An extended abstract of this article appears in the proceedings of CT-RSA 2021. For the full version, please consult the IACR eprint archive.

SoK: Game-based Security Models for Group Key Exchange

Bertram Poettering[1], Paul Rösler[2], Jörg Schwenk[3], and Douglas Stebila[4]

[1] IBM Research – Zurich, Rüschlikon, Switzerland
poe@zurich.ibm.com
[2] TU Darmstadt, Darmstadt, Germany
paul.roesler@tu-darmstadt.de
[3] Ruhr University Bochum, Bochum, Germany
joerg.schwenk@rub.de
[4] University of Waterloo, Waterloo, Canada
dstebila@uwaterloo.ca

Abstract. *Group key exchange* (GKE) protocols let a group of users jointly establish fresh and secure key material. Many flavors of GKE have been proposed, differentiated by, among others, whether group membership is static or dynamic, whether a single key or a continuous stream of keys is established, and whether security is provided in the presence of state corruptions (post-compromise security). In all cases, an indispensable ingredient to the rigorous analysis of a candidate solution is a corresponding formal security model. We observe, however, that most GKE-related publications are more focused on building new constructions that have more functionality or are more efficient than prior proposals, while leaving the job of identifying and working out the details of adequate security models a subordinate task.

In this systematization of knowledge we bring the formal modeling of GKE security to the fore by revisiting the intuitive goals of GKE, critically evaluating how these goals are reflected (or not) in the established models, and how they would be best considered in new models. We classify and compare characteristics of a large selection of *game-based* GKE models that appear in the academic literature, including those proposed for GKE with post-compromise security. We observe a range of shortcomings in some of the studied models, such as dependencies on overly restrictive syntactical constrains, unrealistic adversarial capabilities, or simply incomplete definitions. Our systematization enables us to identify a coherent suite of desirable characteristics that we believe should be represented in all general purpose GKE models. To demonstrate the feasibility of covering all these desirable characteristics simultaneously in one concise definition, we conclude with proposing a new generic reference model for GKE.

Keywords: Group key exchange · key agreement · key establishment · security model · multi-user protocol

1 Introduction

The group key exchange (GKE) primitive was first considered about four decades ago. The aim of early publications on the topic [ITW82, BD95] was to generalize the (two-party) Diffie–Hellman protocol to groups of three or more participants, i.e., to construct a basic cryptographic primitive that allows a fixed set of anonymous participants to establish secure key material in the presence of a passive adversary. Later research identified a set of additional features that would be desirable for GKE, for instance the support of participant authentication, the support of dynamic groups (where the set of participants is not fixed but members can join and leave the group at will), the support of groups where one or many members might temporarily be unresponsive (asynchronous mode of operation), or a maximum resilience against adversaries that can obtain read-access to the states of participants for a limited amount of time (post-compromise security).

Standard applications that require a GKE protocol as a building block include online audio-video conference systems and instant messaging [RMS18]. Indeed, in an ongoing standardization effort the IETF's Messaging Layer Security (MLS) initiative [BBM+20] tests employing GKE protocols for the protection of instant messaging in asynchronous settings.

While, intuitively, most of the GKE protocols proposed in the literature can serve as a building block for such applications, it turns out that effectively no two security analyses of such protocols were conducted in the same formal model, meaning that there is effectively no modularity: For every GKE candidate that is used in some application protocol, a new security evaluation of the overall construction has to be conducted. In fact, as will become clear in the course of this article, the GKE literature has neither managed to agree on a common unified syntax of the primitive, nor on a common approach for developing and expressing corresponding security definitions. In our view, the lack of a common reference framework for GKE, including its security, and the implied lack of modularity and interoperability, imposes an unnecessary obstacle on the way to secure conference and messaging solutions.

1.1 Systemizing Group Key Exchange Models

With the goal of developing a general reference formalization of the GKE primitive, we have a fresh look at how it should be modeled such that it simultaneously provides sufficient functionality and sufficient security. More precisely, we are looking for a formalization that is versatile enough to practically fit and protect generic applications like the envisioned video conferencing. To achieve this, we need to consider questions like the following: What features does an application expect of GKE? What type of underlying infrastructure (network services, authentication services, . . .) can a GKE primitive assume to exist? What types of adversaries should be considered? To obtain a satisfactory reference formalization, our model should be as generic as possible when meeting the requirements of applications, should make minimal assumptions on its environment, and should tolerate a wide class of adversaries.

After identifying the right questions to ask, we derive a taxonomy in which existing models for GKE can be evaluated to determine whether they provide answers to these questions. If they don't, we explore the consequences of this. As a side product, our taxonomy also sheds light on how the research domain of GKE has evolved over time, and how models in the literature relate to each other. It also informs us towards our goal to develop a versatile and uniform model for GKE.

We organize our taxonomy and investigations with respect to four property categories of GKE models:

1. the syntax of GKE (Section 2),
2. the definition of partnering (Section 3.1),
3. the definition of correctness (due to space restriction in Appendix A), and
4. the definition of security (Section 4).

While syntax, correctness, and security are properties generally formalized for all kinds of cryptographic primitives, the partnering notion is specific to the domain of key exchange. In a nutshell, partnering captures the conditions under which remote parties compute the same session key.

For each of these four categories, we discuss their purposes and central features, and classify the literature with respect to them. Having both considered the literature and revisited GKE with a fresh view, we identify a set of desirable characteristics in each of the four categories, from the perspective of generality of use and minimality of assumptions on the context in which GKE takes place. Based on these findings, we see how individual definitional approaches and, to some extent, subparadigms of GKE, do not fully satisfy the needs of GKE analysis. We are further able to synthesize a coherent set of desirable properties into a single, generic model (Sections 2.5, 3.2, and 4.1, and Appendix A.1), demonstrating that it is possible to design a model that simultaneously incorporates all these characteristics.

Choice of Literature. Most of the literature in the domain of GKE revolves around the exposition of a new construction (accompanied either with formal or only heuristic security arguments; see, e.g., [ITW82, BD95]). When selecting prior publications to survey in this SoK article, we focused on those that were developed with respect to a *formal computational game-based security model*. Our comparison covers all publications on GKE with this type of model that appeared in cryptographic "tier-one" proceedings[5] [BCP01, BCPQ01, BCP02a, BCP02b, KY03, KLL04, KS05, CCG+18, ACDT19]. Beyond

[5] CRYPTO, Eurocrypt, Asiacrypt, CCS, S&P, Usenix Security, and the Journal of Cryptology.

that, we browsed through the proceedings of all relevant "tier-two" conferences[6] and selected publications that explicitly promise to enhance the formal modeling of GKE [GBG09, YKLH18].[7] We also include three recently published articles on GKE with post-compromise security (aka. group ratcheting or continuous GKE), one of which is yet only available as a preprint [CCG$^+$18, ACDT19, ACC$^+$19].[8] As computational simulation-based (UC) and symbolic modeling approaches are essentially incomparable with computational game-based notions, we exclude these type of models from our systematization.

Tables 1, 2, 3, and 5 summarize and compare the common features that we identified in the surveyed models. The models reflected in these tables are arranged into three clusters. Leftmost: GKE in static groups [BCPQ01, BCP02b, KY03, KS05, GBG09, CCG$^+$18]; centered: GKE with regular, post-compromise secure key material updates (aka. ratcheting) [CCG$^+$18, ACDT19, ACC$^+$19]; and rightmost: GKE in dynamic groups [ACDT19, ACC$^+$19, BCP01, BCP02a, KLL04, YKLH18]. Within each cluster, models are, where possible, ordered chronologically by publication date. Naturally, works are historically related, did influence each other, and use intuitively similar notations across these clusters (e.g., due to overlapping sets of authors). Our results, however, show that these "soft" properties are almost independent of the factual features according to which we systematized the models. We correspondingly refrain from introducing further "clustering-axes" with respect to historic relations between the considered works as this may mislead more than it supports comprehensibility.

We use symbols ●, ⦿, ◕, ◔, ○, -, and others to condense the details of the considered model definitions in our systematizing tables, and accompany them with textual explanations. Not surprisingly, this small set of symbols can hardly reflect all details encoded in the models but makes "losses due to abstraction" unavoidable. We optimized the selection of classification criteria such that the amount of information loss due to simplifications is minimized.

Relation to Two-Party Key Exchange. While the focus of this article is on GKE, many of the notions that we discuss are relevant also in the domain of two-party key exchange. In our comparisons, we indicate which properties are specific to the setting of GKE, and which apply to key exchange in general. Given the large amount of two-party key exchange literature, we do not attempt to provide more direct comparisons between group and two-party key exchange.

Proposed Model. Since none of the models that we survey achieves all the desirable properties that we identify, we conclude this article with proposing a simple and generic GKE model that achieves all these properties. The components of this model are introduced gradually at the end of each systematization section. We emphasize that it is not necessarily our goal to guide all future research efforts to a unified GKE model. Some modeling design decisions are not universal and cannot be reduced to objective criteria, so we are neither under the illusion that a perfectly unified model exists, nor that the research community will any time soon agree on a single formalization. Our primary goal when writing down a model was rather to demonstrate the relative compatibility of the desirable properties. That said, as our systematization reveals undesirable shortcomings even in very recent GKE models for ratcheting—shortcomings that partially seem inherited from older work—we believe that proposing a better alternative is long overdue.

Although our model can be used for analyzing GKE protocols with various realistic, so far disregarded properties, achieving these properties is not mandatory but optional for covered GKE protocols. For example, dynamic GKE protocols with multi-device support that can handle fully asynchronous interaction can be analyzed by our model as well as static GKE protocols in which the interaction between the participating instances follows a fixed schedule. We consider these properties as implementation details of the protocols to which our model is carefully defined indifferent. The only mandatory property that our model demands is the secrecy of keys in the presence of either active or passive adversaries, which demonstrates the generality and versatility of our proposal.

[6] TCC, PKC, CT-RSA, ACNS, ESORICS, CANS, ARES, ProvSec, FC.

[7] We appreciate that many more publications introduce other GKE constructions [BDR20, BC04, ABCP06, JKT07, NS11, TC06, NPKW07, JL07, Man09, XHZ15, BBM09, YT10, NSS12, LY13, GZ12, FTY19, DLB07, WZ08]. However, we did not identify that they contribute new insights to the modeling of GKE.

[8] Since our analysis started before [ACDT20] was submitted to CRYPTO 2020, we consider a fixed preprint version [ACDT19] here. Note that the two follow-up works [ACJM20, AJM20] use simulation-based security models.

1.2 Basic Notions in Group Key Exchange

A group key exchange scheme is a tuple of algorithms executed by a group of participants with the minimal outcome that one or multiple (shared) symmetric keys are computed.

Terminology of GKE. A **global session** is a joint execution of a GKE protocol. By joint execution we mean the distributed invocation of GKE algorithms by participants that influence each other through communication over a network, eventually computing (joint) keys. Each *local* execution of algorithms by a participant is called a **local instance**. Each local instance computes one or more symmetric keys, referred to as **group keys**. Each group key computed by a single local instance during a global session has a distinct **context**, which may consist of: the set of designated participants, the history of previously computed group keys, the algorithm invocation by which its computation was initiated, etc. Participants of global sessions, represented by their local instances, are called **parties**.[9] If the set of participants in a global session can be modified during the lifetime of the session, this is an example of **dynamic** GKE; otherwise the GKE is **static**.

There are many alternative terms used in the GKE literature for these ideas: local instances are sometimes called *processes*, local *sessions*, or (misleadingly) *oracles*; group keys are sometimes called *session keys*; and parties are sometimes called *users*.

Security Models for GKE. As it is common in game-based key exchange models, an adversary against the security of a GKE scheme plays a game with a challenger that simulates multiple parallel real global sessions of the GKE scheme. The challenge that the adversary is required to solve is to distinguish whether a **challenge key** is a real group key established in one of the simulated global sessions or is a random key. In order to solve this challenge, the adversary is allowed to obtain (through a **key reveal** oracle) group keys that were computed independently, to obtain (through a **state exposure** oracle) ephemeral local secret states of instances that do not enable the trivial solution of the challenge, and to obtain (through a **corruption** oracle) static party secrets that do not trivially invalidate the challenge either. While the GKE literature agrees on these high-level concepts, the crucial details are implemented in various incompatible ways in these articles.

2 Syntax Definitions

Modeling a cryptographic primitive starts with fixing its syntax: the set of algorithms that are available, the inputs they take and the outputs they generate. We categorized the GKE models we consider according to the most important classes of syntactical design choices. In particular, the GKE syntax may reflect (1) imposed limits on the number of supported parties, sessions, and instances; (2) assumptions made on the available infrastructure (e.g., the existence of a PKI); (3) the type of operations that the protocols implement (adding users, removing users, refreshing keys, ...); and (4) the information that the protocols provide to the invoking application (set of group members, session identifiers, ...). We compile the results of our studies in Table 1. If for some models an unambiguous mapping to our categories is not immediate, we report the result that comes closest to what we believe the authors intended. Independently, if in any of the categories one option is clearly more attractive than the other options, we indicate this in the **Desirable** column. (We leave the cells of that column empty if no clearly best option exists.) The **Our model** column indicates the profile of our own GKE model; see also Section 2.5. Note that no two models in the table have identical profiles.[10]

The upcoming paragraphs introduce our categories in detail.

2.1 Quantities

All models we consider assume a universe of parties that are potential candidates for participating in GKE sessions. **Instances per party:** While most models assume that each party can participate—using independent instances—in an unlimited number of sessions, three models impose a limit to a

[9] We further clarify on the relation between local instances and parties and their participation in sessions in Appendix B.

[10] Surprisingly, this holds even for models that appeared in close succession in publications of the same authors.

Syntax	GKE-specific	[BCPQ01]	[BCP02b]	[KY03]	[KS05]	[GBG09]	[CCG+18]	[ACDT19]	[ACC+19]	[BCP01]	[BCP02a]	[KLL04]	[YKLH18]	Desirable	Our model
Quantities															
Instances per party	○	n	n	n	n	n	n	1	(1)	1	n	n	n	n	n
Parties per session	●	**F**	**F**	**V**	**V**	**V**	**V**	**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**
Multi-participation	●	○	○	○	○	○	○	○	○	○	○	○	○	●	●
Setup assumptions															
Authentication by ...	○	**SK**	**PW**	**PK**	**PK**	**PK**	**PK**	**PK**	**(PK)**	**SK**	**PK**	**PK**	**PK**		any
PKI	○	-	-	●*	●*	⦿	⦿	●*	⦿	-	⦿	●*	●		-
Online administrator	●	-	-	-	-	-	-	⦿	◔	●	●	⦿	○	○	○
Operations															
Level of specification	○	○	**G**	○	○	○	**L**	**L**	**L**	**G**	**G**	**G**	◔	**L**	**L**
Algo: Setup	○	○	●	○	○	○	●	●	●	●	●	●	○		-
Algo: Add	●	○	○	○	○	○	○	●	●	●	●	●	●		-
Algo: Remove	●	○	○	○	○	○	○	●	●	●	●	●	●		-
Algo: Refresh/Ratchet	○	○	○	○	○	○	⦿	●	●	○	○	○	○		-
Abstract interface	○	○	○	○	○	○	●	○	○	○	○	○	○	●	●
Return values															
Group key	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Ref. for session	●	○	○	⦿	●	⦿	○	○	○	○	○	⦿	⦿		○
Ref. for group key	○	○	○	⦿	⦿	○	○	○	○	○	○	●	○	●	●
Designated members	●	⦿	⦿	⦿	●	●	⦿	○	⦿	●	○	⦿	⦿	●	●
Ongoing operation	○	-	⦿	-	-	-	○	○	⦿	○	○	○	⦿		○
Status of instance	○	○	○	⦿	○	⦿	⦿	⦿	⦿	○	○	○	⦿	●	⦿

Table 1. Syntax definitions. Notation: n: many; **F**: fixed; **V**: variable; **D**: dynamic; ●: yes; ⦿: implicitly; ◔: partially; ○: no; -: not applicable; (blank): no option clearly superior/desirable; **SK**: symmetric key; **PW**: password; **PK**: public key; **G**: global; **L**: local.

single instance per party.[11] In Table 1 we distinguish these cases with the symbols n and 1, respectively. **Parties per session:** While some models prescribe a fixed number of parties that participate in each GKE session, other models are more flexible and assume either that the number of parties is in principle variable yet bound to a static value when a session is created, or even allow that the number of parties changes dynamically over the lifetime of a session (accommodating parties being added/removed). In the table we encode the three cases with the symbols **F**,**V**,**D**, respectively. **Multi-participation:** In principle it is plausible that parties participate multiple times in parallel in the same session (through different, independent instances, e.g., from their laptop and smartphone; see also Appendix B). We note however that all of the assessed models exclude this and don't allow for more than one participation per party. We encode this in the table by placing the symbol ○ in the whole row. Despite no model supporting it, we argue that a multi-participation feature might be useful in certain cases.

Discussion. We note that security reductions of early ring-based GKE protocols [BD95] require that the number of participants in sessions always be even [BD05]. We take this as an example that clarifies a crucial difference between the **F** and **V** types in the Parties-per-session category, as [BD95] fits into the **F** regime but not into the **V** regime.

[11] The case of [ACC+19] is somewhat special: While their syntax in principle allows that parties operate multiple instances, their security definition reduces this to strictly one instance per party. For their application (secure instant messaging) this is not a limitation as parties are short-lived and created ad-hoc to participate in only a single session.

2.2 Setup Assumptions

Security models are formulated with respect to a set of properties that are assumed to hold for the environment in which the modeled primitive is operated. We consider three classes of such assumptions. The classes are related to the pre-distribution of key material that is to be used for authentication, the availability of a centralized party that leads the group communication, and the type of service that is expected to be provided by the underlying communication infrastructure. **Authentication by . . . :** If a GKE protocol provides key agreement with authentication, its syntax has to reflect that the latter is achievable only if some kind of cryptographic setup is established before the protocol session is executed. For instance, depending on the type of authentication, artifacts related to accessing pre-shared symmetric keys, passwords, or authentic copies of the peers' public keys, will have to emerge in the syntax. In the table we encode these cases with the symbols **SK**,**PW**,**PK**, respectively.[12] **PKI:** In the case of public-key authentication we studied what the models say about how public keys are distributed, in particular whether a public key infrastructure (PKI) is explicitly or implicitly assumed. In the table we indicate this with the symbols ● and ⦿. We further specially mark with ●* the cases of "closed PKIs" that service exclusively potential protocol participants, i.e., PKIs with which non-participants (e.g., an adversary) cannot register their keys. **Online administrator:** The number of participants in a GKE session can be very large, and, by consequence, properly orchestrating the interactions between them can represent a considerable technical challenge.[13] Two of the models we consider resolve this by requiring that groups be managed by a distinguished always-honest leader (either being a group member or an external delivery service) who decides which operations happen in which order, and another two models assume the same but without making it explicit. The model of [ACC$^+$19] is slightly different in that a leader is still required, but it does not have to behave honestly. The model of [YKLH18] does not assume orchestration: Here, protocols proceed execution as long as possible, even if concurrent operations of participants are not compatible with each other. This is argued to be sufficient if security properties ensure that the resulting group keys are sufficiently independent. The remaining models are so simple that they do not require any type of administration.

Discussion. While the authentication component that is incorporated into GKE protocols necessarily requires the pre-distribution of some kind of key material, the impact of this component on the GKE model should be minimal; in particular, details of PKI-related operations should not play a role. It is even less desirable to assume closed PKIs to which outsiders cannot register their keys.

As we have seen, some models require an online administrator where others do not. If an online administrator is available, tasks like ensuring that all participants in a session have the same view on the communication and group membership list become easy. However, in many settings an online adminstrator is just not available. For instance, instant messaging protocols are expected to tolerate that participants, including any administrator, might go offline without notice. Unfortunately, if no adminstrator is available, seemingly simple tasks like agreeing on a common group membership list become hard to solve as, at least implicitly, they require solving a Byzantine Consensus instance. On the other hand, strictly speaking, achieving key security in GKE protocols is possible without reaching consensus.

2.3 Operations

In this category we compare the GKE models with respect to the algorithms that parties have available for controlling how they engage in sessions. **Level of specification:** While precisely fixing the APIs of these algorithms seems a necessity for both formalizing security and allowing applications to generically use the protocols, we found that very few models are clear about API details: Four models leave the syntax of the algorithms fully undefined.[14] Another four models describe operations only as global operations,

[12] In continuation of Footnote 11: The case of [ACC$^+$19] is special in that the requirement is an *ephemeral asymmetric key*, that is, a public key that is ad-hoc generated and used only once.

[13] Consider, for instance, that situations stemming from participants concurrently performing conflicting operations might have to be resolved, as have to be cases where participants become temporarily unavailable without notice.

[14] In some cases, however, it seems feasible to reverse-engineer some information about an assumed syntax from the security reductions also contained in the corresponding works.

i.e., specify how the overall state of sessions shall evolve without being precise about which steps the individual participants shall conduct. Only three models fix a local syntax, i.e., specify precisely which participant algorithms exist and which inputs and outputs they take and generate. In the table, we indicate the three levels of specification with the symbols ○, **G**, and **L**, encoding the terms “missing”, “global”, and “local”, respectively. The model of [YKLH18] sits somewhere between **G** and **L**, and is marked with ◕. **Algo:** The main operations executed by participants are session initialization (either of an empty group or of a predefined set of parties), the addition of participants to a group, the removal of participants from a group, and in some cases a key refresh (which establishes a new key without affecting the set of group members). In the table we indicate which model supports which of these operations. Note that the correlation between the Add/Remove rows and symbol **D** in Quantities/Parties-per-Session is as expected. Only very recent models that emerged in the context of group ratcheting support the key refresh operation. **Abstract interface:** While the above classes Add/Remove/Refresh are the most important operations of GKE, other options are possible, including Merge and Split operations that join two established groups or split them into partitions, respectively. In principle, each additional operation could explicitly appear in the form of an algorithm in the syntax definition of the GKE model, but a downside of this would be that the models of any two protocols with slightly different feature sets would become, for purely syntactic reasons, formally incomparable. An alternative is to use only a single algorithm for all group-related operations, which can be directed to perform any supported operation by instructing it with corresponding commands. While we believe that this flexible approach towards defining APIs to group operations has quite desirable advantages, we have to note that only one of the considered models supports it.

Discussion. Instance-centric (‘**L**-level’) specifications of algorithms are vital for achieving both practical implementability and meaningful security definitions. To see the latter, consider that the only way for adversaries to attack (global) sessions is by exposing (local) instances to their attacks.

2.4 Return Values

The main outcome of a successful GKE protocol execution is the group key itself. In addition, protocol executions might establish further information that can be relevant for the invoking application. We categorize the GKE models by the type of information contained in the protocol outcome. **Group key:** We confirm that all models that we consider have a syntactical mechanism for delivering established keys. **Reference for session:** By a session reference we understand a string that serves as an unambiguous handle to a session, i.e., a value that uniquely identifies a distributed execution of the scheme algorithms. Some of the models we consider require that such a string be established as part of the protocol execution, but not necessarily they prescribe that it be communicated to the invoking application along with the key. (Instead the value is used to define key security.) In Table 1, we indicate with symbols ● and ⦿ whether the models require the explicit or implicit derivation and communication of a session reference. We mark models with ○ if no such value appears in the model. **Reference for group key:** A key reference is similar to a session reference but instead of referring to a session it refers to an established key. While references to sessions and keys are interchangeable in some cases, in general they are not. The latter is the case, for instance, for protocols that establish multiple keys per execution. Further, if communication is not authentic, session references of protocol instances can be matching while key references (and keys) are not. In the table we indicate with symbols ● and ⦿ if the models consider explicit or implicit key references. **Designated members:** Once a GKE execution succeeds with establishing a shared key, the corresponding participants should learn who their partners are, i.e., with whom they share the key. In some models this communication step is made explicit, in others, in particular if the set of partners is *input* to the execution, this step is implicit. A third class of models does not communicate the set of group members at all. In the table we indicate the cases with symbols ●, ⦿, ○, respectively. **Ongoing operation:** In GKE sessions, keys are established as a result of various types of actions, particularly including the addition/removal of participants, and the explicit refresh of key material. We document for each considered model whether it communicates for established group keys through which operation they were established. **Status of instance:** Instances can attain different protocol-dependent internal states. Common such states are that instances can be in an accepted or rejected state, meaning that they consider a protocol execution successful or have given up on it, respectively. In this category we indicate whether the models we consider communicate this status information to the invoking application.

Discussion. In settings where parties concurrently execute multiple sessions of the same protocol, explicit references to sessions and/or keys are vital for maintaining clarity about which key belongs to which execution. (Consider attacks where an adversary substitutes all protocol messages of one session with the messages of another session, and vice versa, with the result that a party develops a wrong understanding of the context in which it established a key .) We feel that in many academic works the relevance of such references could be more clearly appreciated. The formal version of our observation is that session or key references are a prerequisite of sound composition results (as in [BFWW11]). Sound composition with other protocols plays a pivotal role also in the Universal Composability (UC) framework [Can01]. Indeed, not surprisingly, the concept of a session reference emerges most clearly in the UC-related model of [KS05].

Also related to composition is the requirement of explicitly (and publicly) communicating session and key references, member lists, and information like the instance status: If a security model does not make this information readily available to an adversary, a reductionist security argument cannot use such information without becoming formally, and in many cases also effectively, invalid.

Finally, we emphasize that some GKE protocols allow for the concurrent execution of incompatible group operations (e.g., the concurrent addition and removal of a participant) so that different participants might derive keys with different understandings of whom they share it with. This indicates that the **Designated members** category in Table 1 is quite important.

2.5 Our Syntax Proposal

We turn to our syntax proposal that achieves all desirable properties from the above comparison. It is important to note that, in contrast to our *party-centric* perspective in the comparative systematization of this article, we design our model with an *instance-centric* view. That means, we here consider *instances* as the active entities in group key exchange and *parties* as only the passive key-storage in authenticated GKE to which distinct groups of instances have joint access. We discuss the perspectives on the relation between instances and parties in more detail in Appendix B.

A GKE protocol is a quadruple $\mathrm{GKE} = (\mathrm{gen}, \mathrm{init}, \mathrm{exec}, \mathrm{proc})$ of algorithms that generate authentication values, initialize an instance, execute operations according to protocol-dependent commands, and process incoming ciphertexts received from other instances. In order to highlight simplifications that are possible for unauthenticated GKE, we indicate parts of the definition with gray marked boxes that are only applicable to the authenticated case of GKE.

We define GKE protocol GKE over sets $\mathcal{PAU}, \mathcal{SAU}$, $\mathcal{IID}$, $\mathcal{ST}$, $\mathcal{CMD}$, $\mathcal{C}$, $\mathcal{K}$, $\mathcal{KID}$ where $\mathcal{PAU}$ and $\mathcal{SAU}$ are the public and secret authenticator spaces (e.g., verification and signing key spaces, or public group identifier and symmetric pre-shared group secret spaces), $\mathcal{IID}$ is the space of instance identifiers, $\mathcal{ST}$ is the space of instances' local secret states, $\mathcal{CMD}$ is the space of protocol-specific commands (that may include references from $\mathcal{IID}$ to other instances) to initiate operations in a session (such as adding users, etc.), $\mathcal{C}$ is the space of protocol ciphertexts that can be exchanged between instances, $\mathcal{K}$ is the space of group keys, and $\mathcal{KID}$ is the space of key identifiers that refer to computed group keys. The GKE algorithms are defined as follows:

- $\mathrm{gen} \xrightarrow{out} \mathcal{PAU} \times \mathcal{SAU}$. This algorithm takes no input and outputs a pair of public and secret authenticator.
- $\mathcal{IID} \xrightarrow{in} \mathrm{init} \xrightarrow{out} \mathcal{ST}$. This algorithm initializes an instance's secret state.[15]
- $\mathcal{SAU} \times \mathcal{ST} \times \mathcal{CMD} \xrightarrow{in} \mathrm{exec} \xrightarrow{out} \mathcal{ST}$. This algorithm initiates the execution of an operation in a group, e.g., the adding/joining/leaving/removing of instances; affected instances' identifiers can be encoded in the command parameter $cmd \in \mathcal{CMD}$.
- $\mathcal{SAU} \times \mathcal{ST} \times \mathcal{C} \xrightarrow{in} \mathrm{proc} \xrightarrow{out} \mathcal{ST} \cup \{\perp\}$. This algorithm processes a received ciphertext. Return value $\perp$ signals rejection of the input ciphertext.

Interfaces for Algorithms. In contrast to previous works, we model communication to upper layer applications and to the underlying network infrastructure via interfaces that are provided by the environment in which a protocol runs rather than via direct return values. Each of the above algorithms can call the following interfaces (to send ciphertexts and report keys, respectively):

[15] Although exec and proc could implicitly initialize the state internally, we treat the state initialization explicitly for reasons of clarity.

- $\mathcal{IID} \times \mathcal{C} \xrightarrow{in} \mathsf{snd}$. This interface takes a ciphertext (and the calling instance's identifier) and hands it over to the network which is expected to deliver it to other instances for processing. (The receiving instances are encoded in the ciphertext, see below.)
- $\mathcal{IID} \times \mathcal{KID} \times \mathcal{K} \xrightarrow{in} \mathsf{key}$. This interface takes a key identifier and the associated key (and the calling instance's identifier) and delivers them to the upper layer protocol.

Information Encoded in Objects. We assume that certain context information like paired protocol instances and public authenticators is encoded in objects like key identifiers, ciphertexts, and instance identifiers. More precisely, we assume three 'getter functions' $\mathsf{mem}, \mathsf{rec}, \mathsf{pau}$ as follows:

- Function $\mathcal{KID} \xrightarrow{in} \mathsf{mem} \xrightarrow{out} \mathcal{P}(\mathcal{IID})$ extracts from a key identifier the list of identifiers of the instances that are expected to be able to compute the same key.[16]
- Function $\mathcal{C} \xrightarrow{in} \mathsf{rec} \xrightarrow{out} \mathcal{P}(\mathcal{IID})$ extracts the identifiers of the instances that are expected to receive the indicated ciphertext.
- Function $\mathcal{IID} \xrightarrow{in} \mathsf{pau} \xrightarrow{out} \mathcal{PAU}$, in the authenticated setting, extracts the public authenticator of an instance from its identifier.

While this notation is non-standard, it has a number of advantages over alternatives. One advantage has to do with clarity. For instance, function mem is precise about the fact that the list of peers with whom a key is shared is a function of the key itself, represented by its key identifier, and not of the session that established it. Indeed, the latter could establish also further keys with different sets of peers. A second advantage has to do with compactness of description. (This will be discussed in more detail in Section 3.1.) For instance, the notation of the proc algorithm would be more involved if the set of recipient instances of the ciphertext would have to be made explicit as well.

The properties of this syntax proposal are presented in the rightmost column of Table 1. Note that some properties are implied only by the use of this syntax in our partnering, correctness, and security definitions. For example, the flexible consideration of authentication mechanisms and the dispensability of online administrators are due to the game definition in our security notion of Section 4.1. We clarify on the advantages of our model at the end of Section 4.1.

3 Communication Models

The high flexibility in communication (i.e., interaction among participants) in a GKE protocol execution creates various challenges for modeling and defining security of GKE. Firstly, tracing participants of a single global session is a crucial yet typically complex task. Nearly all considered GKE models trace communication partners differently and, in the two-party key exchange literature, there exists an even wider variety of *partnering predicates* (aka. matching mechanisms) for this task. Secondly, normatively defining valid executions of a GKE protocol (versus invalid ones) in order to derive correctness requirements for them is not trivial for a generic consideration of GKE protocols. We note that only five out of the twelve considered models define correctness. In the following, we discuss partnering notions of the analyzed models. Due to space limits we systematize their correctness definitions in Appendix A.

3.1 Partnering

Generally, a partnering predicate identifies instances with related, similar, or even equal contexts of their protocol execution. However, partnering has served many different, somewhat related and somewhat independent purposes in (group) key exchange security models. We distinguish four subtly distinct purposes of partnering.

1. *Forbid trivial key reveals.* In security experiments where an adversary trying to break a challenge key can also reveal "independently" established keys, partnering is used to restrict the adversary's ability to learn a challenge instance's key by revealing partner instances' keys. Here, the partnering predicate must include *at least* those instances that necessarily computed the same key (e.g., group members), but it could be extended to further instances to artificially weaken the adversary (as this restricts its ability to reveal keys), for example, in order to allow for more efficient GKE constructions.

[16] $\mathcal{P}(\mathcal{X})$ denotes the powerset of $\mathcal{X}$.

Partnering/Matching/...	GKE-specific	[BCPQ01]	[BCP02b]	[KY03]	[KS05]	[GBG09]	[CCG+18]	[ACDT19]	[ACC+19]	[BCP01]	[BCP02a]	[KLL04]	[YKLH18]	Desirable	Our model
Defined?	○	●	●	●	●	●	●	⊙	○	●	○	◕	●	●	●
Generic ● or protocol-specific ○	○	●	○	●	●	●	●	●	-	●	-	●	●	●	●
Normative/**P**recise/Retrospect. **V**ariable	○	**N**	-	**N**	**N**	**N**	**N**	**N**	-	**N**	-	**V**	**V**	**P**	**P**
╰ Tight ● (vs. loose ○)	○	●	-	●	○	○	○	●	-	●	-	-	-	-	-
Publicly derivable	○	○	●	⊙	○	⊙	○	●	-	○	-	○	○	●	●
Components included in partnering predicate:															
Transcript															
Matching transcripts	○	⊙	⊙	●	○	○	○	●	-	⊙	-	○	○	○	○
Sequence of matching transcripts	●	●	⊙	○	○	○	○	○	-	●	-	○	○	○	○
Identifiers															
Group identifier	●	○	○	⊙	⊙	●	○	○	-	○	-	●*	●		○
Key identifier	○	○	○	○	○	○	○	○	-	○	-	●*	○		●
Externally input identifier	○	○	○	○	●	○	○	○	-	○	-	○	○		○
Group key															
Whether partners computed a key	○	●	○	◔	●	●	●	○	-	●	-	○	○	⊙	⊙
Whether group computed a key	●	◔	○	○	○	○	○	○	-	◔	-	○	○		○
Whether partners computed same key	○	○	○	○	○	○	●	○	-	○	-	○	○	●	⊙
Members of the group	●	○	○	●	●	●	○	○	-	○	-	●	●	●	⊙

Table 2. Partnering definitions. Notation: ●: yes, ⊙: implicitly, ◕: almost, ◔: partially, ○: no, -: not applicable; (blank): no option clearly superior/desirable.

2. *Detect authentication attacks.* In some explicitly authenticated GKE security definitions, partnering is used to identify successful authentication attacks when one instance completes without there existing partner instances at every designated group member. Here, the partnering predicate must include *at least* those instances belonging to designated members of a computed key, otherwise it is trivial to break authentication. But in this use, compared to use (1) above, the predicate should not be extended to further instances, as actual attacks against authentication might go undetected, if partnering is used for this purpose.
3. *Define correctness.* Partnering is sometimes used to identify instances expected to compute the same key for correctness purposes. In this case, the partnering predicate must include *at most* those instances that are required to compute the same key.
4. *Enabling generic composability.* Partnering also plays a crucial role in the generic composability of (group) key exchange with other primitives: Brzuska et al. [BFWW11] show that a publicly computable partnering predicate is sufficient and (in some cases) even necessary for proving secure the composition of a symmetric key application with keys from an AKE protocol. (Although they consider two-party key exchange, the intuition is applicable to group key exchange as well.)

Even though the first three purposes share some similarities, there are also subtle differences, and defining them via one unified notion can lead to problems.[17]

Our Consideration of Partnering Predicates. We consider the forbidding trivial key attacks ((1) above) as the core purpose of the partnering predicate. If the predicate is defined precisely (i.e., it exactly catches

[17] During the research for this article, we found two recent papers' security definitions for two-party authenticated key exchange that, due to reusing the partnering definition for multiple purposes, cannot be fulfilled: Li and Schäge [LS17] and Cohn-Gordon et al. [CCG+19] both require in their papers' proceedings version for authentication that an instance only computes a key if there exists a partner instance that also computed the key (which is impossible as not all/both participants compute the key simultaneously). Still, the underlying partnering concept suffices for detecting reveals and challenges of the same key (between partnered instances).

the set of same keys that result from a common global session) and is publicly derivable, it also allows for generic compositions of group key exchange with other primitives ((4) above), which we also consider indispensable.

Thereby, it is important to overcome a historic misconception of partnering: for either of the two above mentioned purposes (detection of key reveals and use of established keys in compositions), not the *instances* (that compute keys) are central for the partnering predicate but the *keys themselves* and the *contexts* in which they are computed are. As a result, a partnering definition ideally determines the relation between established keys and their contexts instead of the relation between interacting instances. We elaborate on this in the following: In two-party key exchange, the context of a key is defined by its global session which itself is defined by its two participating instances. In multi-stage key exchange, keys are computed in consecutive stages of a protocol execution. Hence, the context can be determined by the two participating instances in combination with the current (consecutive) stage number. However, in group key exchange—especially if we consider dynamic membership changes—the context of a key is not defined consecutively anymore: due to parallel, potentially conflicting changes of the member set in a protocol execution, it is not necessary that all instances, computing multiple keys, perform these computations in the same order. Consequently, partnering is not a linear, *monotone predicate* defined for instances but an *individual predicate* for each computed group key that reflects its individual context. This context can be protocol-dependent and may include the set of designated member instances, a record of operation by which its computation was initiated, etc. We treat the context information of group keys as an explicit output of the protocol execution also for supporting the use of these keys in upper layer applications (see Table 1).

Models Without Partnering Definitions. Three models do not **define** a partnering predicate at all. In one of these, [ACDT19], a partnering predicate is implicit within their correctness definition. Two of these have no need of partnering since they restrict to (quasi-)passive adversaries [ACDT19] or do not offer adversaries a dedicated access to group keys [ACC$^+$19], however by not defining a partnering predicate, they do not allow for generic composition with symmetric applications. [BCP02a] seemingly rely on an undefined partnering predicate, using the term 'partner' in their security definition but not defining it in the paper. [KLL04] define a partnering predicate of which two crucial components (group and key identifier; see the asterisk marked items in Table 2) are neither defined generically nor defined for the specific protocol that is analyzed in it.

Generality of Predicates. A partnering predicate can be **generic** or **protocol-specific**. From the considered models, only one has a predicate explicitly tailored to the analyzed GKE construction. But many of the generic partnering predicates involve values that are not necessarily part of all GKE schemes (e.g., group identifiers, externally input identifiers, etc.); a sufficiently generic partnering predicate should be able to cover a large class of constructions.

Character of Predicates. Generic partnering predicates can be **normative**, **precise**, or **retrospectively variable**.

Normative predicates define objective, static conditions under which contexts of keys are declared partnered independent of whether a particular protocol, analyzed with it, computes equal keys under these conditions. This has normative character because protocols analyzed under these predicates must implement measures to let contexts that are—according to the predicate—declared unpartnered result in the computation of independent (or no) keys. As almost all security experiments allow adversaries to reveal keys that are not partnered with a challenge key (see Section 4), protocols that do not adhere to a specified normative predicate are automatically declared insecure (because solving the key challenge thereby becomes trivial). These predicates can hence be considered as (hidden) parts of the security definition.

The class of normative predicates can further be divided into **tight** and **loose** ones. **Tight** predicates define only those contexts partnered that result from a joint protocol execution when not attacked by active adversaries. This corresponds to the idea of *matching conversations* being the first tight predicate from the seminal work on key exchange by Bellare and Rogaway [BR94]. Two instances have matching conversations if each of them received a non-empty prefix of, or exactly the same, ciphertexts that their peer instances sent over the network—resulting in partnered contexts at the end of their session.

Matching conversations are problematic for the GKE setting for two reasons. First, achieving security under matching conversations necessitates *strongly unforgeable* signatures or message authentication codes when being used to authenticate the communication transcript.[18] Second, lifting matching conversations directly and incautiously to the group setting, as in [KY03], requires all communication in a global session to be broadcast among all group members so each can compute the same transcript—inducing impractical inefficiency for real-world deployment. If the model's syntax generically allows to (partially) reveal ciphertexts' receivers, as in [ACDT19], pairwise transcript comparison does not require all ciphertexts to be broadcast but the strong unforgeability for authenticating signatures or MACs remains unnecessarily required. Several models [BCPQ01, BCP01] circumvent the necessity of broadcasting all group communication in a matching conversation-like predicate, although their syntax does not reveal receivers of ciphertext: they define two instances and their contexts as partnered if there exists a sequence of instances between them such that any consecutive instances in this sequence have partnered contexts according to matching conversations. (This still needs strongly unforgeable signatures and MACs, however.)

A **loose** partnering predicate is still static but declares more contexts partnered than those that inevitably result in the same key due to a joint, unimpeded protocol execution. This may include contexts of instances that actually did not participate in the same global session, or that did not compute the same (or any) key. An example for loose partnering predicates is *key partnering* [CCG+18] which declares the context of a key as the value of the key itself, regardless of whether it is computed due to participation in the same global session. Clearly, two instances that participated in two independent global sessions (e.g., one global session terminated before the other one begun) should intuitively not compute keys with partnered contexts even if these keys equal. Forbidding the reveal of group keys of intuitively unpartnered contexts results in security definitions that declare protocols 'secure' that may be intuitively insecure. On the other hand, partnering predicates that involve the comparison of a protocol-dependent [GBG09] or externally input [KS05] group identifier are loose because equality of this identifier means being partnered but does not imply the computation of an equal (or any) key.

A **precise** partnering predicate exactly declares those contexts as partnered that refer to equal keys computed *due to* the participation in the same global session. Hence, the conditions for being partnered are not static but depend on the respectively analyzed protocol. As a response to the disadvantages of normative partnering (and in particular tight matching conversations), Li and Schäge [LS17] proposed *original-key partnering* as a precise predicate for two-party key exchange: two instances have partnered contexts if they computed the same key, due to participating in a global session, that they would also have computed (when using the same random coins) in the absence of an adversary. As of yet, there exists no use of original-key partnering for the group setting in the literature, and we discuss drawbacks of this form of precise predicate with respect to the purpose of partnering below.

Variable predicates are parameterized by a customizable input that can be post-specified individually for each use of the model in which they are defined. Hence, these predicates are neither statically fixed nor determined for each protocol (individually) by their model, but can be specified ad hoc instead. As a result, a cryptographer, using a model with a variable predicate (e.g., when proving a construction secure in it), can define the exact partnering conditions for this predicate at will. The main drawback is that different instantiations of the same variable predicate in the same security model can produce different security statements for the same construction. We consider this ambiguity undesirable. Both group identifier and key identifier are left undefined in [KLL04] so they are effectively variable; in [YKLH18] the group ID is outsourced and thus left effectively variable.

Public Derivability of Predicates. A partnering predicate can and—in order to allow for generic compositions—should be **publicly derivable**. That is, the set of partnered contexts should be deducible from the adversarial interaction with the security experiment (or, according to Brzuska et al. [BFWW11], from the communication transcript of all instances in the environment). Only four models considered achieve this as listed in Table 2; ⦿ here refers to the implicit ability to observe whether group keys are computed. Partnering in all remaining models involves private values in instances' secret states. We remark that original-key partnering [LS17] (for two-party key exchange) is the only known precise predicate but it is not publicly computable as it depends on secret random coins.

[18] Note that every manipulated bit in the transcript (including signatures or MAC tags themselves) dissolves partnering.

Components of Predicates. The lower part of Table 2 lists the various parameters on which partnering predicates we consider are defined. These parameters include: the **transcript** of communications, protocol-specific **identifiers**, **external inputs**, the computed **group key**, the set of **group members**, etc.

The two main purposes of partnering ((1) forbidding trivial attacks and (4) allowing for generic composition) use the partnering predicate to determine which keys computed during a protocol execution are meant to be the same and in fact equal (i.e., whether they share the same context). Consequently, an ideal partnering predicate should depend on the context that describes the circumstances under which (and if) the group key is computed. As only for few protocols (e.g., optimal secure ones; cf. [PR18a, PR18b, JS18]) it is reasonable that the entire communicated transcript primarily determines the circumstances (i.e., the context) under which a key is computed, we consider it unsuitable to define partnering based on the transcript generally.

We conclude that it is not the task of the partnering predicate to define security (as normative predicates do). Neither should the variability of partnering predicates lead to ambiguous security notions. Hence, we consider generic, precise, and publicly derivable partnering predicates desirable. With our proposed partnering predicate from Section 3.2, we demonstrate that the problems of the yet only known precise partnering predicate [LS17] can be solved.

3.2 Our Partnering Proposal

Our partnering predicate defines keys with the same explicitly (and publicly; see Section 4.1) output context *kid* partnered:

Definition 1 (Partnering). *Two keys k_1, k_2 computed by instances id_1 and id_2 and output as tuples (id_1, kid_1, k_1) and (id_2, kid_2, k_2) via their* key *interface are partnered iff $kid_1 = kid_2$.*

4 Security Definitions

Although the actual definition of security is the core of a security model, there is no unified notion of "security" nor agreement on how strong or weak "security" should be—in part because different scenarios demand different strengths. Thus we do not aim to compare the strength of models' security definitions, but do review clearly their comparable properties. We focus on the desired security goals, adversarial power in controlling the victims' protocol execution, and adversarial access to victims' secret information. We do not compare the conditions under which adversaries win the respective security experiments (aka. "freshness predicates", "adversarial restrictions", etc.) as this relates to the models' "strength", but we do report on characteristics such as forward-secrecy or post-compromise security.

Security Goals. The analyzed models primarily consider two independent security goals: secrecy of keys and authentication of participants.

Secrecy of keys is in all models realized as **indistinguishability** of actually established **keys** from random values, within the context of an experiment in which the adversary controls protocol executions. During the experiment, the adversary can query a challenge oracle that outputs either the real key for a particular context or a random key; a protocol is *secure* if the adversary cannot dinistinguish between these two. Only one model allows adversaries to query the **challenge** oracle **multiple** times; all others allow only one query to the challenge oracle, resulting in an unnecessary and undesirable tightness loss in reduction-based proofs of composition results.

Key indistinguishability against active adversaries already implies *implicit authentication* of participants. That means keys computed in a session must diverge in case of active attacks that modify communications. Some models require **explicit authentication**: that the protocol explicitly rejects when there was an active attack. ([KLL04] only provide a very specialized notion thereof.) However, the value of explicit authentication in GKE, or even authenticated key exchange broadly, has long been unclear [Sho99]: GKE is never a standalone application but only a building block for some other purpose, providing keys that are implicitly authenticated and thus known only to the intended participants. If the subsequent application aims for explicit authentication of its payload, the diverging of keys due to implicit authentication can be used accordingly.

Security	GKE-specific	[BCPQ01]	[BCP02b]	[KY03]	[KS05]	[GBG09]	[CCG+18]	[ACDT19]	[ACC+19]	[BCP01]	[BCP02a]	[KLL04]	[YKLH18]	Desirable	Our model
Security goals															
Key indistinguishability	○	●	●	●	●	●	●	●	●	●	●	●	●	●	●
∟ Multiple challenges	○	○	○	○	○	○	○	●	○	○	○	○	○	●	●
Explicit authentication	○	●	○	○	●	●	○	○	○	●	○	◔	○		○
Adversarial protocol execution															
All algorithms	○	●	●	●	●	●	●	◕	◕	●	●	●	●	●	●
Instance specific	●	●	●	●	●	●	●	●	●	◔	◔	◔	●	●	●
Concurrent invocations	●	●	●	●	●	●	●	◔	●	○	○	○	●	●	●
Active communication manipulation	○	●	●	●	●	●	◕	○	◔	●	●	●	●	●	●
Adversarial access to secrets															
Corruption of involved parties' secrets	○	●	○	●	●	●	●	-	○	●	●	●	●	●	●
∟ After key exchange	○	●	-	●	●	●	●	-	-	●	●	◕	●	●	●
∟ Before key exchange	○	○	-	○	○	●	●	-	-	○	○	○	●		●
Corruption of independent parties' secrets	○	●	○	●	●	●	●	-	○	●	●	●	●	●	●
∟ Always	○	●	-	●	●	●	●	-	-	○	○	●	●	●	●
Exposure of involved instances' states	○	○	○	○	●	●	◕	●	●	○	●	○	●	●	●
∟ After key exchange	○	-	-	-	●	●	◕	●	●	-	●	-	●	●	●
∟ Before key exchange	○	-	-	-	○	○	●	●	●	-	○	-	○		○
Exposure of independent instances' states	○	○	○	○	●	●	●	●	●	○	●	○	●	●	●
∟ Always	○	-	-	-	●	●	●	●	●	-	○	-	○	●	●
Reveal of independent group keys	○	●	●	●	●	●	●	●	⦿	●	◔	◔	●	●	●
∟ Always	○	●	●	●	●	●	●	●	⦿	●	◔	◔	○	●	●

Table 3. Security definitions. Notation: ●: yes, ⦿: implicitly, ◕: almost, ◔: partially, ○: no, -: not applicable; (blank): no option clearly superior/desirable.

Adversarial Protocol Execution. To model the most general attacks by an adversary, the security experiment should allow adversaries to setup the experiment and control **all** victims' invocations of protocol **algorithms** and operations; all models considered do so. However, in two models the adversary can setup only one group during the entire security experiment (◕); this again introduces a tightness loss in the number of groups for composition results, and means that the use of long-term keys by parties across different sessions, as defined by [ACC+19], cannot be proven secure in the respective model.

Most models allow for **instance-specific** scheduling of invocations. This means that the adversary can let each instance execute the protocol algorithms individually instead of, for example, being restricted to only initiate batched protocol executions (e.g., of all instances involved in a group together). Three models (◔) indeed require that the adversary schedules algorithm and operation invocations that change group membership for all affected instances at once (and not individually); hence, diverging and concurrent operations (e.g., fractions of the group process different actions) cannot be scheduled in these three models. In practice this restriction means that some form of consensus is required (e.g., a central delivery server). While algorithms and operations can be **invoked concurrently** in [ACDT19], this model allows only one of the resulting concurrently sent ciphertexts to be delivered to and processed by the other participants of the same session; this similarly requires some consensus mechanism.

An **active** adversary who **modifies communication** between instances is permitted in almost all models. However, [CCG+18] forbid active attacks during the first communication round, [ACC+19] only allow adversaries to inconsistently forward ciphertexts but not manipulate them, and [ACDT19] require honest delivery of the communication. For the deployment of protocols secure according to the latter two models, active adversaries must be considered impractical or authentication mechanisms must be added.

Adversarial Access to Secrets. GKE models allow the adversary to learn certain secrets used by simulated participants during the security experiment. Below we discuss the different secrets that can be learned and the conditions under which this is allowed. We neglect adversarial access to algorithm invocations' random coins in our systematization as only three models consider this threat in their security experiments [CCG$^+$18, ACDT19, ACC$^+$19].

Corruption of party secrets models a natural threat scenario where parties use static secrets to authenticate themselves over a long period. Corruption is also necessary to model adversarial participation in environments with closed public key infrastructure (see Section 2), allowing the adversary to impersonate some party. Table 3 shows which models allow for corruptions of party secrets *after* and *before* the exchange of a secure group key (i.e., *forward-secrecy* and *post-compromise security*, respectively), and corruptions of independent parties anytime. In [ACDT19] parties do not maintain static secrets so corruption is irrelevant. Two other models do have parties with static secrets but do not provide an oracle for the adversary to corrupt them.[19] Due to imprecise definitions, [KLL04] partially forbids corruptions of involved parties even after a secure key was established, and two other models even forbid corruptions of independent parties before an (independent) secure group key is established. Only three models treat authentication as the sole purpose of party secrets, defining precise conditions that allow corruptions before and after the establishment of a secure group key. As secrecy of a group key should never depend solely on secrecy of independent parties' long-term secrets and *forward-secrecy* is today considered a minimum standard, we deem security despite later corruption of long-term secrets desirable.

Exposure of instance states is especially important in GKE because single sessions may be quite long-lived—such as months- or years-long chats—so local states may become as persistent as party secrets. In most security experiments that provide adversarial access to instance states, their exposure is not permitted before the establishment of a secure group key. Some of these models further restrict the exposure of independent instances' states (e.g., because they were involved in earlier stages of the same session). The three papers that consider *ratcheting* of state secrets allow adversarial access to these states shortly before and after the establishment of a secure group key. [CCG$^+$18] model state expose through the reveal of random coins, which means an exposure at a particular moment reveals only newly generated secrets in the current state, not old state secrets. We consider the ability to expose states independent of and after the establishment of a group key desirable, and leave state exposure before establishment—post-compromise security—as a bonus feature.[20]

The **reveal** of established **group keys** in the security experiment is important to show that different group keys are indeed independent. One motivation for this is that use of keys in weak applications should not hurt secure applications that use different keys from the same GKE protocol. The reveal of keys is furthermore necessary to prove implicit authentication of group keys. Reveals should also be possible to permit composition of key exchange with a generic symmetric key protocol [BFWW11]. Almost all models allow the reveal of different (i.e., unpartnered) group keys unlimitedly. As [BCP02a] and [KLL04] do not define partnering adequately (see Section 3.1), it cannot be assessed which group keys are declared *unpartnered* in their models. The adversary in [ACC$^+$19] is not equipped with a dedicated reveal oracle but since the security in this model is strong enough, the exposure of instance states suffices to obtain all keys without affecting unpartnered keys. [YKLH18] forbid the reveal of earlier group keys in the same session. As unpartnered keys should always be independent we consider it desirable to allow their unrestricted reveal.

4.1 Our Security Proposal

We define security of GKE schemes via a game in which adversaries can interact with these schemes via oracles: For each algorithm of the GKE scheme (see Section 2.5), adversaries can query a corresponding oracle—Init, $\mathsf{Execute}$, $\mathsf{Process}$ in the unauthenticated setting and additionally Gen in the authenticated setting—and thereby choose the respective public input parameters. The public outputs, produced by the respective internal algorithm invocations of these oracles, are given to adversaries via the interfaces snd and key. Adversaries can also query oracles Expose, Reveal and in the authenticated setting additionally $\mathsf{Corrupt}$ to obtain instances' secret states, established group keys, and parties' authentication secrets,

[19] Moreover, in [BCP02b, ACC$^+$19], party secrets cannot be derived via state exposures. Although [ACC$^+$19] allow the exposure of instance states, their syntax, strictly speaking, does not have a method for *using* party secrets in the protocol execution, even though their construction makes use of them (violating the syntax definition).

[20] Note, for example, that post-compromise security is rather irrelevant for short-lived static GKE protocols.

respectively. By querying oracle Challenge, adversaries obtain challenge group keys and win the game if they correctly determine whether these keys were actually established by simulated instances during the game or randomly sampled.

We provide the formal pseudo-code description of this game in Figure 1. The majority of lines of code in this figure only realizes the sound simulation of the game and, therefore, equally appears in our correctness definition from Figure 2. Below we textually describe the remaining parts that constitute restrictions of the adversary and the definition of security.

```
Game $\mathrm{KIND}^b_{\mathrm{GKE}}(\mathcal{A})$
00   $K[\cdot] \leftarrow \bot$; $ST[\cdot] \leftarrow \bot$
01   $SAU[\cdot] \leftarrow \bot$; $CR \leftarrow \mathcal{PAU}$
02   $WK \leftarrow \emptyset$; $CH \leftarrow \emptyset$
03   $CP[\cdot] \leftarrow \emptyset$; $TR[\cdot][\cdot] \leftarrow \bot$
04   $b' \leftarrow_{\$} \mathcal{A}()$
05 · Require $WK \cap CH = \emptyset$
06   Stop with $b'$

Oracle Gen
07   $(pau, sau) \leftarrow_{\$}$ gen
08   $SAU[pau] \leftarrow sau$
09 · $CR \leftarrow CR \setminus \{pau\}$
10   Return $pau$

Oracle Init($iid$)
11   Require $iid \in \mathcal{IID}$
12   Require $SAU[\mathsf{pau}(iid)] \neq \bot$
13   Require $ST[iid] = \bot$
14   $st \leftarrow_{\$} \mathsf{init}(iid)$
15   $ST[iid] \leftarrow st$
16   Return

Oracle Execute($iid, cmd$)
17   Require $ST[iid] \neq \bot$
18   $sau \leftarrow SAU[\mathsf{pau}(iid)]$; $st \leftarrow ST[iid]$
19   $st \leftarrow_{\$} \mathsf{exec}(sau, st, cmd)$
20   $ST[iid] \leftarrow st$
21   Return

Oracle Process($iid, c$)
22   Require $ST[iid] \neq \bot$
23   $sau \leftarrow SAU[\mathsf{pau}(iid)]$; $st \leftarrow ST[iid]$
24   $st \leftarrow_{\$} \mathsf{proc}(sau, st, c)$
25 · If $\nexists iid_s : c = TR[iid_s][iid].peek()$
         $\wedge st \neq \bot$:
26 ·     $WK \xleftarrow{\cup} \{kid \in \mathcal{KID} \setminus CP[iid] :$
                 $\exists iid_{cr} : \{iid, iid_{cr}\} \subseteq \mathsf{mem}(kid)$
                 $\wedge\ \mathsf{pau}(iid_{cr}) \in CR\}$
27   Else: $TR[iid_s][iid].dequeue()$
28   $ST[iid] \leftarrow st$
29   Return

Proc $\mathsf{snd}_{iid}(c)$
30 · For all $iid_r \in \mathsf{rec}(c)$:
31 ·     $TR[iid][iid_r].enqueue(c)$
32   Give $c$ to $\mathcal{A}$

Proc $\mathsf{key}_{iid}(kid, k)$
33   $K[kid] \leftarrow k$
34 · $CP[iid] \xleftarrow{\cup} \{kid\}$
35   Give $kid$ to $\mathcal{A}$

Oracle Reveal($kid$)
36   Require $K[kid] \neq \bot$
37 · $WK \xleftarrow{\cup} kid$
38   Return $K[kid]$

Oracle Challenge($kid$)
39   Require $K[kid] \neq \bot \wedge kid \notin CH$
40   $k_0 \leftarrow K[kid]$
41   $k_1 \leftarrow_{\$} \mathcal{K}$
42 · $CH \xleftarrow{\cup} kid$
43   Return $k_b$

Oracle Expose($iid$)
44   Require $ST[iid] \neq \bot$
45 · $WK \xleftarrow{\cup} \{kid \in \mathcal{KID} \setminus CP[iid] :$
             $iid \in \mathsf{mem}(kid)\}$
46   Return $ST[iid]$

Oracle Corrupt($pau$)
47   Require $SAU[pau] \neq \bot$
48 · $CR \xleftarrow{\cup} \{pau\}$
49   Return $SAU[pau]$
```

Fig. 1. KIND game of GKE modeling unauthenticated or authenticated group key exchange. '·' at the margin highlight mechanisms to restrict the adversary (e.g., to forbid trivial attacks). Almost all remaining code equally appears in game FUNC in Figure 2 and is less important for understanding the security definition. The used variables are explained in Table 4. Line 27 uses iid_s from line 25.

To prevent the trivial solving of challenges, the game forbids the adversary to conduct the following attacks.

1. A group key must not be both revealed via oracle Reveal and queried as a challenge via oracle Challenge (lines 37,42,05).

K	Array of computed group keys
ST	Array of instance states
SAU	Array of secret authenticators
CR	Set of corrupted or external authenticators
WK	Set of weak group keys
CH	Set of keys challenged for $\mathcal{A}$
CP	Set of keys already computed by an instance
TR	Transcript as queue of ciphertexts sent among instances

Table 4. Variables in Figures 1 and 2.

2. After an instance's local state is exposed via oracle Expose, all keys that can be computed by this instance according to their key identifier are declared *weak* (i.e., known to the adversary), if these keys have not already been computed by this exposed instance before (lines 45,34).

As weak keys cannot be challenged but non-weak keys can, we require forward-secrecy—previously computed keys are required to stay secure after an exposure—but not post-compromise security—all future keys of this instance are declared insecure after an exposure. We sketch how to add post-compromise security requirements to this notion below but we consider this weaker security definition sufficient for our demonstration purposes.

Finally, the treatment of active impersonation attacks against the communication in the unauthenticated setting is as follows:

3a) If a ciphertext from an unknown sender (or from a known sender in the wrong order) is processed by an instance without being rejected (lines 30-31,25), then all keys that can be computed by this processing instance according to their key identifier are declared weak, if they have not already been computed by this processing instance before (lines 26,34).

This reflects that in the unauthenticated setting every adversarially generated ciphertext that is accepted as valid by an instance can be considered a successful impersonation of another (honest) instance. Hence, future keys computed by the accepting receiver are potentially known to the adversary.

In the authenticated setting, the set of keys that are declared weak is reduced based on the set of corrupted authenticators. Authenticators are considered *corrupted* if they have not been generated by the challenger (lines 01,09; because thereby they are potentially adversarially generated) or if they have been honestly generated first but then corrupted via oracle Corrupt (lines 01,09,48). As the impersonation of instances with uncorrupted authenticators should be hard in the authenticated setting, active attacks against the communication between instances are treated as follows:

3b) If a ciphertext from an unknown sender (or from a known sender in the wrong order) is processed by an instance without being rejected (lines 30-31,25), then all keys that can be computed by this processing instance according to their key identifier are declared weak, if they have not already been computed by this processing instance before (lines 26,34) *and, according to their key identifier, they are also computable by an instance with a corrupted authenticator* (lines 26,01,09,48).

Definition 2 (Adversarial Advantage). *The advantage of an adversary $\mathcal{A}$ in winning game* KIND *from Figure 1 is* $\mathrm{Adv}^{\mathrm{kind}}_{\mathrm{GKE}}(\mathcal{A}) := |\Pr[\mathrm{KIND}^1_{\mathrm{GKE}}(\mathcal{A}) = 1] - \Pr[\mathrm{KIND}^0_{\mathrm{GKE}}(\mathcal{A}) = 1]|$.

Intuitively, a GKE scheme is secure if all realistic adversaries have negligible advantage in winning this game.

Discussion of the Model. With our proposed model we only want to provide an example definition of security. As mentioned before, we believe that optimal security for GKE is often too strong for practical demands (and hence undesired), and we are not under the illusion that there exists a unified definition of security on which the literature should or aims to agree on. Our contribution is instead that we provide a simple, compact, and precise framework that generically captures GKE and in which the restriction of the adversary (which essentially models the required security) can easily be adjusted. The provided instance of this framework achieves all properties that we identified as desirable in our systematization of knowledge. To name only some advantages of our model: 1. it allows for participation of multiple instances per party per session, 2. it covers unauthenticated, symmetric-key authenticated, and public-key authenticated settings, 3. it imposes no form of key distribution mechanism on GKE constructions and their environment, 4. neither does it impose a consensus mechanism for unifying all session participants' views on the session (although they can be implemented on top), 5. it permits any variant of protocol-specific membership operations, 6. it bases on natural generic interfaces, 7. it outputs the context of group keys along with the group keys themselves to upper-layer applications, 8. it allows for actual asynchronous protocol executions in which not all participants need to agree upon the same order of group key computations, 9. it defines partnering naturally via the context that the protocol itself declares for each group key, 10. it illustrates how a generic model can allow for protocol-dependent definitions of contexts for group keys, 11. it respects the requirements of composition results [BFWW11], 12. it naturally gives adversaries in the security experiment full power in executing the protocol algorithms and determining their public inputs, 13. and it can easily express different strengths of security (see the next paragraph). At the same time, none of the newly captured properties are required to be achieved by analyzed protocols since our model is designed to be *indifferent* to them. We conclude that this model fulfills its main purpose: demonstrating that the desired properties from our systematization framework do not conflict and can hence be achieved simultaneously.

Adding Post-Compromise Security. Extending our proposed security definition to also require secrecy of group keys *after* an involved instance's state was exposed is, due to our flexible key identifiers, straight forward. Intuitively, an instance can recover from a state exposure by contributing new (public) key material to the group. The period between two such contributions by an instance is sometimes called "epoch" (cf. [PR18b, ACDT20]). By encoding in the key identifier of each group key the current epoch of each involved instance, the set of keys that are declared weak due to a state exposure can be reduced accordingly: Instead of declaring all keys weak that an instance can compute in the future after its state was exposed (see item 2 and line 45), only those keys that can be computed by this instance in the current epoch are declared weak. This has the effect that group keys of future epochs are required to be secure again.

5 Concluding Remarks and Open Problems

Our systematization of knowledge reveals some shortcomings in the GKE literature, stemming from a tendency to design a security model hand-in-hand with a protocol to be proven; such a model tends to be less generic, making specific assumptions about characteristics of the protocols it can be used for or the application environment with which it interacts. Sometimes the application environment appeared to be fully neglected. We revisit the underlying concepts of GKE and take into account the broad spectrum of requirements that may arise from the context in which a GKE protocol may be used, such as the type and distribution of authentication credentials of parties, how groups are formed and administered, and whether parties can have multiple devices in the same group. The goal is not to develop a single unified model of group key exchange security, but to support the development of models within the GKE literature that are well-informed by the principle requirements of GKE. Our prototype model demonstrates that these desirable properties of GKE can be satisfied within one generic model, with reduced complexity, increased precision and without restricting its applicability and coverage.

Looking forward, group key exchange is on track for increasing complexity. There now exist prominent applications requiring group key exchange—group instant messaging, videoconferencing—and using a cryptographic protocol in a real-world setting invariably leads to greater complexity in modeling and design. Moreover, the desire for novel properties such as highly dynamic groups and post-compromise security using ratcheting, manifested in proposed standards such as MLS, make it all the more important to have a clear approach to modeling the security of group key exchange.

Among others, our work leaves a number of challenges as open problems. As noted, our model should be seen as a general framework from which versions dedicated to specific use cases can be derived by restricting certain components. Identifying a palette of such submodels that are simultaneously useful and general is challenging, and left for future research. Independently, appropriately integrating the consideration of weakened randomness sources into our model remains an open task. Finally, our work contributes a new model that, so far, has not been tested by analyzing the security of a concrete real-world GKE construction.

Acknowledgments

We thank the reviewers of CT-RSA 2021 for their detailed and helpful comments. B.P. was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 786725 – OLYMPUS. P.R. was supported by the research training group "Human Centered Systems Security" (NERD.NRW) sponsored by the state of North-Rhine Westphalia. D.S. was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2016-05146 and NSERC Discovery Accelerator Supplement grant RGPIN-2016-05146.

A Correctness

Correctness is a quality of protocols that describes the functional guarantees one can expect from their execution. Before discussing these guarantees and the requirements under which one can expect them, we first discuss the role of correctness definitions in models that are aimed for the analysis of another (potentially independent) quality: security.

Safety and Liveness. While usually being defined unified in cryptographic literature, *correctness* is often alternatively considered as the combination of distinct *safety* and *liveness* predicates in other research fields (such as distributed computing). Thereby safety captures consistency guarantees and liveness declares conditions under which actual functionality is guaranteed. Intuitively, liveness expresses under which conditions the protocol execution computes an output (e.g., if the adversary remains passive and the protocol terminates, then a key is computed by all execution participants), and safety expresses which conditions this output must fulfill when it is computed, without requiring that it is ever computed (e.g., if a key is computed by some participants during a session, then it must be the same key for all of them). After explaining the problems with liveness definitions in interactive group protocols, we discuss which of these two components (safety and liveness) are important for defining security.

Liveness in Interactive Group Protocols. For most cryptographic primitives, correctness in the form of liveness is defined by requiring a specific output behavior for a specific execution schedule. In the simplest (non-interactive) case, the nested execution of several algorithms is required to produce a certain output (e.g., for encryption schemes, the decryption of the encryption of a message must produce the message again; for signature schemes, the verification of a message with a signature that was computed for this message must accept; etc). Similarly, for two-party key exchange (if correctness is defined) the honest execution of the protocol is usually required to establish the same key for both execution participants (see e.g., [BR94]).

For protocol executions (i.e., sessions) with multiple participants that can (all) actively influence the output of this execution (e.g., dynamic GKE in which all group members can initiate membership changes), there may exist multiple different execution schedules that results in the same output (e.g., the same group can compute the same key independent of the order of membership changes). Simultaneously, there may not exist only one "correct" output for each specific execution schedule. (Consider, for example, a dynamic GKE session in which two group members concurrently initiate conflicting changes of membership. The resulting set of members and the group key, output by participating members, may differ for different GKE schemes, or even for different executions of the same GKE scheme.) Finally, as argued in Section 2, we consider it undesirable to explicitly model specific operations (and their impact on the session) in GKE syntax definitions. Specifying the exact output of specific execution schedules as part of a correctness definition is thereby undesirable as well. It is, consequently, complicated (if not impossible) to formulate a complete and static definition of execution schedules with their required output in a liveness definition for generic GKE protocols.

Correctness	GKE-specific	[BCPQ01]	[BCP02b]	[KY03]	[KS05]	[GBG09]	[CCG+18]	[ACDT19]	[ACC+19]	[BCP01]	[BCP02a]	[KLL04]	[YKLH18]	Desirable	Our model
Defined	○	○	○	●	●	●	○	●	○	○	○	○	●	●	●
Requirements															
Honest transcript delivery	○	-	-	⦿	●	●	-	●	-	-	-	-	○		○
Two instances are partnered	○	-	-	●	○	○	-	⦿	-	-	-	-	●	⦿	⦿
All group instances are partnered	●	-	-	○	○	○	-	○	-	-	-	-	○		○
A key is computed	○	-	-	●	○	○	-	○	-	-	-	-	●	⦿	⦿
All participating instances share ≧	○	-	-	○	●	○	-	○	-	-	-	-	○		○
Keys are partnered	○	-	-	⦿	○	○	-	⦿	-	-	-	-	⦿	●	●
Guarantees															
Same key	○	-	-	●	●	●	-	●	-	-	-	-	●	●	●
Non-zero key	○	-	-	●	○	○	-	○	-	-	-	-	○		○
Keys are partnered	○	-	-	⦿	⦿	●	-	⦿	-	-	-	-	○		⦿

Table 5. Correctness definitions. Notation: ●: yes, ⦿: implicitly, ◕: almost, ◔: partially, ○: no, -: not applicable; (blank): no option clearly superior/desirable.

Correctness and Security. Intuitively, correctness and security could be considered independent qualities: a scheme can be correct but insecure, incorrect but secure, correct and secure, or neither correct nor secure. Indeed, "lazy" (i.e., incorrect) schemes not doing anything trivially (can be adapted to) reach most security properties immediately. Thereby the question may arise why one needs to consider correctness in a security analysis at all.

On the one hand, there exist security definitions that cannot be achieved by schemes that provide certain correctness guarantees and consequently there exist correctness definitions that schemes cannot achieve when also providing certain security guarantees. Hence, for demonstrating the usefulness of a security definition, either a correctness definition (including liveness guarantees) that is (believed to be) compatible with this security should be provided. Alternatively, a construction accordingly secure, implicitly showing its correctness guarantees, should be provided along with this security definition.

On the other hand, based on the idea that an adversary in a security experiment only needs to be restricted when attacking a correct scheme, the idea of *security up to correctness* (more precisely "indistinguishability up to correctness") was formally proposed by Rogaway and Zhang [RZ18]. Their approach for defining security for a given correctness definition is intuitively as follows: in the security experiment adversaries are given full power over the execution of a primitive simulated by a challenger, potentially including access to special oracles that provide secrets of instances involved in the execution. The challenger then restricts this power by "silencing" (i.e., suppressing) only adversarial queries for which the challenger's response is already determined by the correctness definition. This restriction prevents the trivial solution of embedded challenges in the security experiment (e.g., if correctness requires partnered keys k', k^* to equal $k' = k^*$, then if k^* is a challenge, revealing k' determines the challenge's solution). The major advantage of this approach is reduced ambiguity in defining security. Note that one can still define oracles for adversarial access to secrets freely.

It is immediate that the approach of Rogaway and Zhang [RZ18] requires the definition of correctness for obtaining a definition of security. However, the challenger, when silencing oracle queries based on their determination, does not need to know in advance *when* the protocol computes an output (as the outputs of the protocol execution are directly observable for the challenger). The challenger only needs to know how a computed output relates to other values. This relation is expressed by the definition of safety already.

We note that the overall idea behind *security up to correctness*—silencing oracles or penalizing queries to them, based on the safety guarantees one expects—is not only the basis of this formal definition process but also the intuition behind most informal, manual approaches for defining security.

We conclude that defining correctness in definitions of security *can be desirable* in the form of *liveness* for demonstrating that the security can be met by useful constructions, and *is desirable* in the form of *safety* for guiding restrictions of the adversary in the security experiment. As a definition of security normally comes with an analysis of a (useful) protocol, the definition of liveness can usually be neglected. Furthermore, due to the sketched problems with (complete and static) liveness definitions, we focus on safety guarantees in the following.

A Clean Definition of Safety. The only functionality GKE protocols should provide generically is the establishment of group keys. Thereby keys should be the same if they were output by participants of a joint session and meant to be established as a shared group key. The generic mechanism for tracing commonly computed group keys is the partnering predicate. Consequently, partnered keys (or keys of partnered instances) should be equal in order to fulfill safety (see Table 5).

As our syntax from Section 2.5 allows upper layer protocols to identify keys (with the key identifier), it is actually not necessary that two session participants compute two distinct group keys in the same order. Partnering of instances, however, seems to be a linearly sequential property (if two instances were partnered once and their partnerships is disrupted, they will not become partners again). Partnering of group keys is, in contrast, a time-independent property (if two keys are partnered, they will remain partnered forever). As a consequence, we consider keys (as opposed to instances) being partnered desirable as the requirement of the safety definition.

Discussion of Models. As it can be seen in Table 5, two definitions [KY03, YKLH18] match our idea of requirements for correctness (note that both define partnering w.r.t. instances and [KY03] require honest delivery for being partnered). In addition to requiring key equality as safety guarantee, [KY03] furthermore demand the computation of a non-trivial *real* key.

The remaining three correctness definitions require an honest protocol execution (satisfying their explicit or implicit partnering predicates) for the computation of same keys. The above described problem of declaring an execution *honest* (and requiring a certain output from it) is resolved in these three definitions as follows: The former two models [KS05, GBG09] solely capture static GKE (in which the execution schedule can be predetermined) and the latter one forbids the concurrent processing of conflicting operations in groups such that a fixed, shared output can be expected.

A.1 Our Correctness Proposal

We here shortly describe the details of our correctness definition that is based on the game from Figure 2.

Definition 3 (Safety). *A GKE protocol* GKE *is safe if for all adversaries $\mathcal{A}$ against game* FUNC *according to Figure 2 it holds that* $\Pr[\mathrm{FUNC}_{\mathrm{GKE}}(\mathcal{A}) = 1] = 0$*.*

We declare a GKE protocol incorrect if two computed non-trivial keys (i.e., $k \neq \perp$) with equal key identifiers differ (see Figure 2 line 74).

Beyond the core functionality of establishing group keys, our syntax allows upper layer protocols to derive the set of designated group members for each established group key. For this additional functionality we require that group keys, output by an instance via interface `key`, are actually designated to this instance (Figure 2 line 75). We emphasize that this requirement is indeed a property of safety (and not security) as it does not hinder any independent exposable instance to be able to compute the key internally without outputting it.

All remaining pseudo-code in Figure 2 only describes the necessary framework for executing the algorithms inside the oracles that are provided to adversaries. Consequently, these parts of the Figure are irrelevant for understanding the definition and equally appear in the security experiment in Figure 1.

B The Relation between Parties and Instances

Even with our systematic approach some semantic interpretations of modeling choices are not ultimately determined. One example that we discuss here is that it remains debatable whether *local instances* or *parties* are the *actual participants* of sessions. In the **instance-centric perspective**, instances are *active* entities that may or may not represent *passive* parties. The term party would thereby only refer to static

```
Game FUNC$_{\mathrm{GKE}}(\mathcal{A})$
    $K[\cdot] \leftarrow \bot$; $ST[\cdot] \leftarrow \bot$
    $SAU[\cdot] \leftarrow \bot$
    Invoke $\mathcal{A}()$
    Stop with 0

Oracle Gen
    $(pau, sau) \leftarrow_{\$}$ gen
    $SAU[pau] \leftarrow sau$
    Return $pau$

Oracle Init($iid$)
    Require $iid \in \mathcal{IID}$
    Require $SAU[\mathsf{pau}(iid)] \neq \bot$
    Require $ST[iid] = \bot$
    $st \leftarrow_{\$}$ init($iid$)
    $ST[iid] \leftarrow st$
    Return

Proc snd$_{iid}(c)$
    Give $c$ to $\mathcal{A}$

Oracle Execute($iid$, $cmd$)
    Require $ST[iid] \neq \bot$
    $sau \leftarrow SAU[\mathsf{pau}(iid)]$; $st \leftarrow ST[iid]$
    $st \leftarrow_{\$}$ exec($sau$, $st$, $cmd$)
    $ST[iid] \leftarrow st$
    Return

Oracle Process($iid, c$)
    Require $ST[iid] \neq \bot$
    $sau \leftarrow SAU[\mathsf{pau}(iid)]$; $st \leftarrow ST[iid]$
    $st \leftarrow_{\$}$ proc($sau$, $st, c$)
    $ST[iid] \leftarrow st$
    Return

Proc key$_{iid}(kid, k)$
  · Reward $K[kid] \notin \{\bot, k\}$
  · Reward $iid \notin \mathsf{mem}(kid)$
    $K[kid] \leftarrow k$
    Give $kid$ to $\mathcal{A}$
```

Fig. 2. FUNC game of GKE describing correctness in the form of safety. Gray marked code is only applicable in the authenticated setting. Lines marked with '·' at the left margin highlight safety requirements. For an explanation of the used variables see Table 4.

objects that embody authentication secrets whereas instances would be considered active entities that potentially make use of these static objects. In the **party-centric perspective**, instances are only realizations of parties' participation in sessions. Parties would thereby *actively* outsource the execution of algorithms, necessary for participating in sessions, to their instances. As this distinction may appear as quibble on a language level, we clarify technical differences below.

Instance-Centric Perspective. We first consider a setting that neither of the publications, analyzed in the main body of this article, models: unauthenticated GKE. In unauthenticated GKE the concept of parties remains unclear. While parties usually refer to permanent entities, participants of unauthenticated GKE session only temporarily exist during their participation. Additionally, parties are often linked to authentication secrets. According to this interpretation, parties do not exist in unauthenticated GKE such that only instances could be participants of sessions. Secondly, if instances are the actual participants of sessions, it can be reasonable to allow simultaneous participation of multiple instances per party in sessions. This can be a useful feature for multi-device settings (e.g., in instant messaging). Thirdly, GKE is never a primitive used by humans but a tool used by other cryptographic applications. GKE instances, executing the participation in a GKE session, may thereby be initiated by instances of the upper layer cryptographic application rather than by a central GKE party. Thereby the concept of a party collapses to some static information (e.g., authentication secrets) that these GKE instances use.

Party-Centric Perspective. If parties are active participants of sessions (i.e., *members* thereof) trough their instances, participation in a single session through multiple instances per party is contradictory. Therefore settings in which human users participate in a session with multiple devices would need to be modeled by declaring each device of a user an individual party. We note that sharing authentication secrets between these "device-parties" would be considered insecure in all analyzed models. Nevertheless, the concept of active parties and their realization through controlled instances intuitively describes the idea of (authenticated) entities using the GKE protocol for deriving group keys more comprehensibly than the instance-centric perspective. Even if GKE instances are locally initiated by distributed instances of the cryptographic application that uses the group keys, one can trace the initialization of these application instances back to a central active party. As application executions of the same party are in reality usually initiated and managed centrally, conceiving this central party as an active entity instead of its multiple local instances appears to be practical.

We believe that neither of both perspectives describes the essential truth and preferences for either of them may depend on the individual understanding of what *parties* and *instances* are in reality. Nevertheless, for modeling GKE one needs to commit to either of them, implicating the respective consequences described above. As all analyzed models adopt the party-centric perspective (see the first three rows in Table 1), the terminology in our comparison is chosen accordingly. However, since this intuitively restricts parties from participating in sessions with multiple of their instances, we chose the instance-centric perspective in our proposed model.