $\mathcal{A}$  is allowed to reveal the internal state of  $\Pi_U^s$  during the execution of the **attack** stage (this is because our model separates  $LL_U$  from  $\text{state}_U^s$ ). The goal of the adversary is to influence the honest participants to accept the chosen key. Our game appears stronger than [4] since the adversary can open honest users' internal state (furthermore he can make corruptions in an *adaptive* manner).

*Remark 2.* The main difference to the non-malleability definition from [18] is that we allow  $\mathcal{A}$  to open honest users during the attacked session, however, at the cost that we do not deal with the ability of  $\mathcal{A}$  to bias the probability distribution of the resulting group key (similar to [4]). It seems to be hard to achieve this goal if  $\mathcal{A}$  corrupts  $n - 1$  users and opens the last  $n$ -th honest user, which is allowed by our definition. At least, the commitment techniques used in [18] would not help since committed secrets can be revealed.

### 3 Tree Diffie-Hellman Protocol with Strong Security

In this section we present our constant-round GKE protocol denoted TDH1 and show that it satisfies the strong versions of AKE-, MA-security, and contributiveness. Its AKE-security relies on the Tree Decisional Diffie-Hellman (TDDH) assumption, introduced by Kim *et al.* in [23, 24].

#### 3.1 Number-Theoretic Assumptions

First, we formally specify the TDDH assumption and quantify the reduction to the classical Decisional Diffie-Hellman (DDH) assumption [5]. Our protocol and those (unauthenticated) in [23, 24] require a special multiplicative group  $\mathbb{G}$  in which DDH is assumed to be hard *and* for which there exists an efficient bijection<sup>3</sup> from  $\mathbb{G}$  to  $\mathbb{Z}_{|\mathbb{G}|}$ . Thus, not every DDH-hard group can be used, e.g., no such bijection is known for elliptic curves.

<sup>3</sup> The exponentiation  $x \mapsto g^x$  is a bijection from  $\mathbb{Z}_{|\mathbb{G}|}$  to  $\mathbb{G}$ . We require that there exists an efficiently computable (bijective) mapping in the opposite direction, but we do NOT require this mapping to be the discrete logarithm!

**Algebraic Group** Let  $p$  be a safe prime, i.e.,  $p = 2q + 1$ , with  $q$  a  $\kappa$ -bit prime. The set of quadratic residue modulo  $p$  is a cyclic group  $\mathbb{G}$  of order  $q$ ; let  $g$  be a generator:  $\hat{\mathbb{G}} = \langle g \rangle$ . Consider the following mapping  $u : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q$  defined as

$$u(z) := \begin{cases} z \pmod q & \text{if } z \leq q \\ (p - z) \pmod q & \text{if } q < z < p \end{cases}$$

Consider the set  $\mathbb{G} := \{u(g^i) \mid i \in \mathbb{Z}_q\}$ . It can be shown [24] that the function  $f : x \mapsto u(g^x)$  from  $\mathbb{Z}_q$  to  $\mathbb{G}$  is a bijection, and that the operation  $(a, b) \in \mathbb{G}^2 \mapsto u(ab \pmod p)$  is a group law on  $\mathbb{G}$ . Since  $\mathbb{G} = \mathbb{Z}_q$  (as sets), we can define the exponentiation  $a^b := u(a^b \pmod p)$  for all  $a, b \in \mathbb{G}$ . Also, due to the fact that  $f$  is a bijection, we have  $f(x)$  is random, uniform in  $\mathbb{G}$  as soon as  $x$  is so.

Finally, it is believed [5] that the DDH assumption holds in  $\mathbb{G}$ , that is, the distributions of  $(g^{x_1}, g^{x_2}, g^{x_1 x_2})$  and  $(g^{x_1}, g^{x_2}, g^r)$  are computationally indistinguishable for  $x_1, x_2, r \in_R \mathbb{G}$ .

**Tree Decisional Diffie-Hellman Assumption** Let  $\mathcal{T}_n$  be the set of all full<sup>4</sup> binary trees with  $n$  leaves. For a  $T_n \in \mathcal{T}_n$  of depth  $d_{T_n}$ , each node is identified via a label  $\langle l, v \rangle$ , where  $l \in [0, d_{T_n}]$  is the level of the node and  $v$  its position within this level: the position is such that the child nodes of  $\langle l, v \rangle$  (if present) are labeled  $\langle l + 1, 2v \rangle$  and  $\langle l + 1, 2v + 1 \rangle$  (this implies that the nodes positions are in  $[0, 2^l - 1]$ , but may be not contiguous). The root node is labeled  $\langle 0, 0 \rangle$ . In the following we will denote  $T_n \setminus \langle 0, 0 \rangle$  by  $T_n^*$ . The set of leaf nodes and the set of internal nodes of  $T_n$  are defined as (respectively):

$$LN_{T_n} := \{\langle l, v \rangle \mid \langle l, v \rangle \in T_n, \langle l + 1, 2v \rangle \notin T_n, \langle l + 1, 2v + 1 \rangle \notin T_n\},$$

$$IN_{T_n} := \{\langle l, v \rangle \mid \langle l, v \rangle \in T_n^*, \langle l + 1, 2v \rangle \in T_n, \langle l + 1, 2v + 1 \rangle \in T_n\}.$$

For a set  $X$  of  $n$  randomly chosen variables  $x_{\langle l, v \rangle} \in_R \mathbb{G}$ , with  $\langle l, v \rangle \in LN_{T_n}$ , we (recursively) define for each  $\langle l, v \rangle \in IN_{T_n}$ :

$$x_{\langle l, v \rangle} = g^{x_{\langle l+1, 2v \rangle} x_{\langle l+1, 2v+1 \rangle}} \quad \text{and} \quad \text{TDH}_{T_n}(X) = \left\{ \left( \langle l, v \rangle, g^{x_{\langle l, v \rangle}} \right) \right\}_{\langle l, v \rangle \in T_n^*}$$

Additionally, for a randomly chosen  $r \in_R \mathbb{G}$ , we define the tuples  $\text{TDDH}_{T_n}^*(X)$  and  $\text{TDDH}_{T_n}^{\$}(X)$  as follows:

$$\text{TDDH}_{T_n}^*(X) = \text{TDH}_{T_n}(X) \cup (\langle 0, 0 \rangle, g^{x_{\langle 1, 0 \rangle} x_{\langle 1, 1 \rangle}}) \quad \text{and} \quad \text{TDDH}_{T_n}^{\$}(X, r) = \text{TDH}_{T_n}(X) \cup (\langle 0, 0 \rangle, g^r)$$

The TDDH assumption states that the respective distributions of these tuples induced by uniform choices of  $r$  and the  $x_{\langle l, v \rangle}$ , for  $\langle l, v \rangle \in LN_{T_n}$  are computationally indistinguishable.

**Definition 6.** (TDDH Assumption) For all  $n > 1$ , any  $T_n \in \mathcal{T}_n$ , any group  $\mathbb{G}$ , and any PPT algorithm  $\mathcal{A}$ , the distinguishing advantage  $\text{Adv}_{T_n, \mathbb{G}}^{\text{TDDH}}(\mathcal{A})$ , defined as follows, is negligible (in  $\kappa = \log |\mathbb{G}|$ ):

$$\left| \Pr_X \left[ \mathcal{A}(\text{TDDH}_{T_n}^*(X)) = 1 \right] - \Pr_{X, r} \left[ \mathcal{A}(\text{TDDH}_{T_n}^{\$}(X, r)) = 1 \right] \right|$$

**Theorem 1** (DDH  $\iff$  TDDH). The TDDH problem in  $\mathbb{G}$  is polynomially equivalent to the DDH problem in  $\mathbb{Z}_q$ , and we have:

$$\text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa) \leq \text{Adv}_{T_n, \mathbb{G}}^{\text{TDDH}}(\kappa) \leq (2n - 3) \text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa).$$

The proof appears in Appendix A and is more general than those in [23, 24] that focus only on complete and linear binary trees. For  $n = 2$  we get the classical DDH assumption in  $\mathbb{G}$  with the advantage

$$\text{Adv}_{\mathbb{G}}^{\text{DDH}}(\mathcal{A}') := \left| \Pr_{x_{\langle 1, 0 \rangle}, x_{\langle 1, 1 \rangle}} \left[ \mathcal{A}'(\text{DDH}^*((\langle 1, 0 \rangle, g^{x_{\langle 1, 0 \rangle}}), (\langle 1, 1 \rangle, g^{x_{\langle 1, 1 \rangle}}), (\langle 0, 0 \rangle, g^{x_{\langle 1, 0 \rangle} x_{\langle 1, 1 \rangle}})) = 1 \right] - \Pr_{x_{\langle 1, 0 \rangle}, x_{\langle 1, 1 \rangle}, r} \left[ \mathcal{A}'(\text{DDH}^{\$}((\langle 1, 0 \rangle, g^{x_{\langle 1, 0 \rangle}}), (\langle 1, 1 \rangle, g^{x_{\langle 1, 1 \rangle}}), (\langle 0, 0 \rangle, r)) = 1 \right] \right|.$$

<sup>4</sup> A binary tree is called *full* if each of its nodes has exactly 0 or 2 children. Sometimes such trees are also called *proper*.

### 3.2 Light Description of TDH1

The main mechanism of the protocol is that of [23, 24], so we first recall it. The differences will be in message authentication, key derivation, and applied erasure technique to prevent the ephemeral session secrets from being leaked once the session is finished.

**The Setup Operation** Every oracle is assigned to a leaf node of a so-called *linear* binary tree  $T_n$ : a full binary tree with one leaf at each level, except for the deepest one with two leaves (see Figure 1). In other words  $T_n^* := \{\langle l, v \rangle\}_{l \in [1, n-1], v \in [0, 1]}$ .

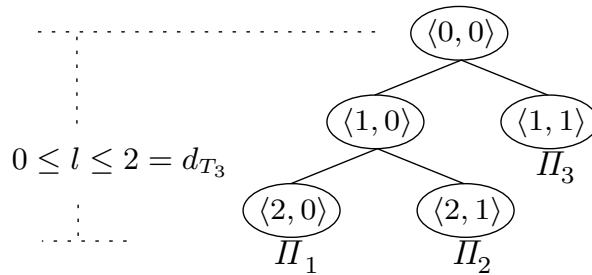
**Round 1 – All.** Each oracle at position  $\langle l_i, v_i \rangle$  chooses a secret exponent  $x_{\langle l_i, v_i \rangle} \in_R \mathbb{G}$  and broadcasts  $y_{\langle l_i, v_i \rangle} := g^{x_{\langle l_i, v_i \rangle}}$ .

**Round 2 – First player.**  $\Pi_1$  at position  $\langle n-1, 0 \rangle$  is able to build a set  $X_1$  of secret values for each node  $x_{\langle l, 0 \rangle}$  in its path up to the root  $\langle 0, 0 \rangle$ . This is because for each internal node  $x_{\langle l, 0 \rangle} = (y_{\langle l+1, 1 \rangle})^{x_{\langle l+1, 0 \rangle}}$ . Then,  $\Pi_1^s$  computes the set  $\hat{Y}$  consisting of  $y_{\langle l, 0 \rangle} := g^{x_{\langle l, 0 \rangle}}$  for each previously computed internal node's secret value  $x_{\langle l, 0 \rangle}$  *except for*  $x_{\langle 0, 0 \rangle}$ , and broadcasts  $\hat{Y}$ .

**Round 3 – No communication.** Upon receiving  $\hat{Y}$ , all other oracles  $\Pi_{i \neq 1}$  are able to compute their own set  $X_i$  consisting of all secret values  $x_{\langle l, v \rangle}$  in their paths up to the root. Hence, every oracle finally learns  $x_{\langle 0, 0 \rangle}$ . We emphasize that  $x_{\langle 0, 0 \rangle}$  is never exposed, and that there is no  $y_{\langle 0, 0 \rangle}$  in the protocol (see description of function  $\text{TDH1\_Exp}^*(l, X)$  below).

**Group Key Confirmation and Derivation** To derive the session group key  $\mathbf{K}$ , each participant iteratively computes a sequence of values  $\rho_0, \dots, \rho_n$  using a pseudo-random function (PRF)  $f$  with a public value  $v_0$  as input. The key (secret seed) of  $f$  is initially set to  $x_{\langle 0, 0 \rangle}$ , and is changed in each invocation of  $f$  by embedding successive nonces using an appropriate one-way permutation  $\pi$ . These nonces are provided by participants during the protocol execution.

Intuitively, these successive evaluations of  $f$  and  $\pi$  prevent malicious participants from influencing values of the PRF keys and ensures contributiveness for the intermediate value  $\rho_n$ , which is then used as a seed for  $f$  to derive the key confirmation token  $\mu$  (on input a constant public value  $v_1$ ) and the actual session group key  $\mathbf{K}$  (on input another constant public value  $v_2 \neq v_1$ ). Prior to accepting  $\mathbf{K}$ : (i) participants exchange and verify signatures on  $\mu$  to ensure MA-security (similar to [20]) and (ii) erase [17] all ephemeral secrets, used to obtain  $\mathbf{K}$  from their internal states, to achieve strong AKE-security.



**Fig. 1.** Example of a Linear Binary Tree  $T_3$  for the Group  $\mathcal{G} = \{\Pi_1, \Pi_2, \Pi_3\}$ .

### 3.3 Detailed Description of TDH1

**Preliminary Notations** We assume that long-lived keys  $LL_i = (sk_i, pk_i)$  are generated via  $\Sigma.\text{Gen}(1^\kappa)$ , where  $\Sigma = (\text{Gen}, \text{Sign}, \text{Verify})$  is an existentially unforgeable (under chosen message attacks) digital signature scheme. We define the following key exchange functions:

**TDH1\_Exp**( $x_{\langle l,v \rangle}$ ): Simple exponentiation. The function returns  $y_{\langle l,v \rangle} := g^{x_{\langle l,v \rangle}}$ .

**TDH1\_Pick**( $1^\kappa$ ): Generation of the ephemeral secret exponent and the corresponding public value. The function picks  $x_{\langle l,v \rangle} \in_R \mathbb{G}$ , computes  $y_{\langle l,v \rangle} := \text{TDH1\_Exp}(x_{\langle l,v \rangle})$  and returns  $(x_{\langle l,v \rangle}, y_{\langle l,v \rangle})$ .

**TDH1\_Exp\***( $l, X$ ) where  $X = \{x_{\langle j,0 \rangle}\}_{1 \leq j \leq l}$ : Computation of corresponding public values for secret exponents in  $X$ . The function returns  $Y := \{y_{\langle j,0 \rangle}\}$  where each  $y_{\langle j,0 \rangle} := \text{TDH1\_Exp}(x_{\langle j,0 \rangle})$ .

**TDH1\_Up**( $l, v, x_{\langle l,v \rangle}, y_{\langle l,1-v \rangle}, Y$ ) where  $Y := \{y_{\langle j,1 \rangle} \mid j \in [1, l-1]\}$ : Iterative computation of the Diffie-Hellman values up the tree starting at position  $\langle l, v \rangle$ . The function computes  $x_{\langle l-1,0 \rangle} := (y_{\langle l,1-v \rangle})^{x_{\langle l,v \rangle}}$ , and returns

$$X := \{x_{\langle l,v \rangle}, x_{\langle l-1,0 \rangle}\} \cup \{x_{\langle j,0 \rangle} := (y_{\langle j+1,1 \rangle})^{x_{\langle j+1,0 \rangle}} \mid y_{\langle j+1,1 \rangle} \in Y, \forall j = l-2, \dots, 0\}.$$

Let  $F := \{\{f_k\}_{k \in \{0,1\}^\kappa}\}_{\kappa \in \mathbb{N}}$  be a collision-resistant pseudo-random function ensemble with domain and range  $\{0,1\}^\kappa$  (see [20] or Appendix B for definition). Let  $\pi : \{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$  be a one-way permutation. We denote by  $v_0, v_1$  and  $v_2$  three distinct, public values in  $\{0,1\}^\kappa$ . The following function is used to compute the *intermediate value*  $K$  and the *key confirmation token*  $\mu$ .

**TDH1\_Con**( $x_{\langle 0,0 \rangle}, r_1 | \dots | r_n$ ): The function computes  $\rho_0 := f_{x_{\langle 0,0 \rangle}}(v_0)$ , and each  $\rho_l := f_{\rho_{l-1} \oplus \pi(r_l)}(v_0)$  for all  $l = \{1, \dots, n\}$ . Let  $K := \rho_n$ . Finally, the function computes  $\mu := f_K(v_1)$ , and returns  $(K, \mu)$ .

**The Protocol TDH1.Setup** In the following we assume that an oracle aborts without accepting if any performed check fails.

**Round 1.** Given the tree structure  $T_n$ , each oracle  $\Pi_i$  proceeds as follows:

- Pick at random nonce  $r_i \in_R \{0,1\}^\kappa$ ;
- Invoke **TDH1\_Pick**( $1^\kappa$ ) to generate a secret exponent  $x_{\langle l_i, v_i \rangle}$  and the value  $y_{\langle l_i, v_i \rangle} = g^{x_{\langle l_i, v_i \rangle}}$ ;
- Invoke  $\Sigma$ .**Sign** to obtain a signature  $\sigma_i$  on  $0|y_{\langle l_i, v_i \rangle}|r_i|\text{pid}_i$  using the private key  $sk_i$ ;
- Broadcast  $U_i|0|y_{\langle l_i, v_i \rangle}|r_i|\sigma_i$ .

**Round 2.** Each oracle  $\Pi_i$  proceeds as follows:

- Check if  $\Sigma$ .**Verify**( $pk_j, 0|y_{\langle l_j, v_j \rangle}|r_j|\text{pid}_j, \sigma_j$ ) = 1 for  $j \neq i$ ; Check if  $|r_j| = \kappa$  for  $j \neq i$ ;
- Define  $\text{sid}_i := r_1 | \dots | r_n$  and  $Y_i := \{y_{\langle l, 1 \rangle}\}_{l=i-1, \dots, 1}$ .

In addition,  $\Pi_1$  does the following:

- Compute  $X_1 := \text{TDH1\_Up}(n-1, 0, x_{\langle n-1,0 \rangle}, y_{\langle n-1,1 \rangle}, Y_1)$  and  $\hat{Y} := \text{TDH1\_Exp}^*(n-2, X_1)$ ;
- Invoke  $\Sigma$ .**Sign** to obtain a signature  $\sigma'_1$  on  $1|\hat{Y}|\text{sid}_1|\text{pid}_1$  using the private key  $sk_1$ ;
- Broadcast  $U_1|1|\hat{Y}|\sigma'_1$ .

**Round 3.** Each  $\Pi_i$  with  $i > 1$  proceeds as follows:

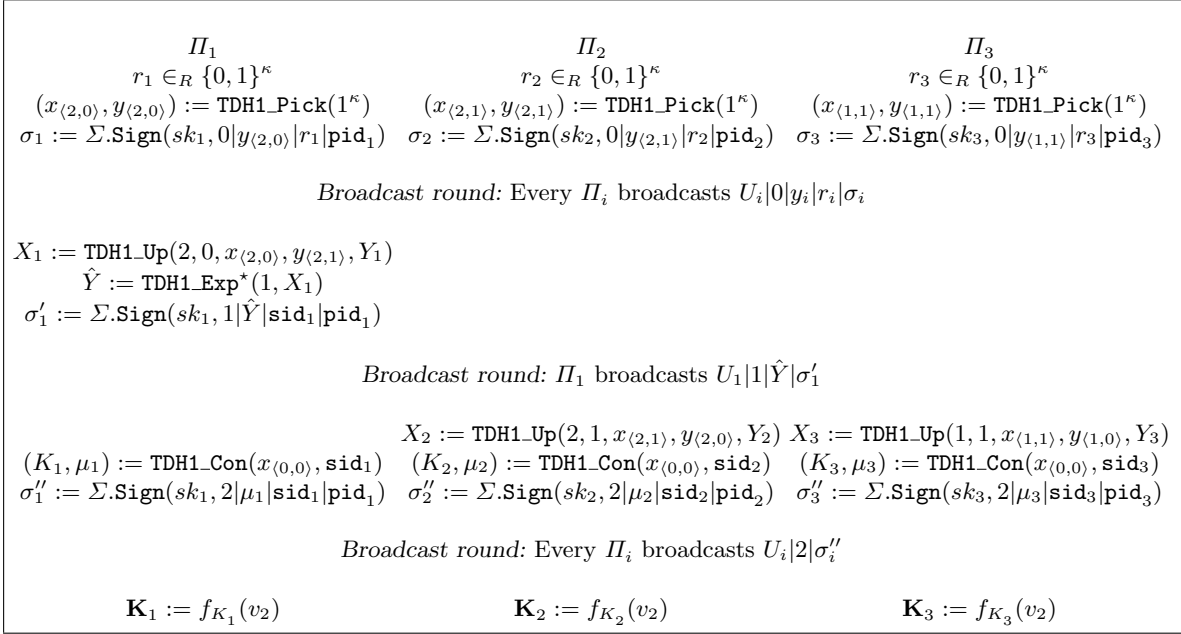
- Check if  $\Sigma$ .**Verify**( $pk_1, 1|\hat{Y}|\text{sid}_i|\text{pid}_i, \sigma'_1$ ) = 1;
- Compute  $X_i$  as  $\text{TDH1\_Up}(l_i, v_i, x_{\langle l_i, v_i \rangle}, y_{\langle l_i, 1-v_i \rangle}, Y_i)$ .

Then every oracle (including  $\Pi_1$ ) does the following:

- Compute both  $K_i$  and  $\mu_i$  using **TDH1\_Con**( $x_{\langle 0,0 \rangle}, \text{sid}_i$ ); [note that  $x_{\langle 0,0 \rangle} \in X_i$ ]
- Erase any private information from  $\text{state}_i$  (including all  $x_{\langle l,v \rangle}$ , and  $\rho_0, \dots, \rho_n$ ) except for  $K_i$ ;
- Invoke  $\Sigma$ .**Sign** to obtain a signature  $\sigma''_i$  on  $2|\mu_i|\text{sid}_i|\text{pid}_i$  using the private key  $sk_i$ ;
- Broadcast  $U_i|2|\sigma''_i$ .

**Group Key Computation.** When  $\Pi_i$  receives  $U_j|2|\sigma''_j$  from all other oracles, it proceeds as follows:

- Check if  $\Sigma$ .**Verify**( $pk_j, 2|\mu_i|\text{sid}_i|\text{pid}_i, \sigma''_j$ ) = 1; Compute  $\mathbf{K}_i := f_{K_i}(v_2)$ ;



**Fig. 2.** Example of Operation `TDH1.Setup` with  $\mathcal{G}=\{\Pi_1, \Pi_2, \Pi_3\}$ . Public values:  $\text{sid}_i = r_1|r_2|r_3$ ,  $Y_1=Y_2=\{y_{\langle 1,1 \rangle}\}$ ,  $Y_3=\emptyset$ ,  $\hat{Y}=\{y_{\langle 1,0 \rangle}\}$ , where  $y_{\langle 1,1 \rangle}=g^{x_{\langle 1,1 \rangle}}$ ,  $y_{\langle 1,0 \rangle}=g^{x_{\langle 1,0 \rangle}}$ . Secret values:  $X_1=\{x_{\langle 2,0 \rangle}, x_{\langle 1,0 \rangle}, x_{\langle 0,0 \rangle}\}$ ,  $X_2=\{x_{\langle 2,1 \rangle}, x_{\langle 1,0 \rangle}, x_{\langle 0,0 \rangle}\}$ ,  $X_3=\{x_{\langle 1,1 \rangle}, x_{\langle 0,0 \rangle}\}$ , where  $x_{\langle 0,0 \rangle}=g^{x_{\langle 1,0 \rangle}x_{\langle 1,1 \rangle}}$ ,  $x_{\langle 1,0 \rangle}=g^{x_{\langle 2,0 \rangle}x_{\langle 2,1 \rangle}}$  and  $x_{\langle 2,0 \rangle}, x_{\langle 2,1 \rangle}, x_{\langle 1,1 \rangle} \in_R \mathbb{G}$ .

- Erase any private information from  $\text{state}_i$  (including  $K_i$ ), and accept with  $\mathbf{K}_i$ .

Figure 2 sketches the execution of `TDH1.Setup` for three participants (necessary checks and erasure steps are omitted). In this example, oracle  $\Pi_1$  builds  $X_1 = \{x_{\langle 2,0 \rangle}, x_{\langle 1,0 \rangle}, x_{\langle 0,0 \rangle}\}$  with  $x_{\langle 1,0 \rangle} := (y_{\langle 2,1 \rangle})^{x_{\langle 2,0 \rangle}}$  and  $x_{\langle 0,0 \rangle} := (y_{\langle 1,1 \rangle})^{x_{\langle 1,0 \rangle}}$  (remember that  $y_{\langle 2,1 \rangle}$  and  $y_{\langle 1,1 \rangle}$  were received in the previous round). The set broadcasted by  $\Pi_1$  is  $\hat{Y} = \{y_{\langle 1,0 \rangle}\}$ , where  $y_{\langle 1,0 \rangle} := g^{x_{\langle 1,0 \rangle}}$ . In the third round oracle  $\Pi_2$  computes  $X_2 = \{x_{\langle 2,1 \rangle}, x_{\langle 1,0 \rangle}, x_{\langle 0,0 \rangle}\}$ , where  $x_{\langle 1,0 \rangle} := (y_{\langle 2,0 \rangle})^{x_{\langle 2,1 \rangle}}$ , and  $x_{\langle 0,0 \rangle} := (y_{\langle 1,1 \rangle})^{x_{\langle 1,0 \rangle}}$ . In parallel, oracle  $\Pi_3$  computes  $X_3 = \{x_{\langle 1,1 \rangle}, x_{\langle 0,0 \rangle}\}$  where  $x_{\langle 0,0 \rangle} := (y_{\langle 1,0 \rangle})^{x_{\langle 1,1 \rangle}}$ . Hence, every oracle is in possession of  $x_{\langle 0,0 \rangle}$ . Finally, all three oracles can compute the intermediate value  $K$ , i.e.,  $\rho_0 := f_{x_{\langle 0,0 \rangle}|\kappa}(v_0)$ ,  $\rho_1 := f_{\rho_0 \oplus \pi(r_1)}(v_0)$ ,  $\rho_2 := f_{\rho_1 \oplus \pi(r_2)}(v_0)$ , and  $K = \rho_3 := f_{\rho_2 \oplus \pi(r_2)}(v_0)$ , and the key confirmation token  $\mu := f_K(v_1)$ . Note that the value  $x_{\langle 0,0 \rangle}$  is never sent over the public channel but computed locally by all participants upon receiving enough information. Furthermore, there exists no  $y_{\langle 0,0 \rangle}$  in the protocol.

### 3.4 Security and Performance of TDH1

In our security proofs following the specification of our model we consider that ephemeral secret information kept in  $\text{state}_U$  is always independent of the long-lived key  $sk_U$ . That is, in each session,  $\text{state}_U$  contains  $X_U$  consisting of all secrets  $x_{\langle l,v \rangle}$  known to  $\Pi_U$ ,  $\rho_0, \dots, \rho_n$  and possibly some (implementation specific) temporary variables used to compute these values. Furthermore, we assume that the signing algorithm  $\Sigma.\text{Sign}$  which implicitly uses  $sk_U$  is executed under the same protection mechanism as  $sk_U$ , e.g., in a smart card as in [9] (although smart cards have limited resources we observe that in `TDH1.Setup` each oracle has to generate at most three signatures). This is important since the signing algorithm may generate some randomness which should also be protected from being revealed via a `RevealState` query; otherwise the adversary may be able to obtain some information about  $sk_U$ .

The following theorems show that TDH1 satisfies the requirements of strong AKE-, MA-security and contributiveness; the last two also under consideration of insider attacks. In all theorems  $q_s$  is the total number of executed protocol sessions during the corresponding attack game.

### Strong AKE-Security of TDH1

**Theorem 2.** *If  $\Sigma$  is existentially unforgeable under chosen message attacks,  $F$  is pseudo-random, and  $\mathbb{G}$  is TDDH-hard then TDH1 provides strong AKE-security, and*

$$\text{Adv}_{\text{TDH1}}^{\text{ake}}(\kappa) \leq 2N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^{\kappa-1}} + 2q_s \text{Adv}_{T_N, \mathbb{G}}^{\text{TDDH}}(\kappa) + 2(N+3)q_s \text{Adv}_F^{\text{prf}}(\kappa).$$

*Proof.* (Sketch) We define a sequence of games  $\mathbf{G}_i$ ,  $i = 0, \dots, 7$  with the adversary  $\mathcal{A}$  against the AKE-security of TDH1. In each game we denote  $\text{Win}_i^{\text{ake}}$  the event that the bit  $b'$  output by  $\mathcal{A}$  is identical to the randomly chosen bit  $b$  in game  $\mathbf{G}_i$ .

**Game  $\mathbf{G}_0$ .** This game is the real game  $\text{Game}_{\mathcal{A}, \text{TDH1}}^{\text{ake}-b}(\kappa)$  where we use a *simulator*  $\Delta$  to simulate the execution of the protocol and answer all queries of  $\mathcal{A}$ .

Remind that the *Test* query is asked to a fresh oracle  $\Pi_i^s$  which has accepted, and that the adversary then receives either a random string or a session group key  $\mathbf{K}_i^s$ . Note that according to our definition of the oracle freshness, no *Corrupt* queries for any  $U_j \in \text{pid}_i^s$  and no *RevealState* queries to  $\Pi_i^s$  or to any of its partners can be asked prior to the *Test* query. Since the *Test* query is asked after  $\Pi_i^s$  has accepted (thus finished this session) and at the moment the *Test* query is asked  $\Pi_i^s$  is fresh we can say that the whole session where the *Test* query is asked in is fresh too.

**Game  $\mathbf{G}_1$ .** This game is identical to Game  $\mathbf{G}_0$  with the only exception that  $\Delta$  fails and bit  $b'$  is set at random if  $\mathcal{A}$  asks a *Send* query on some  $U_i | \text{seqn} | m | \sigma$ , with  $\text{seqn} \in \{0, 1, 2\}$  and  $\sigma$  a valid signature on  $m$ , that has not been previously output by an oracle  $\Pi_i^s$  before querying *Corrupt*( $U_i$ ) (note that *Corrupt* queries can be generally asked after the *Test* query).

In other words the simulation fails if  $\mathcal{A}$  outputs a successful forgery; such event is denoted *Forge*. A classical reductionist argument (see for instance [11]) shows that in that case we can build a forger against the signature scheme and upper-bound the probability distance as

$$|\Pr[\text{Win}_1^{\text{ake}}] - \Pr[\text{Win}_0^{\text{ake}}]| \leq N \text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa).$$

**Game  $\mathbf{G}_2$ .** This game is identical to Game  $\mathbf{G}_1$  except that  $\Delta$  fails if two instances of an honest user used the same nonce twice. Since there are  $N$  users and at most  $q_s$  sessions we get

$$|\Pr[\text{Win}_2^{\text{ake}}] - \Pr[\text{Win}_1^{\text{ake}}]| \leq \frac{Nq_s^2}{2^{\kappa}}.$$

This game excludes replay attacks in operations of TDH1 ensuring the uniqueness of  $\text{sid}_i^s$  which is part of each signed message, and since the previous game excludes forgeries.

**Game  $\mathbf{G}_3$ .** In this game we add the following rule:  $\Delta$  chooses  $q_s^* \in [1, q_s]$  and aborts if the *Test* query does not occur in the  $q_s^*$ -th session. Let  $\mathbf{Q}$  be the event that this guess for  $q_s^*$  is correct and  $\Pr[\mathbf{Q}] = 1/q_s$ . Thus, we get

$$\begin{aligned} \Pr[\text{Win}_3^{\text{ake}}] &= \Pr[\text{Win}_3^{\text{ake}} | \mathbf{Q}] \Pr[\mathbf{Q}] + \Pr[\text{Win}_3^{\text{ake}} | \neg \mathbf{Q}] \Pr[\neg \mathbf{Q}] \\ &= \Pr[\text{Win}_2^{\text{ake}}] \frac{1}{q_s} + \frac{1}{2} \left( 1 - \frac{1}{q_s} \right). \end{aligned}$$

This implies,

$$\Pr[\text{Win}_2^{\text{ake}}] = q_s \left( \Pr[\text{Win}_3^{\text{ake}}] - \frac{1}{2} \right) + \frac{1}{2}.$$

**Game G<sub>4</sub>.** This game is identical to Game G<sub>3</sub> except that  $\Delta$  is given a tuple from the real  $\text{TDDH}_{T_N}^*$  distribution (as specified in Section 3.1) where  $T_N$  is a linear tree. In all sessions except the  $q_s^*$ -th one,  $\Delta$  simulates the honest participants as specified by the protocol. In the  $q_s^*$ -th session with  $n$  participants  $\Delta$  injects public values  $g^{x_{\langle l,v \rangle}}$  from  $\text{TDDH}_{T_N}^*$  into the protocol execution. Note that in the  $q_s^*$ -th session the group size  $n$  might be smaller than  $N$ , thus the simulator will use the subtree  $T_n$  which is composed of  $T_N$ 's nodes from level 0 to level  $n - 1$ . The idea is to assign  $\Pi_1^s$  to the (internal) node  $\langle n - 1, 0 \rangle$ ,  $\Pi_2^s$  to the leaf node  $\langle n - 1, 1 \rangle$ ,  $\dots$ ,  $\Pi_n^s$  to the leaf node  $\langle 1, 1 \rangle$ , and to use for each node  $\langle l, v \rangle \in T_n \setminus \langle 0, 0 \rangle$  public values  $g^{x_{\langle l,v \rangle}}$  taken from the given  $\text{TDDH}_{T_N}^*$  distribution. The secret value  $x_{\langle 0,0 \rangle}$  used for the key confirmation is also taken from this distribution. Since the  $q_s^*$ -th session is fresh, no *RevealState* queries to  $\Pi_i^s$  or to any of its partners have been asked ( $\Delta$  would not be able to answer them since it does not know the secret values  $x_{\langle l,v \rangle}$  of internal and leaf nodes). Of course in all other sessions *RevealState* queries can be easily answered. Since  $\text{TDDH}_{T_N}^*$  is a real distribution we conclude that this game is a “bridging step” (as named in [32]) and

$$\Pr[\text{Win}_4^{\text{ake}}] = \Pr[\text{Win}_3^{\text{ake}}].$$

**Game G<sub>5</sub>.** This game is identical to Game G<sub>4</sub> except that  $\Delta$  is given a tuple from the random  $\text{TDDH}_{T_N}^s$  distribution. Thus, for honest players, the secret  $x_{\langle 0,0 \rangle}$  is simulated using the provided random element  $g^r$ . Obviously,

$$|\Pr[\text{Win}_5^{\text{ake}}] - \Pr[\text{Win}_4^{\text{ake}}]| \leq \text{Adv}_{T_N, \mathbb{G}}^{\text{TDDH}}(\kappa).$$

**Game G<sub>6</sub>.** This game is identical to Game G<sub>5</sub> except that in the  $q_s^*$ -th session  $\Delta$  replaces  $f$  by a truly random function, implying the uniform distribution of  $K = \rho_n$ . Considering  $n \leq N$  we obtain by a “hybrid argument”<sup>5</sup>

$$|\Pr[\text{Win}_6^{\text{ake}}] - \Pr[\text{Win}_5^{\text{ake}}]| \leq (N + 1)\text{Adv}_F^{\text{prf}}(\kappa).$$

**Game G<sub>7</sub>.** This is the continuation of the hybrid argument, but for clarity we specify a separate game; the confirmation token  $\mu$  and the session key  $\mathbf{K}$  are replaced by two random  $\kappa$ -bit values, s.t.,  $|\Pr[\text{Win}_7^{\text{ake}}] - \Pr[\text{Win}_6^{\text{ake}}]| \leq 2\text{Adv}_F^{\text{prf}}(\kappa)$ . Since  $\mathbf{K}$  is uniform:  $\Pr[\text{Win}_7^{\text{ake}}] = 1/2$ . Combining the previous equations, we conclude the proof.  $\square$

### Strong MA-Security of TDH1

**Theorem 3.** *If  $\Sigma$  is existentially unforgeable under chosen message attacks and  $F$  is collision-resistant then TDH1 provides strong MA-security, and*

$$\text{Succ}_{\text{TDH1}}^{\text{ma}}(\kappa) \leq N\text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa) + \frac{Nq_s^2}{2^\kappa} + q_s\text{Succ}_F^{\text{coll}}(\kappa).$$

*Proof.* (Sketch) We define a sequence of games  $\mathbf{G}_i$ ,  $i = 0, \dots, 2$  and corresponding events  $\text{Win}_i^{\text{ma}}$  meaning that  $\mathcal{A}$  wins in  $\mathbf{G}_i$ .

**Game G<sub>0</sub>.** This is the real game  $\text{Game}_{\text{TDH1}}^{\text{ma}}(\kappa)$  played between a simulator  $\Delta$  and  $\mathcal{A}$ . The goal of  $\mathcal{A}$  is to achieve that there exists an honest user  $U_i$  whose corresponding oracle  $\Pi_i^s$  accepts with  $\mathbf{K}_i^s$  and another user  $U_j \in \text{pid}_i^s$  who is uncorrupted at the time  $\Pi_i^s$  accepts and either does not have a corresponding oracle  $\Pi_j^t$  with  $(\text{pid}_j^t, \text{sid}_j^t) = (\text{pid}_i^s, \text{sid}_i^s)$  or has such an oracle but this oracle accepts with  $\mathbf{K}_j^t \neq \mathbf{K}_i^s$ .

**Game G<sub>1</sub>.** In this game we proceed as in the previous proof and eliminate executions in which forgeries occur, obtaining

$$|\Pr[\text{Win}_1^{\text{ma}}] - \Pr[\text{Win}_0^{\text{ma}}]| \leq N\text{Succ}_{\Sigma}^{\text{euf-cma}}(\kappa).$$

<sup>5</sup> More precisely, one constructs  $n + 1$  auxiliary “hybrid games”  $\mathbf{G}_{6,l}$ ,  $l = 0, \dots, n$  and replaces in each game  $\rho_l$  by a random value from  $\{0, 1\}^\kappa$ . The difference between two neighbor hybrids is upper-bounded by the PRF advantage.

**Game G<sub>2</sub>.** This game is identical to Game G<sub>1</sub> except that  $\Delta$  fails if a nonce  $r_i$  is used by any uncorrupted user's oracle  $\Pi_i^s$  in two different sessions. Similar to the previous proof we get

$$|\Pr[\text{Win}_2^{\text{ma}}] - \Pr[\text{Win}_1^{\text{ma}}]| \leq \frac{Nq_s^2}{2^\kappa}.$$

Having excluded forgeries and replay attacks we follow that for every user  $U_j \in \text{pid}_i^s$  that is uncorrupted at the time  $\Pi_i^s$  accepts there exists a corresponding instance oracle  $\Pi_j^t$  with  $(\text{pid}_j^t, \text{sid}_j^t) = (\text{pid}_i^s, \text{sid}_i^s)$ . Thus, according to Definition 4  $\mathcal{A}$  wins in this game only if any of these oracles has accepted with  $\mathbf{K}_j^t = f_{K_j^t}(v_2) \neq f_{K_i^s}(v_2) = \mathbf{K}_i^s$ . However the validity of signatures on tokens  $\mu_i$  and  $\mu_j$  implies that  $\mu_i = \mu_j$ . Thus the probability that the adversary wins in this last game is upper-bounded by

$$q_s \Pr[\mathbf{K}_j^t \neq \mathbf{K}_i^s \wedge f_{K_j^t}(v_1) = f_{K_i^s}(v_1)] = q_s \Pr[f_{K_j^t}(v_2) \neq f_{K_i^s}(v_2) \wedge f_{K_j^t}(v_1) = f_{K_i^s}(v_1)] \leq q_s \text{Succ}_F^{\text{coll}}(\kappa).$$

Combining the previous equations we get the desired result.  $\square$

### Strong Contributiveness of TDH1

**Theorem 4.** *If  $\pi$  is one-way and  $F$  is collision-resistant pseudo-random then TDH1 provides strong contributiveness, and*

$$\text{Succ}_{\text{TDH1}}^{\text{con}}(\kappa) \leq \frac{Nq_s^2 + Nq_s + 2q_s}{2^\kappa} + (N + 2)q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Adv}_F^{\text{prf}}(\kappa) + Nq_s \text{Succ}_\pi^{\text{ow}}(\kappa).$$

Note that in TDH1 the adversary is able to enforce the resulting value for  $x_{\langle 0,0 \rangle}$  by opening oracles of honest users during the protocol execution. More precisely,  $\mathcal{A}$  can enforce that the same  $x_{\langle 0,0 \rangle}$  is computed by the oracles of some uncorrupted user in two different sessions. To show this, assume for simplicity three participants:  $\Pi_1^s$ ,  $\Pi_2^s$ , and  $\Pi_3^s$ , and consider that  $\Pi_1^s$  and  $\Pi_3^s$  are malicious (corrupted) whereas  $\Pi_2^s$  is honest. We consider two different sessions: session A and session B, whereby session B takes place later than A. In both sessions the tree is as in Figure 1. Assume that in session A all oracles behave as specified in the protocol except that neither  $\Pi_1^s$  nor  $\Pi_3^s$  erase their states. At some point before session B is started, the adversary  $\mathcal{A}$  (that can impersonate  $\Pi_1^s$  and  $\Pi_3^s$ ) computes  $z := x_{\langle 1,0 \rangle} x_{\langle 1,1 \rangle}$  where  $x_{\langle 1,0 \rangle}$  is a value computed by  $\Pi_1^s$  and  $x_{\langle 1,1 \rangle}$  is the exponent chosen by  $\Pi_3^s$ , both in session A. Obviously,  $g^z$  equals to  $x_{\langle 0,0 \rangle}$  computed in session A. The goal of  $\mathcal{A}$  is to influence honest  $\Pi_2^{s'}$  to compute the same  $x_{\langle 0,0 \rangle}$  in session B. In session B, the exponent  $x_{\langle 2,1 \rangle}$  used by honest  $\Pi_2^{s'}$  is likely to be different compared to session A. To proceed with the attack  $\mathcal{A}$  waits for  $\Pi_2^{s'}$  to broadcast  $y_{\langle 2,1 \rangle} = g^{x_{\langle 2,1 \rangle}}$  in session B (note the communication is asymmetric). Then, the adversary *opens* the oracle holding node  $x_{\langle 2,1 \rangle}$  (via the *RevealState*( $\Pi_2^{s'}$ ) query); chooses on behalf of  $\Pi_1^{s'}$   $x_{\langle 2,0 \rangle}$  truly at random, computes  $x_{\langle 1,0 \rangle} := g^{x_{\langle 2,0 \rangle} x_{\langle 2,1 \rangle}}$  and  $x_{\langle 1,1 \rangle} := z/x_{\langle 1,0 \rangle}$ . To complete the attack,  $\mathcal{A}$  broadcasts  $y_{\langle 2,0 \rangle} := g^{x_{\langle 2,0 \rangle}}$  and  $y_{\langle 1,1 \rangle} := g^{x_{\langle 1,1 \rangle}}$  on behalf of  $\Pi_1^{s'}$  and  $\Pi_3^{s'}$ , respectively. It is easy to check that  $\Pi_2^{s'}$  computes  $x_{\langle 0,0 \rangle} = g^z$  in session B.

In our proof of Theorem 4 we show that despite of being able to enforce  $x_{\langle 0,0 \rangle}$  the adversary is still unable to enforce the resulting session group key  $\mathbf{K}$ . We make use of the following ‘‘difference lemma’’.

**Lemma 1.** *Let A, B, C be events defined in some probability distribution, and suppose that  $\Pr[\text{B}] = \Pr[\text{A}|\text{C}]$ . Then*

$$\Pr[\text{A}] - \Pr[\text{B}] \leq \Pr[\neg\text{C}].$$

The proof follows from the equation:

$$\begin{aligned} \Pr[\text{A}] &= \Pr[\text{A}|\text{C}] \Pr[\text{C}] + \Pr[\text{A}|\neg\text{C}] \Pr[\neg\text{C}] \\ &= \Pr[\text{B}] \Pr[\text{C}] + \Pr[\text{A}|\neg\text{C}] \Pr[\neg\text{C}] \\ &\leq \Pr[\text{B}] + \Pr[\neg\text{C}]. \end{aligned}$$

With this lemma we can define sequence games based on *condition events*. In game  $\mathbf{G}_{i+1}$  constructed from  $\mathbf{G}_i$  w.r.t. some appropriate condition event  $\mathbf{C}$  the event  $\text{Win}_{i+1}$  is defined as  $\text{Win}_i|\mathbf{C}$ . Then according to Lemma 1  $\Pr[\text{Win}_i] - \Pr[\text{Win}_{i+1}] \leq \Pr[\neg\mathbf{C}]$ . In order to estimate the probability distance between  $\mathbf{G}_i$  and  $\mathbf{G}_{i+1}$  it is sufficient to compute  $\Pr[\neg\mathbf{C}]$ . Note that  $\mathbf{G}_i$  and  $\mathbf{G}_{i+1}$  proceed identical from the adversarial perspective. Therefore, it is not necessary for the simulator to detect whether this condition event occurs or not (this in contrast to failure events, used for example in  $\mathbf{G}_1$  of Theorem 2). Furthermore, by conditioning the success of the adversary with  $\mathbf{C}$  we do not restrict the adversarial strategy. Note that the inequality

$$\begin{aligned} \Pr[\text{Win}_i] &= \Pr[\text{Win}_{i+1}] \Pr[\mathbf{C}] + \Pr[\text{Win}_i|\neg\mathbf{C}] \Pr[\neg\mathbf{C}] \\ &\leq \Pr[\text{Win}_{i+1}] + \Pr[\neg\mathbf{C}] \end{aligned}$$

considers  $\text{Win}_{i+1}$  and  $\text{Win}_i|\neg\mathbf{C}$ , and so focusing on one strategy represented by  $\text{Win}_{i+1} = \text{Win}_i|\mathbf{C}$  in  $\mathbf{G}_{i+1}$  does not rule out all other strategies represented by  $\text{Win}_i|\neg\mathbf{C}$  because of the total probability  $\Pr[\text{Win}_i] \leq \Pr[\text{Win}_{i+1}] + \Pr[\neg\mathbf{C}]$ .

The main idea of the following proof is to use the fact that every honest oracle  $\Pi_i^s \in \mathcal{G}$ ,  $i \in [1, n]$  computes the sequence  $\rho_1, \dots, \rho_n$  prior to the acceptance of  $\mathbf{K}$  so that each  $\rho_l$ ,  $l \in [1, n]$  depends on the previously computed  $\rho_{l-1}$ . We consider the probability that for an honest  $\Pi_{i^*}^s$  the adversary  $\mathcal{A}$  is able to enforce any of the values  $\rho_{i^*}, \dots, \rho_n$  (note that  $K = \rho_n$ ), or  $\mathbf{K}$  computed by the collision-resistant pseudo-random function  $f$ . This is equivalent to the event that in the **prepare** stage  $\mathcal{A}$  is able to output any  $\rho_{i^*}, \dots, \rho_n$ , or  $\mathbf{K}$  which  $\Pi_{i^*}^s$  computes in any session of the **attack** stage. On the other hand, applying Lemma 1 in our proof we do also consider the upper-bound for the success probability of the adversary in case that its strategy differs from influencing any value in  $\rho_{i^*}, \dots, \rho_n$ .

*Proof (of Theorem 4).* (Sketch) Assume that an adversary  $\mathcal{A}$  from Definition 5 wins in  $\text{Game}_{\mathcal{A}, \text{TdH1}}^{\text{con}}(\kappa)$  (which event we denote  $\text{Win}^{\text{con}}$ ). Then at the end of the stage **prepare** it returned  $\tilde{\mathbf{K}}$  such that in the stage **attack** there exist  $i^* \in [1, n]$  and an honest oracle  $\Pi_{i^*}^s \in \mathcal{G}$  that accepts with  $\mathbf{K}_{i^*}^s = \tilde{\mathbf{K}}$ . Remind that  $\tilde{\mathbf{K}} = f_{K_{i^*}^s}(v_2)$  where  $K_{i^*}^s$  is the intermediate value computed by  $\Pi_{i^*}^s$ .

**Game  $\mathbf{G}_0$ .** This is the real game  $\text{Game}_{\mathcal{A}, \text{TdH1}}^{\text{con}}(\kappa)$ , in which the honest players are simulated by  $\Delta$ .

**Game  $\mathbf{G}_1$ .** In this game we  $\Delta$  aborts if the same nonce  $r_i$  is used by any honest oracle  $\Pi_i^s$  in two different sessions. As in previous proofs we get:

$$\Pr[\text{Win}_0^{\text{con}}] - \Pr[\text{Win}_1^{\text{con}}] \leq \frac{Nq_s^2}{2^\kappa}.$$

**Game  $\mathbf{G}_2$ .** This game is identical to Game  $\mathbf{G}_1$  with the condition event that  $\mathcal{A}$  being in the **prepare** stage is NOT able to output  $\rho_{i^*}$  computed by  $\Pi_{i^*}^s$  in any session of the **attack** stage.<sup>6</sup> We show how to evaluate the probability that  $\mathcal{A}$  outputs  $\rho_{i^*}$  in the **prepare** stage. Recall,  $\rho_{i^*}$  is computed as  $f_{\rho_{i^*-1} \oplus \pi(r_{i^*})}(v_0)$  in the **attack** stage. If  $\mathcal{A}$  does not know the PRF key in the **prepare** stage, he can either use a different PRF key (thus finding a PRF-collision) or guess  $\rho_{i^*}$  at random. If  $\mathcal{A}$  knows the PRF key in the first stage, he has to force  $\Pi_{i^*}^s$  to compute that key in the **attack** stage. However, since  $r_i$ 's are uniform and chosen in the second stage,  $\mathcal{A}$  must influence  $\rho_{i^*-1}$ ; this would allow to distinguish  $f$  from a random function. Since there are at most  $q_s$  sessions we have (according to Lemma 1):

$$\Pr[\text{Win}_1^{\text{con}}] - \Pr[\text{Win}_2^{\text{con}}] \leq q_s \text{Succ}_F^{\text{coll}}(\kappa) + q_s \text{Adv}_F^{\text{prf}}(\kappa) + \frac{q_s}{2^\kappa}.$$

**Game  $\mathbf{G}_3$ .** In this game we consider a condition event that  $\mathcal{A}$  (being in the **prepare** stage) is NOT able to output  $K_{i^*}^s := \rho_n$  computed by  $\Pi_{i^*}^s$  in any session of the **attack** stage. Evaluating probabilities

<sup>6</sup> Note, in  $\mathbf{G}_0$  and  $\mathbf{G}_1$  the adversary only outputs a value for the resulting group key. In  $\mathbf{G}_2$  we consider the additional (in)ability of the adversary to output the value for  $\rho_{i^*}$ . Since we are only interested in the success probability of  $\mathcal{A}$  under this condition  $\Delta$  does not need to detect whether  $\mathcal{A}$  is able to output the correct value or not. The same considerations are applicable to  $\mathbf{G}_3$  w.r.t.  $K_{i^*}^s$ .

that  $\rho_n, \rho_{n-1}, \dots$  can be predicted is done via a hybrid argument. In a nutshell, either the adversary can find the same output with a different key (which breaks collision-resistance) or he influences the PRF key  $\rho_{i-1} \oplus \pi(r_i)$ : this can be done either by inverting  $\pi$  or by a random guess. According to Lemma 1 we finally obtain:

$$\Pr[\text{Win}_2^{\text{con}}] - \Pr[\text{Win}_3^{\text{con}}] \leq Nq_s \text{Succ}_F^{\text{coll}}(\kappa) + Nq_s \text{Succ}_\pi^{\text{ow}}(\kappa) + \frac{Nq_s}{2^\kappa}.$$

**Game G<sub>4</sub>.** The condition event here is that  $\mathcal{A}$  (being in the prepare stage) is NOT able to output  $\mathbf{K}_{i^*}^s$  computed by  $\Pi_{i^*}^s$  in any session of the attack stage. Having excluded the case where  $K_{i^*}^s$  is known to  $\mathcal{A}$ , the probability of such event is (as above) bounded by:

$$\Pr[\text{Win}_3^{\text{con}}] - \Pr[\text{Win}_4^{\text{con}}] \leq q_s \text{Succ}_F^{\text{coll}}(\kappa) + \frac{q_s}{2^\kappa}.$$

Having  $\Pr[\text{Win}_4^{\text{con}}] = 0$  (by definition of the game) one can conclude.  $\square$

**Comparison of Security and Performance of TDH1 and Further GKE Protocols** In Table 1 we compare TDH1 protocol with several well-known provably secure GKE protocols in terms of their performance and security goals. Our comparison is done based on the security arguments and adversarial

**Table 1.** Efficiency and Security Goals of TDH1 and other *Static* Provably Secure GKE Protocols

GKE Protocol	Efficiency		Security Goals			Model
	Comm.	Comp.	AKE-Security	MA-Security	Contributiveness	
Abdalla <i>et al.</i> [1]	O(1)	O(n)	weak	-	-	ICM, ROM
Bresson and Catalano [7]	O(1)	O(n)	weak	weak, honest	weak, honest	standard
Bresson <i>et al.</i> [11]	O(n)	O(n)	weak	weak, honest	-	ROM
Bresson <i>et al.</i> [10]	O(n)	O(n)	weak	weak, honest	-	ROM
Desmedt <i>et al.</i> [18]	O(1)	O(n)	weak	weak, malicious	weak, malicious	standard
Dutta <i>et al.</i> [19]	O(1)	O(n)	weak	-	-	standard
Katz and Shin [20]	O(1)	O(n)	strong	strong <sup>7</sup> , malicious	-	standard
Katz and Yung [21]	O(1)	O(n)	weak	weak, honest	-	standard
TDH1	O(1)	O(n)	strong	strong, malicious	strong, malicious	standard

settings given in the original publications (sometimes transformed to the terminology of our model). In general, “weak” (or “strong”) denotes consideration of weak (or strong) corruptions for each of the security requirements, whereas “honest” (or “malicious”) denotes the assumption on the type of the protocol participants. Note again that by strong corruptions we mean not only adaptive attacks revealing the long-lived key (thus, weak corruptions) but also opening attacks which read out the ephemeral secrets. We also distinguish whether a protocol has been proven under standard or non-standard assumptions such as Ideal Cipher Model (ICM) or Random Oracle Model (ROM). We remark that TDH1 is the only protocol which provably satisfies strong versions of AKE-, MA-security and contributiveness (under consideration of malicious insiders where appropriate, that is for MA and contributiveness) while being proven in the standard model. The protocol proposed by Desmedt *et al.* [18] has similar properties as TDH1 but deals only with weak corruptions (ephemeral secrets never leak). The work by Katz and Shin<sup>8</sup> [20] can also be seen as close to ours since they provide MA-security against malicious insiders; the

<sup>7</sup> MA-security related definitions in [20] do not consider opening attacks, i.e., the adversary is not allowed to obtain internal states of other uncorrupted participants.

<sup>8</sup> Note that Katz and Shin proposed an add-on compiler and not a concrete protocol.

difference is that [20] (although considering strong corruptions) does not allow separate opening attacks: the scenario in which adversary learns the ephemeral secrets of other honest users is not considered. Additionally, we note that the overall efficiency of TDH1 is similar to the most efficient currently known provably secure GKE protocols (in the standard model).

## 4 Conclusions and Future Work

In this paper we have addressed security of GKE protocols against strong (adaptive) corruptions which reveal ephemeral secrets of participants and proposed appropriate definitions of strong AKE-, MA-security, and contributiveness. Additionally, we presented a three-round GKE protocol TDH1 which satisfies strong security under standard cryptographic assumptions.

We observe that the function  $\text{TDH1\_Con}(x_{(0,0)}, r_1 | \dots | r_n)$  is of independent interest since it can be seen as an add-on compiler for our definition of contributiveness if  $x_{(0,0)}$  is the common ephemeral secret computed in the underlying GKE protocol. Since many GKE protocols require users to generate random nonces (e.g. for building session ids as in [21]), achieving contributiveness in such protocols does not require any additional communication.

The equivalence between the TDDH and DDH assumptions is also of independent interest since it is valuable for the construction of other cryptographic schemes with provable security in the standard model. An interesting question is still whether TDDH is randomly self-reducible?

Beside the extension of TDH1 towards dynamic groups, general future work in the area of GKE security might address: consideration of strong corruptions in combination with fault-tolerance and security against DoS attacks discussed in [14] and [18]; and strengthening of the simulation-based security models for GKE protocols (e.g. [20]) towards opening attacks due to our Remark 1.

## References

1. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-Based Group Key Exchange in a Constant Number of Rounds. In *Proc. of PKC'06*, vol. 3958 of *LNCS*, p. 427–442. Springer, April 2006.
2. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proc. of ACM CCS'98*, p. 17–26. ACM Press, 1998.
3. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of ACM CCS'93*, p. 62–73. ACM Press, 1993.
4. J.-M. Bohli, M. I. G. Vasco, and R. Steinwandt. Secure Group Key Establishment Revisited. *Intl. Journal of Information Security*, 6(4):243–254, 2007.
5. D. Boneh. The Decision Diffie-Hellman Problem. In *Proc. of ANTS-III*, p. 48–63. Springer, 1998.
6. C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.
7. E. Bresson and D. Catalano. Constant Round Authenticated Group Key Agreement via Distributed Computation. In *Proc. of PKC'04*, vol. 2947 of *LNCS*, p. 115–129. Springer, 2004.
8. E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange — The Dynamic Case. In *ASIACRYPT'01*, vol. 2248 of *LNCS*, p. 290–390. Springer, 2001.
9. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. In *EUROCRYPT'02*, vol. 2332 of *LNCS*, p. 321–336. Springer, 2002.
10. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks. In *ASIACRYPT'02*, vol. 2501 of *LNCS*, p. 497–514. Springer, December 2002.
11. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. In *Proc. of ACM CCS'01*, p. 255–264. ACM Press, 2001.
12. E. Bresson and M. Manulis. Malicious Participants in Group Key Exchange: Key Control and Contributiveness in the Shadow of Trust. In *Proc. of ATC '07*, vol. 4610 of *LNCS*, p. 395–409. Springer, 2007.
13. E. Bresson, M. Manulis, and J. Schwenk. On Security Models and Compilers for Group Key Exchange Protocols. In *Proc. of IWSEC '07*, vol. 4752 of *LNCS*, p. 292–307. Springer, 2007.
14. C. Cachin and R. Strobl. Asynchronous Group Key Exchange with Failures. In *Proc. of PODC'04*, p. 357–366. ACM Press, 2004.

15. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *EUROCRYPT'01*, vol. 2045 of *LNCS*, p. 453–474. Springer, 2001.
16. K.-K. R. Choo, C. Boyd, and Y. Hitchcock. Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. In *ASIACRYPT'05*, vol. 3788 of *LNCS*, p. 585–604. Springer, 2005.
17. G. D. Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson. How to Forget a Secret. In *Proc. of STACS'99*, vol. 1563 of *LNCS*, p. 500–509. Springer, 1999.
18. Y. G. Desmedt, J. Pieprzyk, R. Steinfeld, and H. Wang. A Non-Malleable Group Key Exchange Protocol Robust Against Active Insiders. In *Proc. of ISC'06*, vol. 4176 of *LNCS*, p. 459–475. Springer, 2006.
19. R. Dutta, R. Barua, and P. Sarkar. Provably Secure Authenticated Tree-Based Group Key Agreement. In *Proc. of ICICS'04*, vol. 3269 of *LNCS*, p. 92–104. Springer, 2004.
20. J. Katz and J. S. Shin. Modeling Insider Attacks on Group Key Exchange Protocols. In *Proc. of ACM CCS'05*, p. 180–189. ACM Press, 2005.
21. J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. In *CRYPTO'03*, vol. 2729 of *LNCS*, p. 110–125. Springer, 2003.
22. H.-J. Kim, S.-M. Lee, and D. H. Lee. Constant-Round Authenticated Group Key Exchange for Dynamic Groups. In *ASIACRYPT'04*, vol. 3329 of *LNCS*, p. 245–259, 2004.
23. Y. Kim, A. Perrig, and G. Tsudik. Group Key Agreement Efficient in Communication. *IEEE Transactions on Computers*, 53(7):905–921, 2004.
24. Y. Kim, A. Perrig, and G. Tsudik. Tree-Based Group Key Agreement. *ACM Transactions on Information and System Security*, 7(1):60–96, February 2004.
25. H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. In *CRYPTO'05*, vol. 3621 of *LNCS*, p. 546–566. Springer, 2005.
26. B. LaMacchia, K. Lauter, and A. Mityagin. Stronger Security of Authenticated Key Exchange. In *Proc. of ProvSec'07*, vol. 4784 of *LNCS*, p. 1–16. Springer, 2007.
27. M. Manulis. Survey on Security Requirements and Models for Group Key Exchange. Technical Report 2006/02, Horst-Görtz Institute, November 2006.
28. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.
29. C. J. Mitchell, M. Ward, and P. Wilson. Key Control in Key Agreement Protocols. *Electronic Letters*, 34(10):980–981, 1998.
30. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO '91*, vol. 576 of *LNCS*, p. 129–140. Springer, 1991.
31. V. Shoup. On Formal Models for Secure Key Exchange (Version 4). Technical Report RZ 3120, IBM Research, November 1999.
32. V. Shoup. Sequences of Games: A Tool for Taming Complexity in Security Proofs. Cryptology ePrint Archive, Report 2004/332, January 2006.
33. M. Steiner. *Secure Group Key Agreement*. PhD thesis, Saarland University, March 2002.

## A Proof of Theorem 1 (DDH $\iff$ TDDH)

$\text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa) \leq \text{Adv}_{T_n, \mathbb{G}}^{\text{TDDH}}(\kappa)$ : This holds trivially since distributions  $\text{TDDH}_{T_n}^*$  and  $\text{TDDH}_{T_n}^{\S}$  contain a triple  $((\langle 1, 0 \rangle, g^{x(1,0)}), (\langle 1, 1 \rangle, g^{x(1,1)}), (\langle 0, 0 \rangle, Z))$  where  $Z = g^{x(1,0)x(1,1)}$  in case of  $\text{TDDH}_{T_n}^*$  or  $Z$  is random otherwise.

$\text{Adv}_{T_n, \mathbb{G}}^{\text{TDDH}}(\kappa) \leq (2n - 3)\text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa)$ : To prove this we use a  $\text{TDDH}_{T_n}$ -distinguisher  $\mathcal{A}$ , and show how to solve an instance of the DDH problem: on input  $(A, B, C) \in \mathbb{G}^3$ , where  $A = g^a$  and  $B = g^b$  for random  $a$  and  $b$ , we build a PPT algorithm  $\Delta$  that distinguishes whether  $C = g^{ab}$  or  $C$  is random.

First, we sort the nodes of any  $T_n \in \mathcal{T}_n$  in the postfix order, s.t. each node is listed *after* its two children (if any). For simplicity, we slightly modify this order: we separate nodes by re-numbering all  $n$  leaves to negative indices (without changing their order), and the internal nodes to indices 1 to  $n - 1$  (also without changing their order). In other words, for internal nodes, we “shrink” the sequence that remains after having moved the leaves. This results in the following map  $\sigma$  from  $T_n$  to  $[-n, -1] \cup [1, n - 1]$ :

$$\underbrace{-n, -n + 1, \dots, -1}_{\text{number assigned to leaves}}, \underbrace{1, \dots, n - 1}_{\text{internal nodes}}$$

Note that any node still appears after its children, e.g. the root node is assigned number  $\sigma(\langle 0, 0 \rangle) = n - 1$ .

By a ‘‘hybrid argument’’ we consider the following sequence of games. In each game  $\mathbf{G}_i$  for  $i = 0, \dots, n - 2$ ,  $\Delta$  chooses a set  $X$  of  $n$  random values in  $\mathbb{G}$ , denoted  $x_{-n}$  through  $x_{-1}$ . Additionally, for  $i > 0$ , in  $\mathbf{G}_i$ ,  $\Delta$  chooses a set  $Y_i$  of  $i$  random values in  $\mathbb{G}$ , that are denoted  $x_1$  through  $x_i$ . Then,  $\Delta$  builds a set  $\text{TDH}_i(X, Y_i)$  defined (recursively) as

$$\left\{ \left( \langle l, v \rangle, g^{x_{\langle l, v \rangle}} \right) \right\}_{\langle l, v \rangle \in T_n^*}, \quad \text{with} \quad \begin{cases} x_{\langle l, v \rangle} = x_{\sigma(\langle l, v \rangle)} \in X & \text{if } \sigma(\langle l, v \rangle) < 0, \\ x_{\langle l, v \rangle} = x_{\sigma(\langle l, v \rangle)} \in Y_i & \text{if } 0 < \sigma(\langle l, v \rangle) \leq i, \\ x_{\langle l, v \rangle} = g^{x_{\langle l+1, 2v \rangle} x_{\langle l+1, 2v+1 \rangle}} & \text{otherwise.} \end{cases}$$

Finally, in each game,  $\Delta$  flips a coin  $b$  and provides the  $\mathcal{A}$  with a set  $\text{TDDH}_i(X, Y_i, b, r) = \text{TDH}_i(X, Y_i) \cup \{(\langle 0, 0 \rangle, Z)\}$  where  $Z = g^{x_{\langle 1, 0 \rangle} x_{\langle 1, 1 \rangle}}$  if  $b = 1$  and  $Z = g^r$  is a random element in  $\mathbb{G}$  if  $b = 0$ .

Let  $\Pr_i[\dots]$  denote the probabilities as induced by random choices in  $\mathbf{G}_i$ . In  $\mathbf{G}_0$  the constructed  $\text{TDH}_0(X, Y_i)$  is exactly  $\text{TDH}_{T_n}(X)$  (due to  $Y_0 = \emptyset$ ) s.t.

$$\Pr_0[\mathcal{A}(\text{TDDH}_0(X, \emptyset, b, r)) = 1 | b = 1] - \Pr_0[\mathcal{A}(\text{TDDH}_0(X, \emptyset, b, r)) = 1 | b = 0] \leq \text{Adv}_{T_n, \mathbb{G}}^{\text{TDDH}}(\kappa).$$

On the other hand, in  $\mathbf{G}_{n-2}$ , all values  $x_{\langle l, v \rangle}$  for  $l > 1$  are random and independent, and also independent from random  $x_{\langle 1, 0 \rangle}$  and  $x_{\langle 1, 1 \rangle}$ . Furthermore,  $x_{\langle 0, 0 \rangle} = g^{x_{\langle 1, 0 \rangle} x_{\langle 1, 1 \rangle}}$  iff  $b = 1$  s.t.

$$\Pr_{n-2}[\mathcal{A}(\text{TDDH}_{n-2}(X, Y_{n-2}, b, r)) = 1 | b = 1] - \Pr_{n-2}[\mathcal{A}(\text{TDDH}_{n-2}(X, Y_{n-2}, b, r)) = 1 | b = 0] \leq \text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa).$$

The last experiment  $\mathbf{G}_*$  is identical to  $\mathbf{G}_{n-2}$  except that  $A, B$  and  $C$  are used in the computation of  $\text{TDDH}_{n-2}(X, Y_{n-2}, b, r)$  instead of  $g^{x_{\langle 1, 0 \rangle}}, g^{x_{\langle 1, 1 \rangle}}$  and  $x_{\langle 0, 0 \rangle}$ , respectively. In particular the flipping of  $b$  is ignored here: whatever  $b$ , the last input of  $\mathcal{A}$  is set to  $C$ . Let  $\beta$  denote the hidden bit that  $\Delta$  is trying to guess. Note, the random variables  $(b, g^{x_{\langle 1, 0 \rangle}}, g^{x_{\langle 1, 1 \rangle}}, x_{\langle 0, 0 \rangle})$  and  $(\beta, A, B, C)$  are identically distributed. It follows that (for simplicity we removed  $\mathcal{A}$ 's inputs that are identical and independent from the rest in both cases):

$$\begin{aligned} \Pr_{n-2}[\mathcal{A}(g^{x_{\langle 1, 0 \rangle}}, g^{x_{\langle 1, 1 \rangle}}, x_{\langle 0, 0 \rangle}) = 1 | b = 1] &= \Pr_*[\mathcal{A}(A, B, C) = 1 | \beta = 1] \\ \Pr_{n-2}[\mathcal{A}(g^{x_{\langle 1, 0 \rangle}}, g^{x_{\langle 1, 1 \rangle}}, x_{\langle 0, 0 \rangle}) = 1 | b = 0] &= \Pr_*[\mathcal{A}(A, B, C) = 1 | \beta = 0] \end{aligned}$$

Last, it is straightforward to see that the computational distance between two consecutive games is upper-bounded by  $\text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa)$  since the only difference between them consists in replacing a value  $g^{x_{\langle l+1, 2v \rangle} x_{\langle l+1, 2v+1 \rangle}}$  by a random one. Hence,

$$\begin{aligned} \left| \Pr_*[\mathcal{A}(A, B, C) = 1 | \beta = 1] - \Pr_0[\mathcal{A}(\text{TDDH}_0(X, \emptyset, b, r)) = 1 | b = 1] \right| &\leq (n - 2) \text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa) \\ \left| \Pr_*[\mathcal{A}(A, B, C) = 1 | \beta = 0] - \Pr_0[\mathcal{A}(\text{TDDH}_0(X, \emptyset, b, r)) = 1 | b = 0] \right| &\leq (n - 2) \text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa) \end{aligned}$$

Their sum gives us the desired inequality:

$$\begin{aligned} \text{Adv}_{T_n, \mathbb{G}}^{\text{TDDH}}(\kappa) &= |\Pr_0[\mathcal{A}(\text{TDDH}_0(X, \emptyset, b, r)) = 1 | b = 1] - \Pr_0[\mathcal{A}(\text{TDDH}_0(X, \emptyset, b, r)) = 1 | b = 0]| \\ &\leq |\Pr_*[\mathcal{A}(A, B, C) = 1 | \beta = 1] - \Pr_*[\mathcal{A}(A, B, C) = 1 | \beta = 0]| + 2(n - 2) \text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa) \\ &\leq (2n - 3) \text{Adv}_{\mathbb{G}}^{\text{DDH}}(\kappa). \quad \square \end{aligned}$$

## B Collision Resistant Pseudo-Random Functions

**Definition 7 (Pseudo-Randomness of  $F$ ).** *An ensemble of finite functions  $F := \{ \{ f_k : \{0, 1\}^{p(\kappa)} \rightarrow \{0, 1\}^{p(\kappa)} \}_{k \in \{0, 1\}^\kappa} \}_{\kappa \in \mathbb{N}}$  with  $p$  a polynomial, is called an (efficiently computable) pseudo-random function ensemble if:*

1. Efficient computation: *There exists a polynomial-time algorithm that on input  $k$  and  $x$  returns  $f_k(x)$ .*
2. Pseudo-Randomness: *Choose uniformly  $k \in_R \{0, 1\}^\kappa$  and a function  $\tilde{f}$  in the set of all functions with domain and range  $\{0, 1\}^{p(\kappa)}$ . Consider a PPT adversary  $\mathcal{A}$  asking queries of the form  $\text{Tag}(x)$  and participating in one of the following two games:*
  - $\text{Game}_{\mathcal{A}, F}^{\text{prf}-1}(\kappa)$  where a query  $\text{Tag}(x)$  is answered with  $f_k(x)$ ,
  - $\text{Game}_{\mathcal{A}, F}^{\text{prf}-0}(\kappa)$  where a query  $\text{Tag}(x)$  is answered with  $\tilde{f}(x)$ .*At the end of the execution  $\mathcal{A}$  outputs a bit  $b$  trying to guess which game was played. The output of  $\mathcal{A}$  is also the output of the game.*

We define:  $\text{Adv}_{\mathcal{A}, F}^{\text{prf}}(\kappa) := |2 \Pr[\text{Game}_{\mathcal{A}, F}^{\text{prf}-b}(\kappa) = b] - 1|$

and denote  $\text{Adv}_F^{\text{prf}}(\kappa)$  the maximum advantage over all adversaries  $\mathcal{A}$ . We say that  $F$  is pseudo-random if this advantage is a negligible function in  $\kappa$ .

By an (efficiently computable) pseudo-random function we mean a function  $f_k \in F$  for some random  $k \in_R \{0, 1\}^\kappa$ .

**Definition 8 (Collision-Resistance of  $F$ ).** Let  $F := \{\{f_k\}_{k \in \{0, 1\}^\kappa}\}_{\kappa \in \mathbb{N}}$  be a pseudo-random function ensemble. We say that  $F$  is collision-resistant if there is an efficient procedure **Sample** such that for all PPT adversaries  $\mathcal{A}$  the following success probability (over all adversaries  $\mathcal{A}$ ) is a negligible function in  $\kappa$ :

$$\text{Succ}_F^{\text{coll}}(\kappa) := \Pr \left[ \begin{array}{l} x \leftarrow \text{Sample}(1^\kappa); \\ k, k' \leftarrow \mathcal{A}(1^\kappa, x) : \end{array} \begin{array}{l} k, k' \in \{0, 1\}^\kappa \wedge \\ k \neq k' \wedge \\ f_k(x) = f_{k'}(x) \end{array} \right]$$