Algebraic Adversaries in the Universal Composability Framework

Michel Abdalla[1,2] ⓘ, Manuel Barbosa[3] ⓘ, Jonathan Katz[4], Julian Loss[5] ⓘ, and Jiayu Xu[6] ⓘ

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@gmail.com
[2] DFINITY, Zürich, Switzerland
[3] University of Porto (FCUP) and INESC TEC, Porto, Portugal
mbb@fc.up.pt
[4] University of Maryland, College Park, USA
jkatz2@gmail.com
[5] lossjulian@gmail.com*
[6] Algorand, USA**
jiayux@uci.edu

Abstract. The algebraic-group model (AGM), which lies between the generic group model and the standard model of computation, provides a means by which to analyze the security of cryptosystems against so-called *algebraic* adversaries. We formalize the AGM within the framework of universal composability, providing formal definitions for this setting and proving an appropriate composition theorem. This extends the applicability of the AGM to more-complex protocols, and lays the foundations for analyzing algebraic adversaries in a composable fashion. Our results also clarify the meaning of composing proofs in the AGM with other proofs and they highlight a natural form of independence between idealized groups that seems inherent to the AGM and has not been made formal before—these insights also apply to the composition of game-based proofs in the AGM. We show the utility of our model by proving several important protocols universally composable for algebraic adversaries, specifically: (1) the Chou-Orlandi protocol for oblivious transfer, and (2) the SPAKE2 and CPace protocols for password-based authenticated key exchange.

---

* Work done while at the University of Maryland.
** Work done while at George Mason University.

1 Introduction

Security proofs are often carried out in idealized models that seek to capture certain classes of adversarial behavior. Examples include the random-oracle model [6], in which the attacker is assumed to treat a hash function as an ideal random function; the ideal-cipher model, in which the attacker is assumed to treat a block cipher as an ideal keyed permutation; and the generic-group model (GGM) [25, 26], where the attacker is assumed to treat group elements as abstract identifiers and group operations as black-box operations on those identifiers.

Cryptographers continually seek to refine these models, making them more expressive so they capture larger classes of algorithms and thus come closer to modeling adversaries performing arbitrary computation. With this motivation in mind, Fuchsbauer et al. [16] (based on ideas of Abdalla et al. [2]) proposed the *algebraic-group model* (AGM) as a more expressive version of the GGM. Roughly, the AGM considers *algebraic* adversaries that compute group elements via a sequence of "generic" group operations, but which—in contrast to the GGM—are allowed to utilize the *actual bitstrings* representing group elements in the course of their computation. This model is strictly stronger than the GGM; for example, index-calculus algorithms that apply to certain classes of groups are algebraic and hence allowed in the AGM, even though they are ruled out in the GGM by known lower bounds on the hardness of the discrete-logarithm problem in that model. The AGM has been used to show equivalence of various number-theoretic assumptions [4, 5, 16] and to prove security of SNARKs [14, 16, 23] and blind signatures [17]. An extension called the *strong* AGM has recently been used to prove hardness of the repeated squaring assumption underlying timed commitments and related primitives [21].

Notably, none of the aforementioned results provide any guarantees of security under composition with other protocols (whether proven secure in the AGM or not). Here, we lay the foundations for a composable treatment of algebraic adversaries by formalizing the AGM within the framework of *universal composability* (UC) [10] and proving a corresponding composition theorem. This involves not only formalizing a number of subtle issues related to the AGM itself (which may be of independent interest for subsequent work in the AGM), but also making a number of careful design decisions in defining what algebraic adversaries mean in the UC framework, in part to ensure that a suitable composition theorem holds. We discuss this in more detail in the following section.

We demonstrate the utility of our model by proving several important protocols universally composable for algebraic adversaries. Specifically, we prove security of (1) the Chou-Orlandi protocol for oblivious transfer [15], and (2) the SPAKE2 and CPace protocols for password-based authenticated key exchange [3, 19] in our model. We describe these results further in Section 1.2.

1.1 Defining the AGM Within the UC Framework

We first define some notation and terminology related to the AGM that suffices to understand the discussion that follows. (Our treatment here is deliberately informal, and we refer the reader to Section 2 for technical details.) Fix a group $\mathbb{G}$. An *algebraic representation* of $h \in \mathbb{G}$ with respect to a list of elements $g_1, \ldots, g_n \in \mathbb{G}$ is a tuple $(x_1, \ldots, x_n) \in \mathbb{Z}^n$ with $h = \prod_{i=1}^{n} g_i^{x_i}$. Roughly speaking, the AGM considers adversaries that are *algebraic* (with respect to $\mathbb{G}$), meaning that if an adversary $\mathcal{A}$ outputs a group element $h \in \mathbb{G}$, then $\mathcal{A}$ must also output an algebraic representation of h with respect to the set of group elements (which we call a *base*) that $\mathcal{A}$ has been given as input thus far.

We generalize the AGM to the standard UC framework by restricting our attention to algebraic attackers.[1] While this is a natural idea, it involves dealing with a number of subtle technical issues. First of all, to make this notion meaningful it is not sufficient to restrict the adversary to be algebraic; rather, we require the *environment* to be algebraic as well. Moreover, in order for composition to possibly hold, we must also require the *simulator* used in proving security to be algebraic. That is, in the *UC-AGM* a protocol π securely realizes

---

[1] One can consider formalizing the AGM within the UC framework by introducing a functionality $\mathcal{F}_{\mathsf{AGM}}$ that "forces" arbitrary algorithms to behave algebraically by registering group elements and their representations in a central repository. This has a number of disadvantages that we discuss in Appendix A. Our approach is closer to the spirit of the AGM, which idealizes groups by quantifying over restricted classes of adversaries.
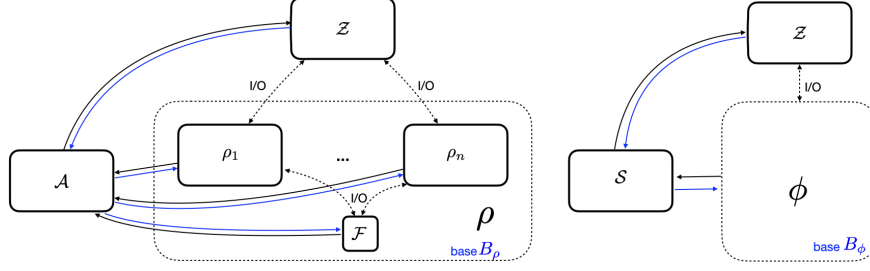
Fig. 1. The AGM UC real and ideal worlds. Blue arrows impose algebraic behaviour.

a functionality $\mathcal{F}$ if, for any efficient algebraic adversary $\mathcal{A}$, there is an efficient algebraic simulator $\mathcal{S}$ such that no efficient algebraic environment can distinguish the execution of $\mathcal{A}$ with π from the execution of $\mathcal{S}$ with $\mathcal{F}$. Under this definition, we can indeed prove that a UC-style composition theorem holds in the UC-AGM.

Our definition of an algebraic algorithm makes a distinction between adversarial entities (real and ideal world adversaries and environments) and non-adversarial entities (uncorrupted protocol participants and ideal functionalities). In the real world, we require the adversary to behave algebraically when it delivers group elements to uncorrupted participants and to ideal functionalities (when the proof is carried out in a hybrid real-world); moreover, we also require the environment to behave algebraically when it delivers group elements to the adversary, but not the converse. Algebraic behavior is defined within the context of a UC AGM proof by specifying what set of group elements occurring during the protocol execution in the real-world must be used by the environment and by the adversary as a base for the provided group element representations. When this is the empty set, we recover the standard UC framework. The natural definition for this set is to include in it *all* the group elements that are produced by non-adversarial entities.

We show this pictorially in Figure 1, where the blue arrows denote that all group elements that are delivered must be accompanied by an algebraic decomposition with respect to base B_ρ, which are the group elements produced by the machines within the boundary of ρ. Note that, in the figure, ρ is specified in a hybrid world that includes a functionality $\mathcal{F}$, which is within that boundary. The ideal world restrictions are equivalent, replacing $\mathcal{A}$ with $\mathcal{S}$ and ρ with the ideal-world protocol ϕ that is realized by ρ.

Formally, the quantification of the UC-emulation notion is subtle. As in UC, we require for all adversaries $\mathcal{A}$, the existence of a simulator $\mathcal{S}$, such that for all enviroments $\mathcal{E}$ the real and ideal worlds are indistinguishable. However, the simulator is only required to work if the pair $(\mathcal{E}, \mathcal{A})$ satisfies the algebraic restrictions specified in the real world. Intuitively, the extra power of the simulator comes from the fact that $\mathcal{E}$ is bound to behave algebraically when interacting with $\mathcal{A}$ and, furthermore, that $\mathcal{A}$ will also behave algebraically if the simulator runs it internally. A caveat is that the simulator must also ensure that $(\mathcal{E}, \mathcal{S})$ satisfy the algebraic restrictions in the ideal world. However, in the most common case when the simulator is interacting with an ideal functionality, if this interaction does not involve group elements, then the algebraic requirement is not a restriction on the simulation strategy (this is the case in all our proofs for concrete protocols).

The UC AGM composition theorem then states, as expected, that $\rho^\pi \sim \rho^{\mathcal{F}}$ if $\pi \sim \mathcal{F}$. We show this scenario in Figure 2. Again the quantification is subtle. The composition theorem guarantees only hold if we restrict our quantification to match the emulation guarantee provided by π: i.e., we have that $\rho^\pi \sim \rho^{\mathcal{F}}$ with respect to pairs $(\mathcal{E}, \mathcal{A})$ that adhere to the base B_π when interacting with machines in π. Note that this means, in particular, that the attacker cannot use group elements produced in ρ when attacking π, unless it is able to provide a representation according to B_π.

The companion UC AGM transitivity theory further highlights a natural notion of independence between UC AGM proofs. Suppose that $\rho^{\mathcal{F}}$ is known to UC AGM emulate some functionality $\mathcal{G}$. Transitivity intuitively implies that $\rho^\pi \sim \mathcal{G}$ if $\rho^\pi \sim \rho^{\mathcal{F}}$. We show that this is the case also in the UC AGM setting, if we restrict the quantification over $(\mathcal{E}, \mathcal{A})$ to those attackers that independently meet the AGM restrictions imposed by the proofs of both π and ρ. This means providing algebraic representations to parties executing π with respect to a base B_π defined in the proof of π and, similarly, respecting the algebraic base B_ρ when interacting with
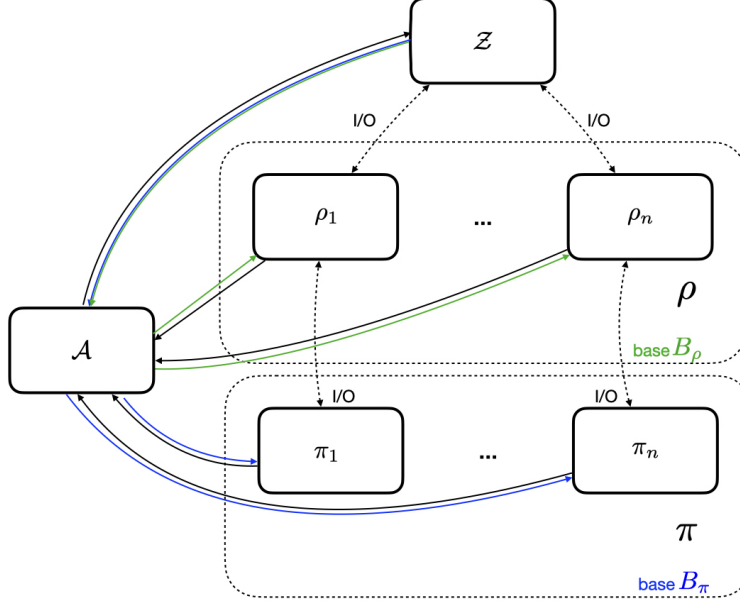
3

Fig. 2. The UC AGM restrictions for composition (blue) and transitivity (blue/green).

parties executing ρ. This restriction means that AGM UC composition works as expected for protocols that operate on groups that can be assumed to be *independent*.

In Section 2.3 we give full technical details and also show that proofs in the UC AGM naturally compose with proofs in the plain UC model; as expected, the composed protocols can only be shown to be secure in the UC AGM setting. We also show that the standard approach of writing UC proofs w.r.t. a dummy adversary still applies in the UC AGM setting.

Discussion. Our theorems show that one should be very careful when composing proofs in the AGM, and not only in the UC setting. For example, when composing game-based reductions carried out in the AGM, the same issues arise. Intuitively, composition can only be guaranteed when the AGM assumptions do not interact badly with each-other, i.e., interacting with one protocol does not allow an attacker to override the extractability assumption that is being captured by the AGM in the proof of another protocol. In practice this seems to imply excluding attackers that take group elements from one protocol and use them to attack another protocol (unless of course the algebraic construction of those elements can be explained with respect to the set of bases defined by the target protocol alone).

Interestingly, in recent independent work Kerber, Kiayias and Kohlweiss [?] encouter a manifestation of the same problem in the constructive cryptography framework. In this work, the authors propose a general notion of proofs w.r.t. knowledge assumptions, which generalizes the AGM: adversaries provide the relevant extractable information when interacting with the protocol. Their goal is to study the composition of protocols that rely on different knowledge assumptions. It is beyond the scope of this paper to make a detailed comparison, since the approaches rely on different compositional frameworks and have different goals, but it is clear that the same restrictions must be imposed in the composition theorem to enable a proof; quoting from the paper: "*Care must be taken that knowledge stemming from one knowledge assumption does not give an advantage in another. . . we conjecture that multiple instances with the AGM with independently sampled groups are sufficiently independent.*"

To conclude, we do not see the restrictions in the UC AGM composition theorems as a limitation of our work, but rather as a limitation inherent to proofs in idealized models—for example, it is easy to establish a parallel with the random oracle model in the UC setting, where the need for independent RO instances is well known [13]. On the contrary, we believe that an important contribution of our work is to clarify

what this limitation means for proofs in the AGM. To overcome these limitations, and similarly to proofs in the random oracle model, one can prove multiple protocol executions secure simultaneously. At the very least, it is important to ensure that AGM UC proofs are carried out with respect to multi-session ideal functionalities, so that multiple executions of the same protocol can be guaranteed to compose securely. We adopt this approach in our proofs. Another option is to strengthen the proofs of each protocol to consider a global/shared source of bases along with a more powerful composition theorem, similarly to UC with global setup [11]. We leave exploring this option as an interesting and important direction for future work.

1.2 Proofs of Security in the UC-AGM

In addition to defining the UC-AGM framework, we also show that several important protocols from the literature—which were previously lacking full proofs of security in the UC framework—can be proven secure in our model.

The Chou-Orlandi protocol. Chou and Orlandi [15] proposed a simple and elegant protocol for oblivious transfer and claimed that it was universally composable (with adaptive corruptions) under a suitable assumption in the random oracle model. Unfortunately, subsequent works [8, 18, 20] uncovered several problems with their proof. While these subsequent works also showed how to address some of these issues, and/or presented modified protocols that could be proven secure, there seems to be no way of proving the original Chou-Orlandi protocol universally composable, even in the random oracle model.

We show that the original Chou-Orlandi protocol can be proven secure in the UC-AGM, based on the discrete-logarithm assumption in the random oracle model. We refer to Section 3 for a high-level overview of the proof and further details.

The SPAKE2 and CPace protocols. SPAKE2 [3] and CPace [19] have attracted a lot of interest recently due to their consideration for standardization by the IETF. The selection process explicitly considered whether these protocols were universally composable, which turned out to be a surprisingly difficult question to answer.[2]

Abdalla et al. [1] recently proved that these protocols are universally composable with respect to a *relaxed* version of the standard functionality for password-based authenticated key exchange (PAKE) that, roughly speaking, allows the adversary to delay its password guess for a session until an arbitrary time after that session ends. The full implications of relying on that relaxed functionality are unclear; in particular, although Abdalla et al. [1] showed that adding a key-confirmation step lifts a UC PAKE protocol to one that provides explicit entity authentication, we do not know if this is the case when we start from a PAKE protocol that only realizes the relaxed PAKE functionality.

In this work, we improve upon these results by showing that both SPAKE2 and CPace are universally composable with respect to the *original* PAKE functionality [12] when we restrict our attention to algebraic adversaries. Interestingly, our proofs are significantly simpler than those of Abdalla et al. [1], since the simulator in our case can leverage the fact that the adversary is algebraic to directly extract password guesses, rather than performing an indirect extraction using the random oracle.

In addition, we also demonstrate that an important variant of SPAKE2, known as SPAKE1, is secure in the UC-AGM. SPAKE1, in contrast to SPAKE2, does not include the password as input to the final key-derivation function, and thus may be advantageous relative to SPAKE2 with regard to side-channel attacks targeting the key-derivation step. Prior to this work, SPAKE1 was not known to satisfy the standard notions of security for game-based and UC PAKE. In particular, it was not known to guarantee even the weaker notion of forward secrecy, in which the attacker can only learn passwords for sessions in which it played the role of a passive eavesdropper.

1.3 Related Work

We are not aware of any prior work modeling algebraic adversaries in the UC framework, however a few works have considered generic groups and other idealized models in that setting. Larangeira and Tanaka [22] analyze

---

[2] For a review of the security proofs available for both protocols at the time, see https://mailarchive.ietf.org/arch/msg/47pnOSsrVS8uozXbAuM-alEk0-s.

universally composable non-committing encryption schemes in the GGM and the generic-ring model (GRM). However, they leave the modeling of the GGM/GRM in the UC framework informal, and in particular do not prove that composition holds in their setting. Bradley et al. [7] prove security of a strong asymmetric PAKE protocol against a generic-group adversary in the UC framework, but their treatment is also informal; in particular, their protocol is split into an "offline part" and an "online part," with the GGM used only in the former, and it is unclear how these two parts are defined for general protocols or what the implications are for composition. Naor et al. [24] model generic-group adversaries in the UC framework by introducing a generic-group functionality $\mathcal{F}_{\mathsf{GGM}}$ in a way similar in spirit to the approach involving the $\mathcal{F}_{\mathsf{AGM}}$ functionality described earlier that we ultimately rejected. A similar approach was followed in [?] for the analysis of time-lock puzzles in the UC setting.

1.4 Overview of the Paper

Section 2 introduces the UC-AGM model. The treatment is quite low level to ensure the appropriate level of formalization and Appendix B.1 reviews the UC framework for this reason.Section 3 then presents a proof of the Chou-Orlandi protocol in the UC-AGM. Next, Section 4 proves security of SPAKE1, SPAKE2, and CPace in the new model. Finally, Appendices B to D provide detailed proofs for theorems in Sections 2 to 4.

2 Defining Algebraic Adversaries in the UC Framework

In this section, we introduce the UC-AGM framework that incorporates algebraic adversaries into the UC framework. We provide a brief overview of the UC framework [10] in Section 2.1; for a more detailed description, see Appendix B.1 . In Section 2.2 we formally define algebraic adversaries and introduce the notation of AGM-emulation that underlies the UC-AGM. We also show there that, analogous to the UC framework, it suffices to consider "algebraically dummy" adversaries when proving AGM-emulation. We prove a composition theorem for the UC-AGM in Section 2.3.

For simplicity, our treatment of the UC-AGM is based on the so-called simplified UC framework [10, Section 2] where the number of parties, their identities, program code, and connectivity are all fixed in advance. In Appendix B.2 we explain how the UC-AGM can be extended to the full UC framework.

2.1 Overview of the UC Framework

A *protocol* consists of a number of machines (or parties) with unique identities, each of which represents some computational entity. Protocol machines communicate with each other via messages labeled input or subroutine-output. In an *execution of the protocol*, two additional machines (whose identities are distinct from any protocol machines) are added: the *environment* $\mathcal{E}$ and the *adversary* $\mathcal{A}$. (Below we assume that $\mathcal{E}$ has identity 0 and $\mathcal{A}$ has identity 1.) The environment $\mathcal{E}$ can send input messages to $\mathcal{A}$ and a subset of the protocol machines (called *main machines*), and protocol machines can send subroutine-output messages to $\mathcal{E}$; the adversary $\mathcal{A}$ can send backdoor messages to $\mathcal{E}$ and all protocol machines, and receive backdoor messages from all protocol machines.

The notion of *UC emulation* involves two protocols, π and ϕ. We say that π emulates ϕ if for any efficient adversary $\mathcal{A}$ in an execution of π, there is an efficient adversary (called the *simulator*) $\mathcal{S}$ in an execution of ϕ that "simulates" the environment's view, in the sense that no efficient environment can distinguish an execution of π with $\mathcal{A}$ from an execution of ϕ with $\mathcal{S}$. A particularly important example of UC emulation is *realizing an ideal functionality*, in which the emulated protocol $\mathrm{IDEAL}_{\mathcal{F}}$ consists of an incorruptible *ideal functionality* $\mathcal{F}$, and the main machines are dummy parties that simply pass messages between the ideal functionality and the environment.

2.2 UC Emulation in the Algebraic Group Model

In this work we put forth a notion of UC emulation (called *AGM-emulation*) in which the adversary is restricted to be algebraic. To this end, we first introduce the concept of *algebraic adversaries* [16]. At a high level, an algebraic adversary has an additional auxiliary tape on which it writes the representation of any

group element it outputs on (some of) its other tapes.[3] We assume for simplicity that the group $\mathcal{G} = (\mathbb{G}, g, p)$ under consideration is cyclic with known order p, though neither of these assumptions is essential.

Definition 1. *Suppose an execution of protocol π involves protocol machines sending elements in group $\mathcal{G} = (\mathbb{G}, g, p)$ (henceforce "protocol π involves group $\mathcal{G}$").[4] A pair of environment $\mathcal{E}$ and adversary $\mathcal{A}$ (in π's execution) is $(\mathcal{G}, \pi)$-algebraic if it satisfies the following:*

(1) $\mathcal{A}$ has a special output tape called the algebraic tape*;*

(2) Whenever $\mathcal{A}$ sends (backdoor, m) to some protocol machine, where m contains some $\mathbf{X} \in \mathbb{G}$, then either (1) $\mathcal{A}$ also writes an algebraic representation *of $\mathbf{X}$ on its algebraic tape, or (2) $\mathcal{A}$ has previously received such representation from $\mathcal{E}$; where the algebraic representation of $\mathbf{X}$ is a list $\Lambda = [(\mathbf{X}_1, \lambda_1), \ldots, (\mathbf{X}_k, \lambda_k)]$ (where $\mathbf{X}_1, \ldots, \mathbf{X}_k \in \mathbb{G}$ and $\lambda_1, \ldots, \lambda_k \in \mathbb{Z}_p$) such that $\mathbf{X} = \mathbf{X}_1^{\lambda_1} \cdots \mathbf{X}_k^{\lambda_k}$, and $\mathbf{X}_1, \ldots, \mathbf{X}_k$ is the ordered list of group elements in messages $\mathcal{E}$ and/or $\mathcal{A}$ has received up to that point in the execution of π.*

We stress that it is necessary to separate the algebraic tape from the other tapes of $\mathcal{A}$ so that, for example, the message m itself does not contain an algebraic representation of $\mathbf{X}$. When clear from the context, we will drop $\mathcal{G}$ and π, and simply say that the environment/adversary is "algebraic."

We note that when considering static corruptions, the adversary runs the corrupt parties internally and hence messages produced by corrupt parties are subject to the restrictions above. The model for adaptive corruptions is the obvious one. Non-corrupt parties compute group elements honestly. So, if no secure erasure is assumed, the representations of any group elements computed by non-corrupt parties are part of their state when they are corrupted (and are given to the adversary). If we assume secure erasure, then any such state will not be available, and so any group elements that are part of a non-corrupt party's state will not have their representations available; in this case they must be added to the adversary's basis.

AGM emulation. We could now consider standard UC emulation restricted to algebraic adversaries and environments. However, looking ahead, in order for composition to hold we will want the simulator to be algebraic as well.

Definition 2. *Suppose protocols π and ϕ involve the same group $\mathcal{G}$. We say that π $\mathcal{G}$-AGM emulates ϕ if the following holds: for any efficient adversary $\mathcal{A}$, there is an efficient adversary $\mathcal{S}$ (called the* simulator*) such that: for any efficient $\mathcal{E}$ such that $(\mathcal{E}, \mathcal{A})$ are $(\mathcal{G}, \pi)$-algebraic, we have that $(\mathcal{E}, \mathcal{S})$ are $(\mathcal{G}, \phi)$-algebraic, and*

$$\mathrm{EXEC}_{\phi, \mathcal{S}, \mathcal{E}} \approx \mathrm{EXEC}_{\pi, \mathcal{A}, \mathcal{E}},$$

where $\mathrm{EXEC}_{\pi, \mathcal{A}, \mathcal{E}}$ denotes environment $\mathcal{E}$'s view in π's execution with adversary $\mathcal{A}$.

Above, we write $\approx$ to denote generic computational indistinguishability. This may refer to either asymptotic indistinguishability, in which case a security parameter is introduced as well, or concrete indistinguishability, in which case we write $\approx_\epsilon$ to denote that the distinguishing advantage is bounded by ϵ.

Definition 3. *Protocol π $\mathcal{G}$-AGM realizes ideal functionality $\mathcal{F}$ if π $\mathcal{G}$-AGM emulates $\mathrm{IDEAL}_\mathcal{F}$, the ideal protocol for $\mathcal{F}$.*

AGM emulation with respect to a sub-protocol. Our definitions of algebraic adversary and environment can be easily extended to the setting where the adversary/environment is restricted within a sub-protocol, namely it can only use group elements in received from *parties in this sub-protocol* as its basis for algebraic representation.

---

[3] Formally, we assume an encoding of group elements that distinguishes them from arbitrary strings. This can be done by simply prefixing any group element with a 0 and any other string (not necessarily representing a group element) with a 1. Following prior work [16], we use bold capital letters to denote group elements (except for the generator g).

[4] Formally, we consider protocols having access to a $\mathcal{F}_{\mathrm{CRS}}$ functionality, where $\mathcal{F}_{\mathrm{CRS}}$ runs a group-generation algorithm to obtain $\mathcal{G}$ (and possibly additional group elements), and then sends $\mathcal{G}$ (and any other elements) to parties that request it. Note that the protocol may use other groups, but we only require the adversary to be algebraic with respect to $\mathcal{G}$.

Definition 4. *Suppose protocol ρ^π involves group $\mathcal{G}$, and π is a sub-protocol of ρ^π. A pair of environment $\mathcal{E}$ and adversary $\mathcal{A}$ (in ρ^π's execution) is $(\mathcal{G}, \pi)$-algebraic if it satisfies the following:*

    (1) $\mathcal{A}$ has a special output tape called the algebraic tape*;*

    (2) Whenever $\mathcal{A}$ sends $(\mathsf{backdoor}, m)$ to some protocol machine, where m contains some $\mathbf{X} \in \mathbb{G}$, then either (1) $\mathcal{A}$ also writes an algebraic representation *(with respect to π) of $\mathbf{X}$ on its algebraic tape, or (2) $\mathcal{A}$ has previously received such representation from $\mathcal{E}$; where the algebraic representation of $\mathbf{X}$ is a list $\Lambda = [(\mathbf{X}_1, \lambda_1), \ldots, (\mathbf{X}_k, \lambda_k)]$ (where $\mathbf{X}_1, \ldots, \mathbf{X}_k \in \mathbb{G}$ and $\lambda_1, \ldots, \lambda_k \in \mathbb{Z}_p$) such that $\mathbf{X} = \mathbf{X}_1^{\lambda_1} \cdots \mathbf{X}_k^{\lambda_k}$, and $\mathbf{X}_1, \ldots, \mathbf{X}_k$ is the ordered list of group elements in messages $\mathcal{E}$ and/or $\mathcal{A}$ have received up to that point from either the environment or protocol machines in π, that is, excluding protocol machines in $\rho^\pi \setminus \pi$. (For the formal definition of a "sub-protocol," see Appendix B.1 .)*

Clearly, Definition 1 can be viewed as Definition 4 in the special case that $\rho^\pi = \pi$. Note that now we can talk about AGM emulation *with respect to a sub-protocol*, i.e., protocol ρ^π $(\mathcal{G}, \pi, \phi)$-AGM emulates ρ^ϕ, where the environment/adversary pair is restricted by the sub-protocol π, and the environment/ simulator pair is restricted by the sub-protocol ϕ. The formal definition exactly follows Definition 2.

The algebraically dummy adversary. Similar to the standard UC framework, we can also define a notion of dummy adversary here; this will be usful in our protocol analyses in the later sections. Recall that in the standard UC framework, the dummy adversary is one that merely passes messages to and from the environment. However, in our setting, the environment might send some algebraic representations to the adversary, which we do not want the protocol parties to receive. Hence, we define the *algebraically dummy adversary* as dropping these algebraic representations.

Definition 5. *Suppose protocol π involves group $\mathcal{G}$. An adversary $\mathcal{D}$ (in π's execution) is $(\mathcal{G}, \pi)$-algebraically dummy if it satisfies the following: for any message $(\mathsf{backdoor}, m)$ sent from some identity $ID \neq 0$ (i.e., from some protocol machine), it sends $(\mathsf{backdoor}, (ID, m))$ to the environment $\mathcal{E}$; for any message $(\mathsf{input}, (ID, m))$ sent from $\mathcal{E}$, it sends $(\mathsf{backdoor}, m)$ to identity ID, except that if m contains $\mathbf{X} \in \mathbb{G}$ and its algebraic representation Λ, then $\mathcal{A}$ sends $(\mathsf{backdoor}, m')$ to identity ID instead, where m' is m with Λ deleted.*

Since $\mathcal{D}$ does not write anything on its algebraic tape, for $(\mathcal{E}, \mathcal{D})$ to be algebraic, $\mathcal{E}$ must send all necessary algebraic representations to $\mathcal{D}$. To simplify notations, we may say "$\mathcal{E}$ is algebraic" in this case.

Now we can define AGM emulation with respect to the dummy adversary:

Definition 6. *Suppose protocols π and ϕ involve the same group $\mathcal{G}$. π $\mathcal{G}$-AGM emulates ϕ with respect to the dummy adversary if the following holds: there is an efficient simulator $\mathcal{S}$ such that: for any efficient and $(\mathcal{G}, \pi)$-algebraic environment $\mathcal{E}$, we have that $(\mathcal{E}, \mathcal{S})$ are $(\mathcal{G}, \phi)$-algebraic, and*

$$\mathrm{EXEC}_{\phi, \mathcal{S}, \mathcal{E}} \approx \mathrm{EXEC}_{\pi, \mathcal{D}, \mathcal{E}},$$

where $\mathcal{D}$ is the $(\mathcal{G}, \pi)$-algebraically dummy adversary.

Similar to the standard UC framework, we can show that emulation is equivalent to emulation with respect to the dummy adversary. This simplifies protocol analysis, since from now on we can simply assume that the adversary is algebraically dummy.

Theorem 1. *Suppose protocols π and ϕ involve the same group $\mathcal{G}$. Then π $\mathcal{G}$-AGM emulates ϕ (as in Definition 2) iff π $\mathcal{G}$-AGM emulates ϕ with respect to the dummy adversary (as in Definition 6).*

The proof is tedious and is therefore deferred to Appendix B.3.

2.3 Composition in the UC-AGM

The composition theorem. We are now ready to prove the composition theorem in our UC-AGM framework:

Theorem 2. *Suppose protocols π and ϕ involve the same group $\mathcal{G}$, such that ϕ is a sub-protocol of ρ^ϕ, π $\mathcal{G}$-AGM emulates ϕ, and π is identity-compatible with ρ^ϕ and ϕ. Then ρ^π (ρ^ϕ with its sub-protocol ϕ replaced with π) $(\mathcal{G}, \pi, \phi)$-AGM emulates ρ^ϕ. (For formal definitions of "identity-compatibility" and "sub-protocol replacement", see Appendix B.1)*

Proof. Let $\mathcal{D}_\pi$ be the algebraically dummy adversary in an execution of π. Since π $\mathcal{G}$-AGM emulates ϕ, we know that there is an efficient simulator $\mathcal{S}_\pi$ such that: for any efficient and $(\mathcal{G}, \pi)$-algebraic environment $\mathcal{E}_\pi$, we have that $(\mathcal{E}_\pi, \mathcal{S}_\pi)$ are $(\mathcal{G}, \phi)$-algebraic, and

$$\text{EXEC}_{\phi, \mathcal{S}_\pi, \mathcal{E}_\pi} \approx \text{EXEC}_{\pi, \mathcal{D}_\pi, \mathcal{E}_\pi}.$$

Let $\rho = \rho^\phi \setminus \phi$, i.e., ρ is the "caller" part of ρ^π.

Construction of simulator $\mathcal{S}$. By Theorem 1, it suffices to consider the $(\mathcal{G}, \pi)$-algebraically dummy adversary $\mathcal{D}$ in an execution of ρ^π. We construct a simulator $\mathcal{S}$ (in an execution of ρ^ϕ) which simulates $\mathcal{E}$'s view for any efficient and $(\mathcal{G}, \pi)$-algebraic environment $\mathcal{E}$. $\mathcal{S}$ essentially "combines" $\mathcal{D}$ and $\mathcal{S}_\pi$. Concretely, $\mathcal{S}$ works as follows:

1. On message $(\mathsf{input}, (ID, m_0))$ from identity 0 (recall that this means that $\mathcal{E}$ instructs $\mathcal{S}$ to send message m_0 to the protocol party with identity ID), $\mathcal{S}$ checks if there is a machine in ϕ with identity ID.
   (a) If so, then $\mathcal{S}$ activates $\mathcal{S}_\pi$ with input $(\mathsf{input}, (ID, m_0))$ (as from the environment), and follows $\mathcal{S}_\pi$'s instruction until the activation of $\mathcal{S}_\pi$ completes.
   (b) Otherwise, i.e., ID is the identity of a machine in ρ, $\mathcal{S}$ parses $m_0 = (m_0', \Lambda)$ (where Λ is the algebraic representations of the group elements in m_0') and sends $(\mathsf{backdoor}, m_0')$ to ID, and writes Λ on its algebraic tape.
2. On message $(\mathsf{backdoor}, m_1)$ from some identity $ID \neq 0$ (i.e., from a protocol party), $\mathcal{S}$ checks if there is a machine in ϕ with identity ID.
   (a) If so, then $\mathcal{S}$ activates $\mathcal{S}_\pi$ with input $(\mathsf{backdoor}, m_1)$ (as from ID), and follows $\mathcal{S}_\pi$'s instruction until the activation of $\mathcal{S}_\pi$ completes.
   (b) Otherwise, i.e., ID is the identity of a machine in ρ, $\mathcal{S}$ sends $(\mathsf{backdoor}, m_1)$ to identity 0 (i.e., to $\mathcal{E}$).

Analysis of simulator $\mathcal{S}$. It is straightforward to see that if $\mathcal{S}_\pi$ is efficient, then $\mathcal{S}$ is also efficient. We now show that $(\mathcal{E}, \mathcal{S})$ are $(\mathcal{G}, \phi)$-algebraic. Recall that $(\mathcal{E}, \mathcal{S})$ are $(\mathcal{G}, \phi)$-algebraic iff whenever $\mathcal{S}$ sends $(\mathsf{backdoor}, m)$ to identity $ID \neq 1$, it also writes on its algebraic tape the algebraic representations (w.r.t. ϕ) of all group elements in m. According to the description of $\mathcal{S}$ above, $\mathcal{S}$ sends $\mathsf{backdoor}$ messages to identity $ID \neq 1$ in step 1(b) only; in this case $\mathcal{S}$ writes the algebraic representation Λ on its algebraic tape, so $\mathcal{E}$ is $(\mathcal{G}, \pi)$-algebraic implies that $(\mathcal{E}, \mathcal{S})$ are $(\mathcal{G}, \phi)$-algebraic.

Moreover, $\mathcal{S}$ plays the role of an $(\mathcal{G}, \pi)$-*algebraic* environment when activating $\mathcal{S}_\pi$ with message $(\mathsf{input}, (ID, m_0))$. This is because $\mathcal{S}$ copies $\mathcal{E}$'s message payload m_0, so $\mathcal{E}$ is $(\mathcal{G}, \pi)$-algebraic implies that m_0 contains the algebraic representations (w.r.t. π) of its all group elements.

Next we show the validity of $\mathcal{S}$. We construct another environment $\mathcal{E}_\pi$, which aims to distinguish between π's execution with $\mathcal{D}$ and ϕ's execution with $\mathcal{S}_\pi$. $\mathcal{E}_\pi$ simulates instances of $\mathcal{E}$ and runs the codes of ρ and $\mathcal{S}$ locally, and essentially "combines" $\mathcal{E}$, ρ, and $\mathcal{S}$. Concretely, $\mathcal{E}_\pi$, on initial input z, activates $\mathcal{E}$ with initial input z. Then $\mathcal{E}_\pi$ works as follows:

1. When $\mathcal{E}$ completes this activation,
   (a) If $\mathcal{E}$ halts with some output, then $\mathcal{E}_\pi$ also halts with the same output.
   (b) If $\mathcal{E}$ generates an outgoing message (input, m_0) to some identity ID such that there is a machine $\mu \in \rho$ with identity ID, then $\mathcal{E}_\pi$ runs the code of μ on message (input, m_0). When μ halts, $(*)$
      i. If μ generates an outgoing message $(\mathsf{subroutine\text{-}output}, m_1)$ to identity 0, then $\mathcal{E}_\pi$ activates $\mathcal{E}$ with message $(\mathsf{subroutine\text{-}output}, m_1)$ (as from ID) and jumps to the beginning of this step.
      ii. If μ generates an outgoing message $(\mathsf{backdoor}, m_1)$ to identity 1, then $\mathcal{E}_\pi$ runs the code of $\mathcal{S}$ on message $(\mathsf{backdoor}, m_1)$.

9

iii. If μ generates an outgoing message (input, m_1) to identity ID', which is the identity of a machine $\mu' \in \rho$, then $\mathcal{E}_\pi$ runs the code of μ' on input (input, m_1) and jumps to $(*)$ (with μ replaced by μ').

iv. If μ generates an outgoing message (input, m_1) to identity ID', which is the identity of a machine in ϕ/π, then $\mathcal{E}_\pi$ sends (input, m_1) to identity ID'.

(c) If $\mathcal{E}$ generates an outgoing message $(\mathsf{input}, (ID, m_0))$ to identity 1, then $\mathcal{E}_\pi$ runs the code of $\mathcal{S}$ on message $(\mathsf{input}, (ID, m_0))$.

2. When $\mathcal{S}$ halts (as in case (b)ii or (c) in step 1; recall that $\mathcal{S}$ is a piece of code run by $\mathcal{E}$ itself),

(a) If $\mathcal{S}$ generates an outgoing message $(\mathsf{backdoor}, m_2)$ to identity 0, then $\mathcal{E}_\pi$ activates $\mathcal{E}$ with message $(\mathsf{backdoor}, m_2)$ and jumps to step 1.

(b) If $\mathcal{S}$ generates an outgoing message $(\mathsf{backdoor}, m_2)$ to identity ID, which is the identity of a machine $\mu \in \rho$, then $\mathcal{E}_\pi$ runs the code of μ on message (input, m_2) and jumps to $(*)$.

(c) If $\mathcal{S}$ activates $\mathcal{S}_\pi$[5] with message $(\mathsf{input}, (ID, m_2))$, then $\mathcal{E}_\pi$ sends $(\mathsf{input}, (ID, m_2))$ to identity 1 (i.e., to $\mathcal{D}_\pi$ or $\mathcal{S}_\pi$).

3. On message $(\mathsf{backdoor}, (ID, m_3))$ from identity 1, $\mathcal{E}_\pi$ runs the code of $\mathcal{S}$ on message $(\mathsf{backdoor}, (ID, m_3))$ (as from $\mathcal{S}_\pi$) and jumps to step 2.

4. On message $(\mathsf{backdoor}, m_3)$ from some identity $ID \neq 1$ (i.e., from a machine in ϕ or π) aimed at some identity ID',

(a) If there is a machine $\mu' \in \rho$ with identity ID', then $\mathcal{E}_\pi$ runs the code of μ' on message (input, m_3) and jumps to $(*)$ (with μ replaced by μ').

(b) Otherwise, i.e., if ID' is an external identity, then $\mathcal{E}_\pi$ activates $\mathcal{E}$ with message $(\mathsf{backdoor}, m_3)$ (as from ID) and jumps to step 1.

It is straightforward to see that if $\mathcal{E}$ is efficient, then $\mathcal{E}_\pi$ is also efficient. Also, $\mathcal{E}_\pi$ perfectly simulates an instance of $\mathcal{D}$ in π's execution, and an instance of $\mathcal{S}_\pi$ in ϕ's execution, i.e.,

$$\mathrm{EXEC}_{\pi, \mathcal{D}_\pi, \mathcal{E}_\pi} = \mathrm{EXEC}_{\rho^\pi, \mathcal{D}, \mathcal{E}}, \quad \text{and} \quad \mathrm{EXEC}_{\phi, \mathcal{S}_\pi, \mathcal{E}_\pi} = \mathrm{EXEC}_{\rho^\phi, \mathcal{S}, \mathcal{E}}.$$

Next we claim that if $\mathcal{E}$ is $(\mathcal{G}, \pi)$-algebraic, then $\mathcal{E}_\pi$, as the environment in an execution of π, is also $(\mathcal{G}, \pi)$-algebraic. Recall that $\mathcal{E}_\pi$ is $(\mathcal{G}, \pi)$-algebraic iff whenever it sends (input, m) to identity 1, m contains the algebraic representations (w.r.t. π) of its all group elements. According to the description of $\mathcal{E}_\pi$ above, $\mathcal{E}_\pi$ sends input messages to identity 1 in step 2(c) only. The message payload m_2 is copied from $\mathcal{S}$'s message aimed at $\mathcal{S}_\pi$; we have argued above that $\mathcal{S}$ plays the role of a $(\mathcal{G}, \pi)$-algebraic environment while communicating with $\mathcal{S}_\pi$, which implies that m_2 contains the algebraic representations (w.r.t. π) of its all group elements.

Since $\mathcal{E}_\pi$ is both efficient and $(\mathcal{G}, \pi)$-algebraic, by the definition of $\mathcal{S}_\pi$, we have that

$$\mathrm{EXEC}_{\phi, \mathcal{S}_\pi, \mathcal{E}_\pi} \approx \mathrm{EXEC}_{\pi, \mathcal{D}_\pi, \mathcal{E}_\pi}.$$

Combining the three results above, we conclude that

$$\mathrm{EXEC}_{\rho^\phi, \mathcal{S}, \mathcal{E}} \approx \mathrm{EXEC}_{\rho^\pi, \mathcal{D}, \mathcal{E}},$$

completing the proof. $\qquad\square$

Transitivity of AGM-emulation. The following theorem is straightforward to prove, similarly to the standard UC framework.

Theorem 3. *Suppose protocols π, π', ϕ involve the same group $\mathcal{G}$, such that π $\mathcal{G}$-AGM emulates π' and π' $\mathcal{G}$-AGM emulates ϕ. Then π $\mathcal{G}$-AGM emulates ϕ.*

---

[5] Note that this $\mathcal{S}_\pi$ is an imaginary machine supposed to run inside $\mathcal{S}$, whereas the "actual" $\mathcal{S}_\pi$ is the simulator in the execution of ϕ. Same with step 3 below.

Proof. Our goal is to give a simulator $\mathcal{S}$ such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{E}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ when $(\mathcal{E},\mathcal{A})$ are $(\mathcal{G},\pi)$-algebraic. Furthermore, $(\mathcal{E},\mathcal{S})$ must be $(\mathcal{G},\phi)$-algebraic.

By assumption, since π AGM-emulates π', there is an efficient algebraic adversary $\mathcal{A}'$ such that $\text{EXEC}_{\pi',\mathcal{A}',\mathcal{E}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ when $(\mathcal{E},\mathcal{A})$ are $(\mathcal{G},\pi)$ algebraic. Furthermore, $(\mathcal{E},\mathcal{A}')$ are $(\mathcal{G},\pi')$-algebraic.

Moreover, since π' AGM-emulates ϕ, there is an efficient algebraic adversary $\mathcal{S}$ such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{E}} \approx \text{EXEC}_{\pi',\mathcal{A}',\mathcal{E}}$ when $(\mathcal{E},\mathcal{A}')$ are $(\mathcal{G},\pi')$-algebraic. Furthermore, $(\mathcal{E},\mathcal{S})$ are $(\mathcal{G},\phi)$-algebraic. This implies that $\mathcal{S}$ is the required simulator, which concludes the proof. $\square$

In the standard UC framework, the guarantees given by the UC composition theorem can be plugged in as hypothesis of the transitivity theorem, which allows deriving a natural corollary when ϕ is an ideal functionality. Intuitively, in the standard UC setting, composition allows us to derive that ρ^π emulates $\rho^{\mathcal{F}}$, when π emulates $\mathcal{F}$. If, in turn $\rho^{\mathcal{F}}$ has been shown to emulate $\mathcal{F}'$, then transitivity yields that ρ^π emulates $\mathcal{F}'$.

However, this is not the case in the UC AGM setting. The composition theorem guarantees that ρ^π emulates $\rho^{\mathcal{F}}$ with respect to $(\mathcal{G},\pi)$-algebraic attackers, rather than $(\mathcal{G},\rho^\pi)$-algebraic attackers. This means that, in order to plug-in composition results with transitivity to obtain a result for ideal functionality emulation, we require a refined theorem that considers the specific case of composed protocols.

Theorem 4. *Suppose protocols $\rho^{\mathcal{F}}$, π and ideal functionalities $\mathcal{F}$, $\mathcal{F}'$ involve the same group $\mathcal{G}$, such that:*

1. $\text{IDEAL}_{\mathcal{F}}$ *is a sub-protocol of $\rho^{\mathcal{F}}$,*
2. *the π protocol $(\mathcal{G},\pi)$-AGM realizes $\mathcal{F}$,*
3. *the $\rho^{\mathcal{F}}$ protocol $(\mathcal{G},\rho)$-AGM realizes $\mathcal{F}'$, and*
4. *π is identity-compatible with $\rho^{\mathcal{F}}$ and $\text{IDEAL}_{\mathcal{F}}$.*

Then the instantiated protocol ρ^π AGM realizes $\mathcal{F}'$ *with respect to attackers that are* both $(\mathcal{G},\rho)$- and $(\mathcal{G},\pi)$-algebraic.

Proof (Sketch). To prove this statement we need to recall the structure of the simulator for ρ^π that is implied by the composition theorem; here we will call it $\mathcal{A}'$ consistently with the transitivity theorem proof.

This simulator runs $\mathcal{A}$ internally and, when $\mathcal{A}$ communicates with machines executing π, it uses the simulator $\mathcal{S}_\pi$ as a *translator* that communicates to $\mathcal{F}$ instead. On the other hand, communications between $\mathcal{A}$ and parties executing ρ are just passed along.

Note that, to use this simulator we need to apply the composition theorem, which means that $(\mathcal{E},\mathcal{A})$ must be $(\mathcal{G},\pi)$ algebraic; this is guaranteed by the stronger restriction that attackers are both $(\mathcal{G},\rho)$ *and* $(\mathcal{G},\pi)$ algebraic.

At this point we can now follow the same strategy adopted in the proof of the transitivity theorem: simulator $\mathcal{A}'$ is used as an attacker against $\rho^{\mathcal{F}}$. The crucial observation now is that, this simulator guarantees that, if $(\mathcal{E},\mathcal{A})$ are $(\mathcal{G},\pi)$ algebraic *and* $(\mathcal{G},\rho)$ algebraic, then $(\mathcal{E},\mathcal{A}')$ is also $(\mathcal{G},\rho)$ algebraic. This is because communications with ρ are just passed along between $\mathcal{A}$ and ρ.

We can now apply the hypothesis that the $\rho^{\mathcal{F}}$ protocol $(\mathcal{G},\rho)$-AGM realizes $\mathcal{F}'$ and take simulator $\mathcal{S}$ implied by this hypothesis to conclude the proof. $\square$

Extension to the full UC framework and relation to UC proofs. In Appendix B.2 we explain how our treatment here can be extended to the full UC framework, which models fully dynamic and evolving distributed computing systems.

UC emulation implies AGM emulation. For completeness, we note that UC emulation implies AGM emulation whenever the algebraic restriction on the simulator is moot. To see this, fix protocols π, ϕ where π UC emulates ϕ and ϕ does not impose any algebraic restriction on $\mathcal{S}$. Any efficient algebraic environment $\mathcal{E}$ is in particular an efficient environment, so there is an efficient simulator $\mathcal{S}$ for which $\text{EXEC}_{\phi,\mathcal{S},\mathcal{E}} \approx \text{EXEC}_{\pi,\mathcal{D},\mathcal{E}}$ holds for any efficient algebraic environment $\mathcal{E}$. Furthermore, $\mathcal{S}$ is trivially algebraic since there is no such requirement when interacting with ϕ.

In Appendix E we discuss in detail how UC AGM proofs compose with stronger standard UC emulation results, and further clarify the implications of the UC AGM composition theorems. The discussion also clarifies what happens in the setting where different groups are used by different protocols.

We finally note that the fact that we refer to protocols that use the same group in our theorems because this is the more problematic case, and it serves to highlight the limitations to composition in the AGM. All our results carry without change to the case where different groups are used; in this case excluding attacks that prevent using group elements occurring in one protocol in an attack against another protocol, unless a representation can be provided, seems less of a limitation.

3 Analysis of the Chou-Orlandi Protocol

In this section, we analyze the security of the Chou-Orlandi protocol for oblivious transfer in the UC-AGM. For convenience, we present the standard OT functionality $\mathcal{F}_{\mathrm{OT}}$ in Figure 3. We describe the Chou-Orlandi protocol Π_{CO} in Figure 4. All messages sent in the protocol are via a message authentication functionality $\mathcal{F}_{\mathrm{AUTH}}$, as presented in [10].

Functionality $\mathcal{F}_{\mathrm{OT}}$

- On input (receive, sid, b) where $b \in \{0,1\}$ from R do: if no message of the form (receive, sid, b), $b \in \{0,1\}$ has been stored, store (receive, sid, b). Output (receive, sid) to $\mathcal{S}$.
- On input (send, sid, m_0, m_1) where $m_0, m_1 \in \{0,1\}^\ell$ from S do: if no message of the form (send, sid, m_0, m_1) has been stored, store (send, sid, m_0, m_1). Output (send, sid) to $\mathcal{S}$.

Adversary $\mathcal{S}$:

- On input (receive, sid, b) where $b \in \{0,1\}$ from $\mathcal{S}$ do: if no message of the form (receive, sid, b), $b \in \{0,1\}$ has been stored, store (receive, sid, b).
- On input (send, sid, m_0, m_1) where $m_0, m_1 \in \{0,1\}^\ell$ from $\mathcal{S}$ do: if no message of the form (send, sid, m_0, m_1) has been stored, store (send, sid, m_0, m_1).
- On input (deliver, sid, R) from $\mathcal{S}$: if both (receive, sid, b) and (send, sid, m_0, m_1) have previously been stored, do:
    - If R is honest: output (output, sid, m_b) to R.
    - If R is corrupted: output (output, sid, m_b) to $\mathcal{S}$.
  Otherwise, output $\perp$ to $\mathcal{S}$.
- On input (deliver, sid, S) from $\mathcal{S}$, if (output, sid, m_b) was previously output (to R or to $\mathcal{S}$) and S is honest, output (output, sid) to S. Otherwise, output $\perp$ to $\mathcal{S}$.

Fig. 3. Functionality for 1-out-of-2 OT

Protocol Π_{CO}

- **Step 1:** Upon receiving input (send, sid, m_0, m_1), S samples $y \leftarrow \mathbb{Z}_p$ and computes $\mathbf{A} := g^y, \mathbf{B} := g^{y^2}$. It sends $\mathbf{A}$ to R via $\mathcal{F}_{\mathrm{AUTH}}$.
- **Step 2:** Upon receiving input (receive, sid, b) and $\mathbf{A} \in \mathbb{G}$ from S, R samples $x \leftarrow \mathbb{Z}_p$ and computes $\mathbf{U} := \mathbf{A}^b g^x$. It sends $\mathbf{U}$ to S via $\mathcal{F}_{\mathrm{AUTH}}$.
- **Step 3:** Upon receiving $\mathbf{U}$ from R, S computes $k_b := H(\mathbf{A}, \mathbf{U}^y \mathbf{B}^{-b-1}), e_b := m_b \oplus k_b$ for $b \in \{0,1\}$. It sends e_0, e_1 to R via $\mathcal{F}_{\mathrm{AUTH}}$ and outputs (output, sid).
- **Step 4:** Upon receiving e_0, e_1 from S, R computes $k_{b^*} := H(\mathbf{A}, \mathbf{A}^x), m_{b^*} := e_{b^*} \oplus k_{b^*}$ for $b^* \in \{0,1\}$. It outputs (output, sid, m_{b^*}).
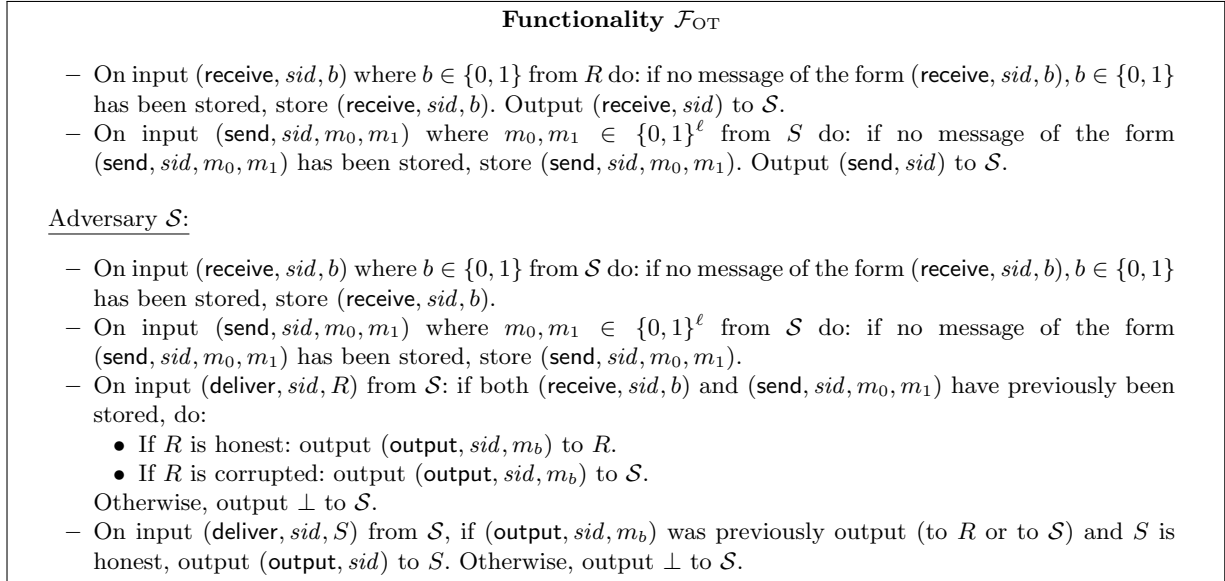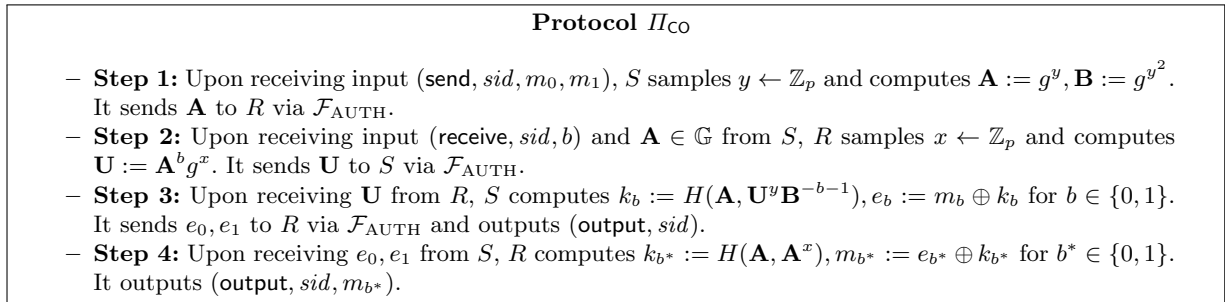

Fig. 4. The Chou-Orlandi OT protocol.

We now turn toward proving security of the protocol. In the following, we denote S and R as the sender and the receiver in protocol Π_{CO}, respectively. We describe a simulator $\mathsf{Sim}_{\mathsf{CO}}$ for Π_{CO} by considering the different options for the order of corruptions. We assume that the simulator immediately aborts if it obtains syntactically ill-formed messages from a corrupted party as part of Π_{CO}. We first give an outline of the proof.

Proof intuition. Roughly speaking, our proof must overcome two main challenges from the original work of Chou and Orlandi. The first is how to simulate the internal state of parties upon adaptive corruption. Namely, in Chou and Orlandi's proof, there seems to be no way of of explaining the secret exponent x chosen by R if S is statically corrupted and can send an arbitrary group element $\mathbf{A}$ in Step 1 for which R does not know the discrete logarithm y. This issue is easily resolved using the AGM, as the simulator always learns the exponent y from the algebraic coefficients provided for $\mathbf{A}$.

The second issue in their proof comes from an improperly defined $\mathcal{F}_{\mathsf{OT}}$ functionality. Roughly speaking, their version of this functionality does not notify S upon R obtaining the message m_b. If the corrupted R never makes the query for one of the keys k_b to H then the simulator cannot extract the correct bit and complete the simulation of the protocol. Note that this issue cannot be overcome by the simulator naively completing the simulation before R makes this query by prompting the message (output, sid) from $\mathcal{F}_{\mathsf{OT}}$ to S prematurely via a query on some arbitrary b. The reason is that $\mathcal{E}$ can always make the opposite query to H, i.e., for k_{1-b}, with probability 1 after the simulation is complete. In this case, there is no way to obtain m_{1-b} from $\mathcal{F}_{\mathsf{OT}}$ again, since $\mathcal{S}$ already had to make the query in order to force (output, sid) being output to $\mathcal{E}$. Both of these issues can be overcome when requiring that the environment $\mathcal{E}$ be algebraic. In this case, y is revealed when the corrupted S sends it in Step 1. For the issue of extraction, $\mathcal{S}$ observes that R either sends $\mathbf{U} := \mathbf{A}^b g^x$ as specified by the protocol, or sends $\mathbf{U} := g^x$ that does not satisfy this format. In the either case, $\mathcal{S}$ can safely carry out the extraction according to either b or an arbitrary bit (in case $\mathbf{U}$ is not of the specified format). The only way for $\mathcal{E}$ to distinguish the simulation from the real world is by making a query from which a discrete logarithm instance can be solved (using algebraic coefficients provided by $\mathcal{E}$ as part of that query to H).

Let g denote a generator for a cyclic group $\mathbb{G}$ of prime order q and let DL denote the problem of computing a when given a random element $\mathbf{A} = g^a$ in $\mathbb{G}$. Moreover, denote $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{DL}} := \Pr[a' = a \mid a' \leftarrow \mathcal{B}(g^a)]$ the advantage of adversary $\mathcal{B}$ in solving DL. Then Theorem 5 shows that the Π_{CO} protocol for oblivious transfer AGM realizes $\mathcal{F}_{\mathsf{OT}}$.

Theorem 5. *Π_{CO} UC-realizes $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model under adaptive corruptions. More precisely, there exists an algebraic simulator $\mathsf{Sim}_{\mathsf{CO}}$ for the algebraically dummy adversary $\mathcal{D}$ such that, for every algebraic environment $\mathcal{E}$ that makes at most q_H queries to $\mathcal{F}_{\mathsf{RO}}$, there exist attackers $\mathcal{B}_1$ and $\mathcal{B}_2$ running in roughly the same time as $\mathcal{E}$, such that $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{OT}},\mathcal{S},\mathcal{E}} \approx_\epsilon \mathrm{EXEC}_{\Pi_{\mathsf{CO}},\mathcal{D},\mathcal{E}}$, with*

$$\epsilon \leq q_H \cdot (\mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{DL}} + \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{DL}}).$$

Proof. The simulator $\mathsf{Sim}_{\mathsf{CO}}$ is as follows:

S is corrupted before Step 1.

- **R is corrupted before Step 2.** In this case, there is nothing for $\mathsf{Sim}_{\mathsf{CO}}$ to simulate.
- **R is corrupted between Step 2 and 4.** In this case, R has received $\mathbf{A} \in \mathbb{G}$ from S (but has not yet received e_0, e_1). In addition, $\mathsf{Sim}_{\mathsf{CO}}$ learns $y \in \mathbb{Z}_p$ s.t. $\mathbf{A} = g^y$. $\mathsf{Sim}_{\mathsf{CO}}$ samples $u \leftarrow \mathbb{Z}_p$ and computes $\mathbf{U} := g^u$, which it sends to S. When R becomes corrupted, $\mathsf{Sim}_{\mathsf{CO}}$ learns b and sets sets $x := u - yb$. It outputs (b, x) to $\mathcal{E}$. In addition, it simulates the random oracle H as described in the next subcase.
- **R is corrupted after Step 4.** In this case, R receives $\mathbf{A} \in \mathbb{G}$ at Step 2 and e_0, e_1 at Step 4. In addition, $\mathsf{Sim}_{\mathsf{CO}}$ learns $y \in \mathbb{Z}_p$ s.t. $\mathbf{A} = g^y$. $\mathsf{Sim}_{\mathsf{CO}}$ samples $u \leftarrow \mathbb{Z}_p$ and computes $\mathbf{U} := g^u$, which it sends to S. To program H, when $\mathcal{E}$ queries H on input $(\mathbf{I}, \mathbf{J})$ (together with the algebraic representations of $\mathbf{I}, \mathbf{J}$), $\mathsf{Sim}_{\mathsf{CO}}$ does as follows.
  - It first checks whether $H[\mathbf{I}, \mathbf{J}] \neq \perp$. In this case, it returns $H[\mathbf{I}, \mathbf{J}]$. Thus, assume in the following that $\mathcal{E}$ queries H on some input for the first time. In addition, for any fresh query $\mathbf{I}, \mathbf{J}$, assume that $\mathsf{Sim}_{\mathsf{CO}}$ sets $H[\mathbf{I}, \mathbf{J}]$ to the value it returns.

- If the query is of the form $H(g^y, \mathbf{U}^y g^{y^2(b-1)})$ for $b \in \{0,1\}$, $\mathsf{Sim_{CO}}$ sets $k_b \leftarrow \{0,1\}^\kappa$. It returns k_b.
- Otherwise $\mathsf{Sim_{CO}}$ samples $k \leftarrow \{0,1\}^\kappa$ and returns k.
- After observing both $\mathbf{A}$ and $\mathbf{U}$ in the protocol, $\mathsf{Sim_{CO}}$ also retroactively checks whether it has previously set $H[g^y, \mathbf{U}^y g^{y^2(b-1)}]$. If so, it sets $k_b := H[g^y, \mathbf{U}^y g^{y^2(b-1)}]$.

Upon having received e_0, e_1 from S, $\mathsf{Sim_{CO}}$ samples $m_0, m_1 \leftarrow \{0,1\}^\kappa$. For all $b \in \{0,1\}$ for which $k_b = \perp$ at this point, it sets $k_b \leftarrow \{0,1\}^\kappa$ and programs $H[g^y, \mathbf{U}^y g^{y^2(b-1)}] = k_b$ (it does not resample k_b in case it has already been defined). It then sets $m_0 := e_0 \oplus k_0, m_1 := e_1 \oplus k_1$ and inputs $(\mathsf{send}, sid, m_0, m_1)$ and $(\mathsf{deliver}, sid, R)$ to $\mathcal{F}_{\mathrm{OT}}$. This prompts the output $(\mathsf{output}, sid, m_b)$ to the honest R, since R has previously input $(\mathsf{receive}, sid, b)$ to $\mathcal{F}_{\mathrm{OT}}$. When R is corrupted, $\mathsf{Sim_{CO}}$ learns b and sets $x := u - yb \pmod q$. It outputs (b, x, m_b) to $\mathcal{E}$.

S is corrupted between Step 1 and Step 3. To simulate the behaviour of S, $\mathsf{Sim_{CO}}$ samples $y \leftarrow \mathbb{Z}_p$ and computes $\mathbf{A} := g^y, \mathbf{B} := g^{y^2}$. It sends $\mathbf{A}$ to R. When S is corrupted, $\mathsf{Sim_{CO}}$ learns m_0, m_1. It outputs (y, m_0, m_1) to $\mathcal{E}$.

- **R is corrupted before Step 2.** In this case, $\mathsf{Sim_{CO}}$ only needs to simulate H before S becomes corrupted (afterwards, both parties are corrupt and there is nothing to simulate). When R sends $\mathbf{U}$ in Step 2, $\mathsf{Sim_{CO}}$ learns u, v s.t. $\mathbf{U} = g^u \mathbf{A}^v$. $\mathsf{Sim_{CO}}$ now proceeds to simulate H exactly as in the case where S is corrupted before Step 1, except that it aborts if it ever sets both $k_0, k_1 \neq \perp$. As in the case where the Sender is corrupted before Step 1, $\mathsf{Sim_{CO}}$'s simulation is indeed efficient, since it knows y and can hence check the necessary relations in the exponent of $\mathbf{U}$.

  Claim. $\mathsf{Sim_{CO}}$ does not abort except with probability $\frac{1}{q_H}\mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}_1}$, where $\mathcal{B}_1$ is an adversary that runs in roughly the same time as $\mathcal{E}$.

  Proof. $\mathsf{Sim_{CO}}$ aborts in this case only if the adversary queries $g^y, \mathbf{U}^y g^{y^2)}$, as it queries both $g^y, \mathbf{U}^y g^{y^2(b-1)}$ for both $b = 0, b = 1$ to H by assumption. In this case, we can construct $\mathcal{B}_1$ as follows. On input a discrete logarithm challenge $\mathbf{A} = g^y$ in game DL, $\mathcal{B}_1$ samples $i \in [q_H]$ uniformly at random and runs $\mathcal{E}$. It simulates the behavior of $\mathsf{Sim_{CO}}$ by sending the element $\mathbf{A}$ in Step 1. If $\mathcal{E}$ corrupts R, $\mathcal{B}_1$ aborts. When $\mathcal{E}$ (controlling R) queries H on input $\mathbf{A}, \mathbf{J}$, $\mathsf{Sim_{CO}}$ learns coefficients a, b s.t. $\mathbf{J} = g^a \mathbf{A}^b$. If $\mathbf{J} = \mathbf{A}$, and $v = 0$, then $\mathcal{B}_1$ sets $k_b \leftarrow \{0,1\}^\kappa$ and programs $H[\mathbf{I}, \mathbf{J}] = k_b$. For the i-th such query for which $v \neq 0$, $\mathcal{B}$ solves the equation $(1 + v) \cdot y^2 + (u - b) \cdot y - a = 0 \pmod q$ for y, and outputs y. (Note that this yields the correct solution in case the i-th query is of the form $g^y, \mathbf{U}^y g^{y^2)}$). Since $\mathcal{B}_1$ guesses q correctly with probability at least $\frac{1}{q_H}$ and perfectly simulates the behavior of $\mathsf{Sim_{CO}}$ up that point perfectly, the claim follows. $\qquad\square$

- **R is corrupted between Steps 2 and 4.** In this case, the simulation for R can be carried out as in the case where S is corrupted before Step 1.
- **R is corrupted after Step 4.** In this case, R receives e_0, e_1 from the corrupted sender S at Step 4. The only difference to the case where S is corrupted before Step 1 is that that $\mathsf{Sim_{CO}}$ knows $y \in \mathbb{Z}_p$ s.t. $\mathbf{A} = g^y$ from sampling it in the first part of the simulation (rather than learning it from the algebraic coefficients output by the corrupted S).

S is corrupted after Step 3. To simulate the behaviour of S, in Step 1, $\mathsf{Sim_{CO}}$ samples $y \leftarrow \mathbb{Z}_p$ and computes $\mathbf{A} := g^y, \mathbf{B} := g^{y^2}$. It sends $\mathbf{A}$ to R.

- **R is corrupted before Step 2.** When R sends $\mathbf{U} \in \mathbb{G}$ to S, $\mathsf{Sim_{CO}}$ checks whether the algebraic coefficients provided by R are such that $\mathbf{U} = \mathbf{A}^b g^x$ for some $b \in \{0,1\}, x \in \mathbb{Z}_p$.
  - If so, $\mathsf{Sim_{CO}}$ inputs $(\mathsf{receive}, sid, b)$ and then $(\mathsf{deliver}, sid, R)$ to $\mathcal{F}_{\mathrm{OT}}$.
  - Otherwise, $\mathsf{Sim_{CO}}$ samples $b \leftarrow \{0,1\}, x \leftarrow \mathbb{Z}_p$ and inputs $(\mathsf{receive}, sid, b)$ and then $(\mathsf{deliver}, sid, R)$ to $\mathcal{F}_{\mathrm{OT}}$.

  Either case prompts the output $(\mathsf{output}, sid, m_b)$ to $\mathsf{Sim_{CO}}$, since S is honest at this point and has previously input $(\mathsf{send}, sid, m_0, m_1)$ to $\mathcal{F}_{\mathrm{OT}}$. To simulate H on input $(\mathbf{I}, \mathbf{J})$, $\mathsf{Sim_{CO}}$ does as follows.

14

- It first checks whether $H[\mathbf{I}, \mathbf{J}] \neq \perp$. In this case, it returns $H[\mathbf{I}, \mathbf{J}]$. Thus, assume in the following that H is queried on some input for the first time. In addition, for any fresh query $\mathbf{I}, \mathbf{J}$, assume that $\mathsf{Sim}_{\mathsf{CO}}$ sets $H[\mathbf{I}, \mathbf{J}]$ to the value it returns.
- For any query to H, $\mathsf{Sim}_{\mathsf{CO}}$ checks whether it is of the form $H(g^y, g^{y^2}, \mathbf{U}^y g^{y^2(b-1)})$ for $b \in \{0, 1\}$, (i.e., it checks with respect to both $b = 0$ *and* $b = 1$). If any queries have been made before $\mathbf{U}$ was set by the corrupted R, $\mathsf{Sim}_{\mathsf{CO}}$ also checks whether they have this format.
  * If not, it samples $k \leftarrow \{0, 1\}^\kappa$ and returns k.
  * Otherwise, if $\mathsf{Sim}_{\mathsf{CO}}$ has previously set $e_b \neq \perp$ (see below) it sets $k_b := e_b \oplus m_b$. Else, it sets $k_b \leftarrow \{0, 1\}^\kappa$. It returns k_b.
- If during this process, $\mathsf{Sim}_{\mathsf{CO}}$ ever sets both $k_0, k_1 \neq \perp$, it aborts.

In Step 3, $\mathsf{Sim}_{\mathsf{CO}}$ samples $e_0, e_1 \leftarrow \{0, 1\}^\kappa$ and sends them to R. After $\mathsf{Sim}_{\mathsf{CO}}$ performs Step 3 of the protocol, $\mathsf{Sim}_{\mathsf{CO}}$ inputs $(\mathsf{deliver}, sid, S)$ to $\mathcal{F}_{\mathrm{OT}}$ which prompts the output (output, sid) to S. Once S is corrupted (after Step 3), $\mathsf{Sim}_{\mathsf{CO}}$ learns m_0, m_1. It outputs (y, m_0, m_1) to $\mathcal{E}$.

- **R is corrupted between Step 2 and Step 4.** In this case, the simulation for R can be carried out as in the case where S is corrupted before Step 1. In addition, after S performs Step 3 of the protocol, $\mathsf{Sim}_{\mathsf{CO}}$ inputs $(\mathsf{receive}, sid, b), (\mathsf{deliver}, sid, S)$ to $\mathcal{F}_{\mathrm{OT}}$ (in this order). This prompts the output (output, sid) to S, since S is honest at this point and thus has input $(\mathsf{send}, sid, m_0, m_1)$ to $\mathcal{F}_{\mathrm{OT}}$. Moreover, $\mathsf{Sim}_{\mathsf{CO}}$ aborts upon setting both $k_0, k_1 \neq \perp$.
- **R is corrupted after Step 4.** Same as previous case, except that $\mathsf{Sim}_{\mathsf{CO}}$ does not have to abort if R is corrupted after S.

The proof of the following claim is almost identical to the one given for the case where S is corrupted between Step 1 and Step 3.

Claim. $\mathsf{Sim}_{\mathsf{CO}}$ does not abort in case the sender is corrupted after Step 3 except with probability $\frac{1}{q_H}\mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}_2}$, and $\mathcal{B}_2$ runs in roughly the same time as $\mathcal{E}$.

As long as $\mathsf{Sim}_{\mathsf{CO}}$ does not abort, it perfectly simulates the behavior of a party in Π_{CO}, as all outputs of the random oracle H are uniformly distributed in this case from the view of $\mathcal{E}$. Moreover, $\mathsf{Sim}_{\mathsf{CO}}$ can consistently simulate the view of $\mathcal{E}$. Finally, it is easy to see that all $\mathsf{Sim}_{\mathsf{CO}}$ can output algebraic representations of all elements that it outputs relative to group elements it receives as input, and hence $\mathsf{Sim}_{\mathsf{CO}}$ is algebraic. This concludes the proof. $\qquad\square$

4 Analysis of PAKE protocols: SPAKE2 and CPace

In this section we analyze the UC security of PAKE protocols SPAKE2 and CPace in the algebraic group model. We show that, modeling the hash functions used by these protocols as random oracles, they both achieve full UC security. The proofs are simpler than the ones we encountered in the literature for the UC and game-based security of the same protocols and they are based on standard (non-interactive) assumptions (we do not need *gap* assumptions). We use the standard definition of $\mathcal{F}_{\mathsf{pwKE}}$ from [1, 12] supporting multiple sessions Fig. 5.

Remark. Throughout the paper we present the simulators as running their own instances of the random oracle functionality used by the protocols, which means that we assume that the random oracle is local to the protocol [11]. However, in this section, we make it clear that none of the given simulators needs to program the random oracle functionality and, in the case of the SPAKE2 protocol, it does not even need to know which adversarial queries were made to the random oracle. These observations indicate that our proofs of security may carry over to a setting with global random oracle as in [9, 13]. Providing a full formalization of the referred works in the AGM is beyond the scope of this paper; however, we believe that our formal approach will carry naturally to extensions of UC with global functionalities.

Remark. The SPAKE2 simulator does not need to program the common reference string and the CPace protocol does not use one (in addition to the group description). We also show for both protocols that the simulators are algebraic. This means that the UC-AGM composition applies to both protocols.

Functionality $\mathcal{F}_{\mathsf{pwKE}}$

Upon receiving a query $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \mathsf{role})$ **from party** P:
  Ignore this query if record $(sid, P, \cdot, \cdot, \cdot)$ already exists. Otherwise, record $(sid, P, P', \mathsf{pw}, \mathsf{role})$ marked fresh
  and send $(\mathsf{NewSession}, sid, P, P', \mathsf{role})$ to $\mathcal{S}$.

Upon receiving a query $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ **from** $\mathcal{S}$:
  If $\exists$ a fresh record $(sid, P, P', \mathsf{pw}, \cdot)$ then:
    − If $\mathsf{pw}^* = \mathsf{pw}$ then mark it compromised and return "correct guess";
    − If $\mathsf{pw}^* \neq \mathsf{pw}$ then mark it interrupted and return "wrong guess".

Upon receiving a query $(\mathsf{NewKey}, sid, P, K^*)$ **from** $\mathcal{S}$:
  If $\exists$ a record $(sid, P, P', \mathsf{pw}, \mathsf{role})$ not marked completed then do:
    − If the record is compromised, or either P or P' is corrupted, then set $K := K^*$.
    − If the record is fresh and $\exists$ a completed record $(sid, P', P, \mathsf{pw}, \mathsf{role}', K')$ with $\mathsf{role}' \neq \mathsf{role}$ that was fresh
      when P' output (sid, K'), then set $K := K'$.
    − In all other cases pick K uniformly at random.
  Finally, append K to record $(sid, P, P', \mathsf{pw}, \mathsf{role})$, mark it completed, and output (sid, K) to P.

Fig. 5. The password-based key-exchange functionality $\mathcal{F}_{\mathsf{pwKE}}$.

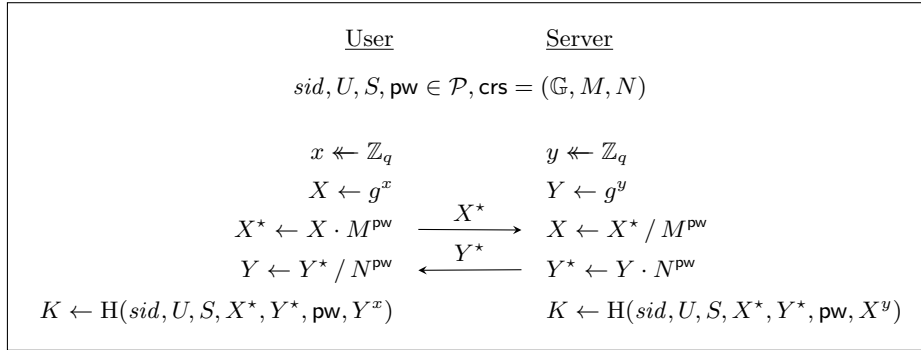| User		Server
	$sid, U, S, \mathsf{pw} \in \mathcal{P}, \mathsf{crs} = (\mathbb{G}, M, N)$	
$x \twoheadleftarrow \mathbb{Z}_q$		$y \twoheadleftarrow \mathbb{Z}_q$
$X \leftarrow g^x$		$Y \leftarrow g^y$
$X^\star \leftarrow X \cdot M^{\mathsf{pw}}$	$\xrightarrow{\quad X^\star \quad}$	$X \leftarrow X^\star / M^{\mathsf{pw}}$
$Y \leftarrow Y^\star / N^{\mathsf{pw}}$	$\xleftarrow{\quad Y^\star \quad}$	$Y^\star \leftarrow Y \cdot N^{\mathsf{pw}}$
$K \leftarrow \mathrm{H}(sid, U, S, X^\star, Y^\star, \mathsf{pw}, Y^x)$		$K \leftarrow \mathrm{H}(sid, U, S, X^\star, Y^\star, \mathsf{pw}, X^y)$

Fig. 6. The SPAKE2 protocol [3]. The CRS includes the group description $\mathbb{G}$, where $|\mathbb{G}| = q$ and two group elements $M, N \in \mathbb{G}$ sampled uniformly at random.

4.1 SPAKE2

Fig. 6 shows a SPAKE2 protocol execution between an user U and a server S. SPAKE2 is a two-pass protocol, where we assume the user plays the role of the initiator and the server that of the responder.

Let SqDH denote the problem of computing g^{a^2}, when given a random element $\mathbf{A} = g^a$ in $\mathbb{G}$, and $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{SqDH}}$ the probability that attacker $\mathcal{B}$ succeeds in solving this problem. Theorem 6 shows that SPAKE2 AGM realizes $\mathcal{F}_{\mathsf{pwKE}}$ assuming that SqDH and the discrete-logarithm problems are hard in $\mathbb{G}$.

Theorem 6. *SPAKE2 AGM-emulates* $\mathcal{F}_{\mathsf{pwKE}}$ *in the* $(\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}})$-*hybrid model under static corruptions. More precisely, there exists an algebraic simulator* $\mathcal{S}$ *for the (algebraic) dummy adversary* $\mathcal{D}$ *such that, for every efficient algebraic environment* $\mathcal{E}$ *creating at most* q_S *sessions and placing at most* q_H *queries to the random oracle, there exist attackers* $\mathcal{B}_1^1$, $\mathcal{B}_1^2$, *and* $\mathcal{B}_2$ *running in roughly the same time as* $\mathcal{E}$ *such that* $\mathrm{EXEC}_{\mathsf{pwKE}, \mathcal{S}, \mathcal{E}} \approx_\epsilon \mathrm{EXEC}_{\mathrm{SPAKE2}, \mathcal{D}, \mathcal{E}}$, *where*

$$\epsilon \leq \mathsf{Adv}_{\mathcal{B}_1^1}^{\mathsf{DL}} + \mathsf{Adv}_{\mathcal{B}_1^2}^{\mathsf{DL}} + q_H \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{SqDH}} + \frac{q_S + 1}{q} .$$

Note, that the DL problem and the SqDH problems are equivalent when we consider algebraic attackers, so the theorem follows with a reduction to the DL problem even if the proof relies on an apparently stronger assumption.

We also remark that the structure of this proof is much simpler than the proof that SPAKE2 satisfies relaxed UC PAKE security [1]. This is because, in the AGM, the password guessing event can be detected

16

directly by the simulator (and hence by the reductions) and one does not need to rely on the random oracle to extract passwords in active attacks.

We give a sketch of the proof and provide the full proof in Appendix C .

Proof (Sketch). Simulator $\mathcal{S}$ is shown in Fig. 7. Recall that, whenever the dummy adversary is instructed to deliver a group element to an uncorrupted party, it will output on its auxiliary tape the algebraic representation of that element with respect to group elements that appear in the view of the environment. In this case, the bases for such representations include $\mathbf{M}$, $\mathbf{N}$ and any messages $\mathbf{X}^\star$ or $\mathbf{Y}^\star$ produced by an uncorrupted party.

Simulation strategy. The simulator generates all messages of uncorrupted parties by raising either $\mathbf{M}$ or $\mathbf{N}$ to a random exponent. It does so because it does not know the corresponding password. The distribution of such messages is identical to those produced by honest parties in the real world, which are of the form $g^x \mathbf{M}^{\mathsf{pw}}$ or $g^y \mathbf{N}^{\mathsf{pw}}$. The simulator then keeps track of whether the adversary is launching a passive attack or an active attack: where the former means that there exists another simulated session with a consistent view. All passively attacked sessions are not interrupted by the simulator, which means that $\mathcal{F}_{\mathsf{pwKE}}$ will choose independent keys at the associated dummy parties' outputs.

For actively attacked sessions, the simulator checks if the delivered message was constructed as per the protocol and, if so, it extracts the password. All malformed messages cause the simulator to interrupt the session in the functionality by calling TestPw with $\mathsf{pw} = \perp$. For well formed messages, the simulator queries TestPw on the extracted password and, if the password is correct, computes the correct key: this is possible because, even though the simulator does not know the correct exponent implicit in the simulated honest party's state, it knows the algebraic decomposition of the delivered message and this is well formed (this means it can compute the key as the adversary would). If the password is incorrect, the simulator generates a totally random key (this is ignored by the functionality if there are no corrupt parties involved in the session, but it is relevant otherwise as we discuss below).

The simulation is perfect for all sessions with well-formed messages and correct password guesses. It looks perfect for all other sessions, unless the attacker can query the random oracle on the group element that such a session would compute in the real world. Our proof shows that any such query can, with overwhelming probability, be used to solve the SqDH problem. Two important observations for the proof: i. the simulator never uses the random exponents it generates for the honest party messages to perform any computation; and ii. the simulator never constructs any group element for which it cannot provide an algebraic decomposition to bases g, $\mathbf{M}$ and $\mathbf{N}$. The second observation guarantees that our simulator is an algebraic adversary as required by the composition theorem in Section 2.

Corrupt parties. Fig. 7 does not show explicitly the simulator's handling of sessions involving corrupt parties. In this case, the environment tells the simulator what the corrupt party should be doing, and the simulator does not keep the state of the corrupt party. Moreover, any group elements transmitted by the corrupt party come with their algebraic decomposition as above. Our simulator is structured to handle this case identically to the setting where the uncorrupted party is actively attacked while interacting with another uncorrupted party; we explain why this is the case in the detailed version of the proof provided in Appendix C . The proof below covers this scenario as a particular case.

Proof of simulator correctness. From this point on we consider only interactions involving uncorrupted parties. The first observation we make is that the distribution of the protocol messages produced by the simulator is identical to that occurring in the real world, even though they are constructed differently. It therefore remains to prove that the outputs of the ideal functionality match the distribution of the parties' outputs in the real world. We observe that the real and ideal worlds are identical until bad, where *bad* is defined as the event that a secret key that is selected uniformly at random by the functionality at the output of an uncorrupted party is inconsistent with the answer given by H to the adversary. This is because in all other cases the simulator programs the output of the ideal functionality consistently with the real world. This means formally that, for $\epsilon = \Pr[\text{EXEC}_{\mathsf{pwKE},\mathcal{S},\mathcal{E}} \Rightarrow \mathsf{bad}]$, we have

$$\text{EXEC}_{\mathsf{pwKE},\mathcal{S},\mathcal{E}} \approx_\epsilon \text{EXEC}_{\text{SPAKE2},\mathcal{D},\mathcal{E}}$$

SIMULATOR $\mathcal{S}$ for SPAKE2

proc INITIALIZE( )
Get CRS=$(\mathbf{M}, \mathbf{N})$

On input (NewSession, sid, P, P', role) from $\mathcal{F}_{\mathsf{pwKE}}$
If $\neg(\pi_P^{sid} = \perp)$ discard input.
If role = init
    $x \twoheadleftarrow \mathbb{Z}_q$; $\mathbf{X}^\star \leftarrow \mathbf{M}^x$; $\pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}^\star, \perp), \perp, \mathrm{init})$
    Send SENDINIT$(P, P', sid, \mathbf{X}^\star)$ to $\mathcal{E}$
Else
    $y \twoheadleftarrow \mathbb{Z}_q$; $\mathbf{Y}^\star \leftarrow \mathbf{N}^y$; $\pi_P^{sid} \leftarrow (y, (P', P, sid, \perp, \mathbf{Y}^\star), \perp, \mathrm{resp})$

On message SENDINIT$(P, P', sid, (\mathbf{X}^\star, \mathsf{alg}))$ from $\mathcal{E}$
Ignore if $\pi_{P'}^{sid} \neq (y, (P, P', sid, \perp, \mathbf{Y}^\star), \perp, \mathrm{resp})$
(A unique $\pi_{P'}^{sid}$ satisfies the above check for some y and $\mathbf{Y}^\star$)
$K \twoheadleftarrow \mathcal{K}$
(Can't interrupt passive sessions so $\mathcal{F}$ sets $=K$ at output)
If $\pi_P^{sid} = (\cdot, (P, P', sid, \mathbf{X}^\star, \perp), \perp, \mathrm{init})$ Jump to COMPLETE
(First check whether $\mathbf{X}^\star$ was constructed as per protocol)
If $\mathsf{alg} = [(g, x); (\mathbf{M}, \mathsf{pw})]$
    Query (TestPwd, sid, P', pw) to $\mathcal{F}_{\mathsf{pwKE}}$
    If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess"
        $Y \leftarrow \mathbf{Y}^\star / \mathbf{N}^{\mathsf{pw}}$; $K \leftarrow \mathrm{H}(P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, Y^x)$
(Interrupt all other sessions so independent key is set)
Else Query (TestPwd, $sid, P', \perp$) to $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: Send SENDRESP$(P', P, sid, \mathbf{Y}^\star)$ to $\mathcal{E}$
           $\pi_{P'}^{sid} \leftarrow (y, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), K, \mathrm{resp})$
           Query (NewKey, sid, P', K) to $\mathcal{F}_{\mathsf{pwKE}}$

On message SENDRESP$(P', P, sid, (\mathbf{Y}^\star, \mathsf{alg}))$ from $\mathcal{E}$
Ignore if $\pi_P^{sid} \neq (x, (P, P', sid, \mathbf{X}^\star, \perp), \perp, \mathrm{init})$
(A unique π_P^{sid} satisfies the above check for some x and $\mathbf{X}^\star$)
$K \twoheadleftarrow \mathcal{K}$
(Can't interrupt passive sessions so $\mathcal{F}$ sets $=K$ at output)
If $\pi_{P'}^{sid} = (\cdot, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), \cdot, \mathrm{resp})$ Jump to COMPLETE
(First check whether $\mathbf{Y}^\star$ was constructed as per protocol)
If $\mathsf{alg} = [(g, y); (\mathbf{N}, \mathsf{pw})]$
    Query (TestPwd, sid, P, pw) to $\mathcal{F}_{\mathsf{pwKE}}$
    If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess"
        $X \leftarrow \mathbf{X}^\star / \mathbf{M}^{\mathsf{pw}}$; $K \leftarrow \mathrm{H}(P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, X^y)$
(Interrupt all other sessions so independent key is set)
Else Query (TestPwd, $sid, P, \perp$) to $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: $\pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), K, \mathrm{init})$
           Query (NewKey, sid, P, K) to $\mathcal{F}_{\mathsf{pwKE}}$

Fig. 7. The operation of the SPAKE2 simulator. The simulator does not need to observe adversarial random oracle queries nor program either of the random oracle or the CRS.

More precisely, we define event bad as the existence within the set of queries placed by $\mathcal{E}$ to the random oracle of a query $(sid, P, P', \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, \mathbf{Z})$ that is *consistent* with the trace of a session at an uncorrupted party, which accepted after a passive attack or after an active attack where the simulator did not place a correct TestPw query. We define now these conditions formally.

We say a random oracle query $(sid, P, P', \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, \mathbf{Z})$ is *consistent* with an initiator session π_P^{sid} if this instance was created following a $\mathsf{NewSession}$ query by $\mathcal{E}$ using pw and $\pi_P^{sid} = (\,\cdot\,, (P, P', \mathbf{X}^\star, \mathbf{Y}^\star), \,\cdot\,, \mathsf{init})$. Similarly, the condition for responder session $\pi_{P'}^{sid}$ is $\pi_{P'}^{sid} = (\,\cdot\,, (P, P', \mathbf{X}^\star, \mathbf{Y}^\star), \,\cdot\,, \mathsf{resp})$. Note that the order of party identities in the trace determines the role of the party.

We say an initiator session π_P^{sid} *accepted after a passive attack* if it completed following a $\mathrm{SENDRESP}(P', P, sid, (\mathbf{Y}^\star, \mathsf{alg}))$ message from $\mathcal{E}$, when $\pi_{P'}^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), \,\cdot\,, \mathsf{resp})$. We say a responder session π_P^{sid} *accepted after a passive attack* if it completed following a $\mathrm{SENDINIT}(P, P', sid, (\mathbf{X}^\star, \mathsf{alg}))$ message from $\mathcal{E}$, when $\pi_P^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), \,\cdot\,, \mathsf{init})$. All other sessions are considered to be *under active attack*.

We reduce the probability of bad to SqDH. Intuitively, our reduction embeds the SqDH challenge $\mathbf{A}$ in the global parameters $(\mathbf{M} = \mathbf{A}, \mathbf{N} = \mathbf{A}^\delta)$ for δ sampled uniformly at random from $\mathbb{Z}_q^\star$ (this accounts for the $1/q$ term in the theorem statement). Suppose bad is first set for a random oracle entry that is consistent with a session accepted by an initiator session. The attacker delivered a message $\mathbf{Y}^\star$ and an algebraic representation that we can see as $[(a, g), (b, \mathbf{M}), (c, \mathbf{N})]$. The reduction can transform this algebraic representation into $[(a, g), (b + \delta c, \mathbf{A})]$. Let $\mathsf{CDH}(\mathbf{A}, \mathbf{B}) = g^{ab}$ for $\mathbf{A} = g^a$ and $\mathbf{B} = g^b$. This means that any problematic random oracle query will include a group element of the form

$$\mathbf{Z} = \mathsf{CDH}(\mathbf{A}^{x-\mathsf{pw}}, g^a \mathbf{A}^{b+\delta(c-\mathsf{pw})}) = \mathsf{CDH}(\mathbf{A}^{x-\mathsf{pw}}, g^a) \cdot \mathsf{CDH}(\mathbf{A}^{x-\mathsf{pw}}, \mathbf{A}^{b+\delta(c-\mathsf{pw})})$$

where x was chosen by the reduction following the simulator code. Since the reduction can compute the first factor, we can recover

$$\mathsf{CDH}(\mathbf{A}, \mathbf{A})^{(x-\mathsf{pw})(b+\delta(c-\mathsf{pw}))}$$

which means that the required SqDH solution can be recovered when $(x - \mathsf{pw})(b + \delta(c - \mathsf{pw})) \neq 0$. The case of responders is similar, but we can only recover the SqDH result provided $\delta(y - \mathsf{pw})(b + \delta c - \mathsf{pw}) \neq 0$.

The detailed proof given Appendix C bounds the probability that our reduction strategy fails using a statistical term and reductions $\mathcal{B}_1^1$ and $\mathcal{B}_1^2$ to the discrete logarithm problem. Once this possibility is excluded, we can reduce the probability of bad to SqDH. The detailed proof also includes the code for the algorithm $\mathcal{B}_2$ that breaks SqDH if the bad event occurs. In this case, we know that the random oracle table will contain a solution to SqDH if the event bad has occurred. When the experiment terminates, $\mathcal{B}_2$ therefore samples a random oracle query uniformly at random[6] and looks for a consistent session. It could find one or two consistent sessions, where the latter case corresponds to a passive attack with matching passwords on both sides. In any case, it computes a candidate SqDH value using the appropriate initiator or responder-side formula we described above. If the randomly selected random oracle entry was the first to cause the bad event, the algorithm solves SqDH. This accounts for the q_H multiplicative loss in the theorem statement. □

Remark. The above proof strategy can be used almost without change for an alternative version of the protocol that does not include the password pw in the input to the key derivation hash. This has practical advantages, as the password need not be kept in memory after computing the outgoing message. This version of the protocol was introduced as SPAKE1 in [3], and it was previously not known that this protocol could achieve forward secrecy or UC security. The only point where the current proof would need to be modified is in the final computation of the SqDH solution: in the particular case of a passive attack there now could be two protocol instances at P and P' with different passwords, but matching the same random oracle entry. In this case, the reduction would toss a coin and choose one of them at random to fix the password used to compute the SqDH solution. This adds only a factor of 2 to the final reduction step.

Furthermore, the same proof applies to both protocols when we can rely on a DDH oracle to the fixed basis $\mathbf{A}$ to look for the offending random oracle entry. In this case, we get a tight reduction to *Strong* SqDH

---

[6] This step could be replaced with a search for a consistent entry using a DDH oracle to the fixed basis $\mathbf{A}$, resulting in a tighter reduction to Strong SqDH where the q_H factor disappears.

for both protocols, i.e., the strong DH assumption adapted to the computation of g^{a^2}. Finally, the proof also applies to variants of the protocol discussed in [**?**], whereby the CRS is defined as $(M, N = M)$, or when the CRS is simply the group description and $(M, N) = H(sid, U, S)$ and H is modeled as a random oracle.

4.2 CPace

Fig. 8 shows a CPace protocol execution between an user U and a server S. CPace is a two-pass protocol, where we assume the user plays the role of the initiator and the server that of the responder. We give a sketch of the proof here and provide the complete proof in Appendix D .
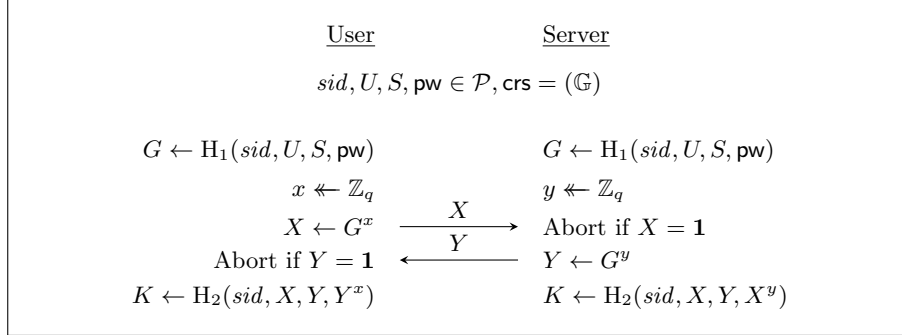
User	Server
$sid, U, S, \mathsf{pw} \in \mathcal{P}, \mathsf{crs} = (\mathbb{G})$	

User		Server
$G \leftarrow H_1(sid, U, S, \mathsf{pw})$		$G \leftarrow H_1(sid, U, S, \mathsf{pw})$
$x \twoheadleftarrow \mathbb{Z}_q$		$y \twoheadleftarrow \mathbb{Z}_q$
$X \leftarrow G^x$	$\xrightarrow{\quad X \quad}$	Abort if $X = \mathbf{1}$
Abort if $Y = \mathbf{1}$	$\xleftarrow{\quad Y \quad}$	$Y \leftarrow G^y$
$K \leftarrow H_2(sid, X, Y, Y^x)$		$K \leftarrow H_2(sid, X, Y, X^y)$

Fig. 8. The CPace protocol [19]. CRS includes the group description $\mathbb{G}$ s.t. $|\mathbb{G}| = q$.

Let InvCDH denote the problem of computing $g^{1/a}$ when given a random element $\mathbf{A} = g^a$ in $\mathbb{G}$ and let $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{InvCDH}}$ denote the probability that attacker $\mathcal{B}$ solves this problem. Theorem 7 shows that CPace AGM realizes $\mathcal{F}_{\mathsf{pwKE}}$ if InvCDH is hard in $\mathbb{G}$.

Theorem 7. *CPace AGM-emulates $\mathcal{F}_{\mathsf{pwKE}}$ under static corruptions, in a hybrid model with the random oracle functionality. More precisely, there exists an algebraic simulator $\mathcal{S}$ for the (algebraic) dummy adversary $\mathcal{D}$ such that, for every efficient algebraic environment $\mathcal{E}$ creating at most q_S sessions, querying H_1 at most q_{H_1} times and querying H_2 at most q_{H_2} times, there exists $\mathcal{B}_{\ell_1, \ell_2}$ running in roughly the same time as $\mathcal{E}$ such that $\mathrm{EXEC}_{\mathsf{pwKE}, \mathcal{S}, \mathcal{E}} \approx_\epsilon \mathrm{EXEC}_{\mathrm{CPACE}, \mathcal{D}, \mathcal{E}}$, where*

$$\epsilon \leq q_{H_1} \cdot q_{H_2} \cdot \mathsf{Adv}_{\mathcal{B}_{\ell_1, \ell_2}}^{\mathsf{InvCDH}}() + \frac{q_{H_1}^2 + q_S}{q} .$$

Note that the InvCDH problem is equivalent to the DL problem when we consider algebraic attackers, so the theorem follows with a reduction to the DL problem even if the proof relies on this apparently stronger assumption.

Proof. (Sketch) Simulator $\mathcal{S}$ is shown in Fig. 9. The simulation strategy here is identical to that we adopt for the SPAKE2 proof, with the caveat that the simulator must learn the environment's queries to H_1 in order to extract the password in an active attack. (In this case, the bases for the algebraic representations of adversarially constructed messages include the outputs of the random oracle H_1 and any messages $\mathbf{X}$ or $\mathbf{Y}$ produced by an uncorrupted party.) Also here the simulator never generates any group element for which it cannot give an algebraic decomposition with respect to base g, and hence it is an algebraic adversary. The handling of corrupt parties is also the same.

Proof of simulator correctness. This part of the proof is also similar in structure to that of SPAKE2. We first eliminate some corner cases, where the distribution of real world and the ideal world views do not match, but are straightforward to bound using a statistical term; this includes collisions at the random oracle output. We then conclude that the real and ideal worlds are identical until bad, where *bad* is defined as the existence within the set of queries placed by $\mathcal{E}$ to H_2 of a query $(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$ that is *consistent* with the

SIMULATOR $\mathcal{S}$ for CPace

proc $H_1(sid, P, P', \mathsf{pw})$ (non-repeat queries)
$r \twoheadleftarrow \mathbb{Z}_q$; $\mathbf{G} \leftarrow g^r$; $T_1[sid, P, P', \mathsf{pw}] \leftarrow \mathbf{G}$; return $\mathbf{G}$
Simulator aborts if at any point T_1 is non-injective.

On input $(\mathsf{NewSession}, sid, P, P', \mathsf{role})$ from $\mathcal{F}_{\mathsf{pwKE}}$
If $\neg(\pi_P^{sid} = \perp)$ discard input.
If role = init
     $\hat{x} \twoheadleftarrow \mathbb{Z}_q^*$; $\mathbf{X} \leftarrow g^{\hat{x}}$
     $\pi_P^{sid} \leftarrow (\hat{x}, (P, P', sid, \mathbf{X}, \perp), \perp, \text{init})$
     Send $\textsc{SendInit}(P, P', sid, \mathbf{X})$ to $\mathcal{D}$
Else $\hat{y} \twoheadleftarrow \mathbb{Z}_q^*$; $\mathbf{Y} \leftarrow g^{\hat{y}}$
     $\pi_P^{sid} \leftarrow (\hat{y}, (P', P, sid, \perp, \mathbf{Y}), \perp, \text{resp})$

On message $\textsc{SendInit}(P, P', sid, (\mathbf{X}, \mathsf{alg}) \neq \mathbf{1})$ from $\mathcal{E}$ via $\mathcal{D}$
Ignore if $\pi_{P'}^{sid} \neq (\hat{y}, (P, P', sid, \perp, \mathbf{Y}), \perp, \text{resp})$
(A unique $\pi_{P'}^{sid}$ satisfies the above check for some $\hat{y}$ and $\mathbf{Y}$)
$K \twoheadleftarrow \mathcal{K}$
(Can't interrupt passive sessions so $\mathcal{F}$ sets $=K$ at output)
If $\pi_P^{sid} = (\cdot, (P, P', sid, \mathbf{X}, \perp), \perp, \text{init})$ Jump to COMPLETE
(First check whether $\mathbf{X}$ was constructed as per protocol)
If $\mathsf{alg} = [(\mathbf{G}, x))] \wedge (sid, P, P', \mathsf{pw}, \mathbf{G}) \in T_1$
     Query $(\mathsf{TestPwd}, sid, P', \mathsf{pw})$ to $\mathcal{F}_{\mathsf{pwKE}}$
     If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" Then $K \leftarrow H_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Y}^x)$
(Interrupt all other non-passive sessions.)
Else Query $(\mathsf{TestPwd}, sid, P', \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: Send $\textsc{SendResp}(P', P, sid, \mathbf{Y})$ to $\mathcal{D}$
         $\pi_{P'}^{sid} \leftarrow (\perp, (P, P', sid, \mathbf{X}, \mathbf{Y}), K, \text{resp})$
         Query $(\mathsf{NewKey}, sid, P', K)$ to $\mathcal{F}_{\mathsf{pwKE}}$

On message $\textsc{SendResp}(P', P, sid, (\mathbf{Y}, \mathsf{alg}) \neq \mathbf{1})$ from $\mathcal{E}$ via $\mathcal{D}$
Ignore if $\pi_P^{sid} \neq (\hat{x}, (P, P', sid, \mathbf{X}, \perp), \perp, \text{init})$
(A unique π_P^{sid} satisfies the above check for some $\hat{x}$ and $\mathbf{X}$)
$K \twoheadleftarrow \mathcal{K}$
(Can't interrupt passive sessions so $\mathcal{F}$ sets $=K$ at output)
If $\pi_{P'}^{sid} = (\cdot, (P, P', sid, \mathbf{X}, \mathbf{Y}), \cdot, \text{resp})$ Jump to COMPLETE
(First check whether $\mathbf{Y}$ was constructed as per protocol)
If $\mathsf{alg} = [(\mathbf{G}, y)] \wedge (sid, P, P', \mathsf{pw}, \mathbf{G}) \in T_1$
     Query $(\mathsf{TestPwd}, sid, P, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{pwKE}}$
     If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" Then $K \leftarrow H_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{X}^y)$
(Interrupt all other non-passive sessions.)
Else Query $(\mathsf{TestPwd}, sid, P, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: $\pi_P^{sid} \leftarrow (\perp, (P, P', sid, \mathbf{X}, \mathbf{Y}), K, \text{init})$
         Query $(\mathsf{NewKey}, sid, P, K)$ to $\mathcal{F}_{\mathsf{pwKE}}$

Fig. 9. The operation of the CPace simulator. The simulator needs to observe adversarial random oracle queries on H_1 but not on H_2, and it does not need to program either of the random oracles. T_1 is initially empty.

trace of a session at an uncorrupted party, which accepted after a passive attack or after an active attack where the simulator did not place a correct TestPw query. We define now these conditions formally.

We say an H_2 query $(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$ is *consistent* with an initiator session π_P^{sid} if $\pi_P^{sid} = (\,\cdot\,, (\,\cdot\,, \,\cdot\,, \mathbf{X}, \mathbf{Y}), \,\cdot\,,$ init). Similarly, the condition for responder session $\pi_{P'}^{sid}$ is $\pi_{P'}^{sid} = (\,\cdot\,, (\,\cdot\,, \,\cdot\,, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{resp})$. We say an initiator session π_P^{sid} *accepted after a passive attack* if it completed following a $\text{SENDRESP}(P', P, sid, (\mathbf{Y}, \mathsf{alg}))$ message from $\mathcal{E}$, when $\pi_{P'}^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{resp})$. Responder session $\pi_{P'}^{sid}$ *accepted after a passive attack* if it completed after a $\text{SENDINIT}(P, P', sid, (\mathbf{X}, \mathsf{alg}))$ message from $\mathcal{E}$, when $\pi_P^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{init})$. All other sessions are considered to be *under active attack*. Finally, we say a T_1 entry of the form $(sid, P, P', \mathsf{pw})$ is *consistent* with an initiator (resp. responder) instance, if that instance was initialized by the environment in a NEWSESSION query with $(sid, P, P', \mathsf{pw}, \mathsf{init})$ (resp. $(sid, P', P, \mathsf{pw}, \mathsf{resp})$).

We bound the probability of bad in the ideal world using a sequence of games.

Guessing the RO entries that cause bad. We modify ideal world as follows: sample (ℓ_1, ℓ_2) uniformly at random in $[q_{H_1}] \times [q_{H_2}]$. Then, if bad first occurs due to the i-th H_2 query such that $i \neq \ell_2$, abort. Furthermore, if the offending T_1 entry (i.e., the T_1 unique entry consistent with the session where the bad event was detected) is not the ℓ_1-th one, abort. Clearly, we can still bound the probability of bad in the previous game with the pessimistic bound $q_{H_1} \cdot q_{H_2} \cdot \Pr[\mathsf{bad}]$, where we only check for bad if the experiment has not aborted. We give in Appendix D a reduction $\mathcal{B}_{\ell_1, \ell_2}$ that solves the InvCDH problem whenever bad occurs in this modified game.

Final reduction. The reduction strategy is as follows. The generator returned by H_1 for the problematic session associated with the ℓ_2-th password query is programmed to be $\mathbf{A}$, the InvCDH problem instance. All messages generated by uncorrupted parties are generated as $g^{\hat{x}}$ or $g^{\hat{y}}$. All random oracle queries consistent with a session with trace $(\mathbf{X}, \mathbf{Y})$ and generator $\mathbf{A}$ include the key element $\mathbf{Z}$ satisfying the following equation:

$$\mathbf{Z} = \mathbf{X}^{\mathsf{dlog}_\mathbf{A}(\mathbf{Y})} = \mathbf{Y}^{\mathsf{dlog}_\mathbf{A}(\mathbf{X})} = \mathbf{A}^{(\mathsf{dlog}_\mathbf{A}(\mathbf{X}) \cdot \mathsf{dlog}_\mathbf{A}(\mathbf{Y}))}$$

Observing that $\mathsf{dlog}_\mathbf{A}(\mathbf{X}) = \mathsf{dlog}_g(\mathbf{X})/\mathsf{dlog}_g(\mathbf{A})$ the equation can be re-written as:

$$\mathbf{Z} = g^{\frac{\mathsf{dlog}_g(\mathbf{X}) \cdot \mathsf{dlog}_g(\mathbf{Y})}{\mathsf{dlog}_g \mathbf{A}}}$$

In the simplest case of a passive attack, it is immediate that we recover the solution to the InvCDH problem if $\hat{x} \cdot \hat{y} \neq 0$, which we know to be the case.

Now let us suppose the problematic case occurs with an actively attacked initiator session. Then we know that $\mathbf{Y} = g^\alpha \mathbf{A}^\beta$ and $\alpha \neq 0$; otherwise this would be a correct password guess and the bad event could never have occurred for this session—recall the experiment would have aborted if H_1 did not program $\mathbf{A}$ as the output for the password associated with this session. We can therefore refine the equation above to:

$$\mathbf{Z} = g^{\frac{\hat{x} \cdot (\alpha + \beta \cdot \mathsf{dlog}_g(\mathbf{A}))}{\mathsf{dlog}_g \mathbf{A}}} = g^{\frac{\hat{x} \cdot \alpha}{\mathsf{dlog}_g \mathbf{A}} + \hat{x} \cdot \beta}$$

Again, the InvCDH solution can be recovered, as long as $\hat{x} \neq 0$. The responder session case is identical. $\square$

Remark. As in the proof of SPAKE2, we could eliminate the q_{H_2} factor in the reduction by using a DDH oracle to the fixed basis $\mathbf{A}$ to detect which of the entries in H_2 is consistent with the bad event; however, we would still be guessing the problematic H_1 query in order to program the hard problem instance, and the q_{H_1} factor would remain.

Remark. In the proofs for SPAKE2 and CPace, we have seen that it is possible to have tighter reductions to a stronger gap assumption that excludes the need to guess an entry in the key derivation random oracle. However, we should also mention that in the algebraic group model, the gap versions of the SqDH and InvCDH assumptions are actually equivalent to the standard versions, provided that the reduction is able to give algebraic decompositions of all the elements queried to the DDH oracle. This is the case in our proofs, provided that the attacker is also required to give algebraic decompositions of the group elements it queries

to the random oracle. Note that this is a requirement for algebraic environments in the UC AGM model, as explained in Section 2. The take away from this discussion is that our proof of SPAKE2 implies a tight reduction to SqDH in the algebraic group model for both SPAKE1 and SPAKE2 (SPAKE1 is the variant that does not include pw in the key derivation hash). The CPace proof implies a reduction to InvCDH in the AGM with a loss of q_{H_1}.

References

1. M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In *Advances in Cryptology—Crypto 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, 2020.
2. M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy*, pages 571–587. IEEE, 2015.
3. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Cryptographers' Track—RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, 2005.
4. B. Auerbach, F. Giacon, and E. Kiltz. Everybody's a target: Scalability in public-key encryption. In *Advances in Cryptology—Eurocrypt 2020, Part III*, volume 12107 of *LNCS*, pages 475–506. Springer, 2020.
5. B. Bauer, G. Fuchsbauer, and J. Loss. A classification of computational assumptions in the algebraic group model. In *Advances in Cryptology—Crypto 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, 2020.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conf. on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
7. T. Bradley, S. Jarecki, and J. Xu. Strong asymmetric PAKE based on trapdoor CKEM. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 798–825. Springer, 2019.
8. M. Byali, A. Patra, D. Ravi, and P. Sarkar. Fast and universally-composable oblivious transfer and commitment scheme with adaptive security. Cryptology ePrint Archive, Report 2017/1165, 2017. https://eprint.iacr.org/2017/1165.
9. J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The wonderful world of global random oracles. LNCS, pages 280–312. Springer, 2018.
10. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001.
11. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *4th Theory of Cryptography Conference—TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, 2007.
12. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology—Eurocrypt 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, 2005.
13. R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In *21st ACM Conf. on Computer and Communications Security (CCS)*, pages 597–608. ACM Press, 2014.
14. A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Advances in Cryptology—Eurocrypt 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, 2020.
15. T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology—Latincrypt 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.
16. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology—Crypto 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, 2018.
17. G. Fuchsbauer, A. Plouviez, and Y. Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In *Advances in Cryptology—Eurocrypt 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, 2020.
18. Z. A. Genç, V. Iovino, and A. Rial. "The simplest protocol for oblivious transfer" revisited. Cryptology ePrint Archive, Report 2017/370, 2017. http://eprint.iacr.org/2017/370.
19. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.
20. E. Hauck and J. Loss. Efficient and universally composable protocols for oblivious transfer from the CDH assumption. Cryptology ePrint Archive, Report 2017/1011, 2017. http://eprint.iacr.org/2017/1011.
21. J. Katz, J. Loss, and J. Xu. On the security of time-locked puzzles and timed commitments. Cryptology ePrint Archive, Report 2020/730, 2020. https://eprint.iacr.org/2020/730.
22. M. Larangeira and K. Tanaka. Programmability in the generic ring and group models. *J. Internet Serv. Inf. Secur.*, 1(2/3):57–73, 2011.

23. M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *26th ACM Conf. on Computer and Communications Security (CCS)*, pages 2111–2128. ACM Press, 2019.
24. M. Naor, S. Paz, and E. Ronen. Crisp: Compromise resilient identity-based symmetric pake. Cryptology ePrint Archive, Report 2020/529, 2020. https://eprint.iacr.org/2020/529.
25. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
26. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology—Eurocrypt '97*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.

A On using an ideal functionality to idealize a group

One can try to formalize the AGM within the UC framework by introducing an appropriate functionality $\mathcal{F}_{\mathsf{AGM}}$ that "forces" arbitrary algorithms to behave algebraically. For example, the functionality could generate a group $\mathbb{G}$ along with a group element g (that it gives to all parties), and then maintain a set $D \subseteq \mathbb{G}$ (initialized as $D = \{g\}$) containing the group elements that are currently "registered." Any party can check whether a group element h is registered by querying $\mathcal{F}_{\mathsf{AGM}}$ to see whether $h \in D$; a party can register a new group element h by sending h to $\mathcal{F}_{\mathsf{AGM}}$ along with a representation of h with respect to the elements that are currently registered, after which h will be added to D.

This approach would have the advantage of leaving the UC framework itself unchanged; once an appropriate $\mathcal{F}_{\mathsf{AGM}}$ functionality is defined, one could work in the $\mathcal{F}_{\mathsf{AGM}}$-hybrid model, while automatically inheriting existing composition theorems. Nevertheless, although we considered this approach we ultimately found it unsatisfactory for several reasons:

– This approach requires honest parties to be aware of the fact that they are working in the AGM; in particular, any protocol designed in this model would need to instruct honest parties that when they receive a group element h they must make a query to $\mathcal{F}_{\mathsf{AGM}}$ checking whether h is registered. While not a fundamental barrier, this is clunky and conceptually awkward, and does not match what would be done in a real-world protocol execution.
– A more serious problem is that it is not technically clear how composition would work for the version of $\mathcal{F}_{\mathsf{AGM}}$ described above (and we did not see any reasonable way to define $\mathcal{F}_{\mathsf{AGM}}$ that circumvents this issue). Consider the case of a protocol relying on the oblivious-transfer functionality $\mathcal{F}_{\mathsf{OT}}$. In the spirit of the AGM, any group elements h_0, h_1 sent to $\mathcal{F}_{\mathsf{OT}}$ by the sender should be registered—but this seems to require $\mathcal{F}_{\mathsf{OT}}$ *itself* to query $\mathcal{F}_{\mathsf{AGM}}$ to check whether that is the case. The same problem arises, e.g., when relying on the random-oracle functionality, if we insist that only registered group elements can be queried to the random oracle. Resolving this issue would thus, at least, require modifying standard functionalities when working in the AGM; even worse, it would require functionalities to communicate with each other, something that is disallowed in the standard UC framework.
– The preceding issue could perhaps be solved, at least partially, by making $\mathcal{F}_{\mathsf{AGM}}$ a *global* functionality [11]. This, however, is problematic because then the simulator is unable to learn the representations of group elements when they are registered, which seems to limit the applicability of working in the AGM to begin with.

For these reasons, we decided to reject the above in favor of our current approach, which adopts the standard AGM idea of restricting the classes of attackers, rather than changing the rules of the environments in which the attackers execute. This leads to simpler and easy to check proofs and it preserves the syntax of ideal functionalities and protocols.

B Further Details Regarding the UC-AGM

B.1 Detailed Description of the Simplified UC Framework

In this section, we give a detailed description of the (simplified) UC framework [10, Section 2]; see Appendix B.2 for discussion on the full UC framework.

Protocols. A *protocol* is treated as a set of machines, where each machine has a fixed program and a set of specifications telling it how to communicate and with whom. A machine can be viewed roughly as a "party," but machines can also be abstract computational entities such as ideal functionalities.

More concretely, a *machine* is a tuple $\mu = (ID, C, \tilde{\mu})$, where

- ID is μ's *identity*, which is unchanged throughout an execution of the protocol.
- C is μ's *communication set*, which specifies the parties to which μ can send messages.[7] It consists of pairs of the form (ID_i, L_i), where L_i is either input, subroutine-output, or backdoor. If $L_i = $ input, then μ is allowed to send input to the machine with identity ID_i (we say μ is a *caller* of ID_i); if $L_i = $ subroutine-output, then μ is allowed to send output to the machine with identity ID_i (we say μ is a *subroutine* of ID_i). As we shall see soon, backdoor is reserved for information sent to and from the adversary in an execution of the protocol.
- $\tilde{\mu}$ is μ's *program*. A program can be formally defined via some fixed computational model, such as interactive Turing machines (ITMs) or random access machines (RAMs). For concreteness, in this work we use ITMs as the model of computation.

Definition 7. *A* protocol *is a set of machines* $\pi = \{\mu_1, \ldots, \mu_n\}$, *where* $\mu_i = (ID_i, C_i, \tilde{\mu}_i)$, *with the following properties:*

- *The identity ID_i of each machine is unique.*
- *If π contains a machine $\mu = (ID, \star, \star)$ which is a caller of ID', then π also contains a machine $\mu' = (ID', \star, \star)$ which is a subroutine of ID.*
- *If π contains a machine $\mu' = (ID', \star, \star)$ which is a subroutine of ID, and π also contains a machine $\mu = (ID, \star, \star)$,[8] then μ is a caller of ID'.*

Every machine in a protocol π is either an *internal machine* or a *main machine*. A main machine is a machine that is a subroutine of an external identity.

For now we say that a machine is *efficient* if it halts within a number of steps polynomial in a global security parameter κ. (See Appendix B.2 for more discussion on this.)

Protocol execution. We distinguish between a protocol and its execution. Let $\pi = \{\mu_1, \ldots, \mu_n\}$ be a protocol, where $\mu_i = (ID_i, C_i, \tilde{\mu}_i)$. Without loss of generality, assume $\mu_1, \ldots, \mu_m$ (where $m \leq n$) are π's main machines. In an *execution of protocol π*, two additional machines are added. One is the *environment*

$$\mathcal{E} = \left(0, \left\{(ID_j, \mathsf{input})\right\}_{j=1}^m \cup \left\{(1, \mathsf{input})\right\}, \tilde{\mathcal{E}}\right);$$

the other is the *adversary*

$$\mathcal{A} = \left(1, \left\{(ID_i, \mathsf{backdoor})\right\}_{i=1}^n \cup \left\{(0, \mathsf{backdoor})\right\}, \tilde{\mathcal{A}}\right).$$

In other words, the environment $\mathcal{E}$ has identity 0, and can send inputs to all main machines of π, as well as the adversary; the adversary $\mathcal{A}$ has identity 1, and can send backdoor messages to all machines of π, as well as the environment. (We require that none of the identities of machines in π, nor π's external identities, is 0 or 1.) Furthermore, in an execution of π, we change C_i to $C_i \cup \{(1, \mathsf{backdoor})\}$. That is, every machine in π is additionally allowed to send backdoor message to the adversary.

An execution of π starts by running the environment $\mathcal{E}$; in other words, $\mathcal{E}$ is *activated*. Then when machine μ wants to transmit some message m (with identity ID) to some identity ID' together with a label (input, subroutine-output, or backdoor) as allowed by μ's communication set, the execution proceeds as follows:

- If $\mu = \mathcal{E}$, then m is added to the state of μ' with identity ID', together with a sender identity chosen by $\mathcal{E}$ from π's external identities.

---

[7] As we will see, this determines the machines from which μ can receive messages.

[8] Note that π may not contain such a machine. If not, then ID is called an *external identity* of μ'. Intuitively this is some party outside the protocol (e.g., in some higher-level protocol) that calls μ'.

– If $\mu \neq \mathcal{E}$ and ID' corresponds to some machine μ' in π, then m is added to the state of μ', together with the sender identity ID.

– If $\mu \neq \mathcal{E}$ and ID' is an external identity, then m is added to the state of $\mathcal{E}$, together with the sender identity ID.[9]

After that, μ's activation completes and μ' (or $\mathcal{E}$, if ID' is an external identity) is activated. If μ halts without sending any message, then $\mathcal{E}$ is activated.[10]

In the "plain" model, the adversary can modify any message as it wants. Restrictions on the adversary's ability can be modeled via adding suitable ideal functionalities (see below), such as the message authentication functionality and the secure message transmission functionality.

A message has four components: the sender identity, the (intended) receiver identity, the label, and the payload. For readability, we use (label, payload) to denote a message; when it is necessary to make the identity of the sender and/or receiver explicit, we may use the term "from/to (or aimed at) identity."

We use $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ to denote environment $\mathcal{E}$'s view in π's execution with $\mathcal{A}$.

Protocol emulation. We are now ready to define UC emulation of a protocol. Roughly speaking, a protocol π *emulates* another protocol ϕ, if any efficient environment's view in an execution of π (for any efficient adversary) can be "simulated" in an execution of ϕ. Concretely,

Definition 8. *Protocol π* emulates *protocol ϕ if the following holds: for any efficient adversary $\mathcal{A}$ in π's execution, there is an efficient adversary $\mathcal{S}$ (called the* simulator*) in ϕ's execution, such that for any efficient environment $\mathcal{E}$ we have*

$$\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{E}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}}.$$

An immediate observation is that if π emulates ϕ, then the number and identities of main machines they each have must be the same; otherwise the environment can distinguish trivially. (We say that π is *compatible* with ϕ.) Also note that both $\mathcal{A}$ and $\mathcal{S}$ have identity 1.

Party corruption. Party corruption must be modeled carefully; in particular, it must be accounted for by the environment somehow, since otherwise the simulator could simply corrupt all parties at the outset and then simulate somewhat trivially. We provide the following simple mechanism. A main machine $\mathcal{R}$, called the *record keeping machine*, is added to the protocol. When the adversary sends a (backdoor, corrupt) message to a protocol machine μ, μ notifies $\mathcal{R}$ that it is corrupted. $\mathcal{R}$ maintains a list of all corrupted parties; upon query from the environment, $\mathcal{R}$ returns this list.

The two common corruption settings are *adaptive corruption*, where the adversary can corrupt a party at any time; and *static corruption*, where party corruption must happen at the outset (so a party remains either corrupted or uncorrupted throughout protocol execution). Static corruption can be modeled via modifying the code of $\mathcal{R}$ as follows: upon activation of the adversary, it can send $\mathcal{R}$ a set of parties to be corrupted; all subsequent corrupt messages sent to $\mathcal{R}$ are ignored.

Realizing an ideal functionality. The most common form of UC emulation is *realizing an ideal functionality*. That is, the protocol emulated is an *ideal protocol* $\mathrm{IDEAL}_{\mathcal{F}}$ consisting of an incorruptible internal machine called the *ideal functionality* $\mathcal{F}$, and m main machines called *dummy parties*, which merely pass all messages between their external identities and $\mathcal{F}$ (recall that in a protocol execution, messages from/to an external identity eventually comes from/goes to the environment); in particular, they ignore all messages from the adversary (except for corrupt messages discussed above).

---

[9] Formally, an ITM has three *externally writable tapes*: an input tape, a subroutine-output tape, and a backdoor tape. The incoming message m is written on one of these tapes of μ' (or $\mathcal{E}$), depending on the label.

[10] As we can see, the environment $\mathcal{E}$ essentially covers all external identities of π. It might look odd that we let the main machines communicate with some external identities, rather than with $\mathcal{E}$ directly. This is for the sake of composability: in the case that π is a sub-protocol of some higher-level protocol ρ^{π}, we need π's main machines to communicate with machines in $\rho := \rho^{\pi} \setminus \pi$, which has the corresponding external identity. If we only allowed π's main machines to communicate with identity 0 (i.e., the environment), then they would not be able to communicate with their "callers" in ρ, and thus composability could not be well-defined.

Definition 9. *Protocol π* realizes *ideal functionality $\mathcal{F}$ if π emulates* IDEAL$_{\mathcal{F}}$.

Universal composition. The *universal composition* operation, which is the key concept behind the composition theorem, involves three protocols ϕ, ρ^{ϕ} and π, with the following properties:

1. ϕ is a *sub-protocol* of ρ^{ϕ}: that is, $\phi \subset \rho^{\phi}$ (recall that we simply treat a protocol as a set of machines), and ϕ is a protocol in itself.
2. π is *compatible* with ϕ: recall that this means that the number and identities of π's and ϕ's main machines are the same.
3. π is *identity-compatible* with ϕ and ρ^{ϕ}: that is, no machine in π has the same identity as any machine in $\rho^{\phi} \setminus \phi$.

It is easy to see that, given the three conditions above, $\rho^{\pi} := \rho^{\phi} \setminus \phi \cup \pi$ also forms a protocol which is compatible with ρ^{ϕ}; that is, in ρ^{ϕ} we can replace the sub-protocol ϕ with π.

The following *universal composition theorem* is proven in [10]:

Theorem 8. *Suppose protocols ρ^{ϕ}, π, ϕ are such that ϕ is a sub-protocol of ρ^{ϕ}, π emulates ϕ, and π is identity-compatible with ρ^{ϕ} and ϕ. Then ρ^{π} emulates ρ^{ϕ}.*

When an ideal protocol IDEAL$_{\mathcal{F}}$ is a sub-protocol of $\rho^{\text{IDEAL}_{\mathcal{F}}}$, we simplify the notation to $\rho^{\mathcal{F}}$ and say that protocol $\rho^{\mathcal{F}}$ is in the $\mathcal{F}$-*hybrid world*.

B.2 The Full UC Framework

We explain the main differences between the simplified UC framework used in Section 2 and the full UC framework, as well as how our treatment of algebraic adversaries there can be extended to the full framework. Our discussion here is mostly informal; for a formal treatment, we refer the readers to [10].

Running time. Recall that in Appendix B.1 we define a machine to be efficient if it halts within a number of steps polynomial in the security parameter. While helping keep the concept simple, this definition appears overly restrictive; for example, in an encryption scheme where the encryption algorithm is "efficient" in this sense, the length of plaintexts is *a priori* bounded.

A natural alternative approach is to replace the security parameter with the input length in the definition of efficient machines. However, in a distributed system where there are multiple *interactive* Turing machines, things are more subtle. In particular, if we define an efficient ITM as halting within a number of steps polynomial in its *overall* input length, then this does not place any realistic limit on its running time: since two ITMs may repeatedly send inputs to each other with increasing length, they may run infinitely even though they are both "efficient."

To resolve this issue, [10] replaces input length with "runtime budgets." That is, every message is associated with a natural number called *import*; at any specific state, the *runtime budget* of an ITM is defined as the overall import of messages it has received so far minus the overall import of messages it has sent so far. An ITM is efficient if at any state, the number of steps it has taken is at most some polynomial of its runtime budget at this state. In essence, the import of an message can be viewed as a "token" that provides running time; these "tokens" are circulated around the system, but the total number is an invariant.

To make the definition of UC emulation meaningful, [10] also considers only *balanced environments*, whose overall import to protocol machines never exceeds the overall import to the adversary. For more detailed discussion, see [10, Section 4.2].

Sender and receiver identities. Recall that a message in Appendix B.1 consists of label, payload, sender identity, and receiver identity. There we assume that the identities of all parties are public. However, in a dynamic system, such assumption cannot be made; for example, it is possible that the sender only knows part of the receiver's identity, or the sender might even want to invoke a new machine as the receiver.

To this end, [10] adds two more fields (apart from the import discussed above) to the message: a *forced-write* flag f, and a *reveal-sender-id* flag r. If $f = 1$ then the receiver's identity must exactly match the

one specified in the message, μ'; if no machine in the system has identity μ', then one with identity μ' is invoked. If $f = 0$ then μ' is interpreted as a predicate on identities, and the message is delivered to the earliest-invoked machine whose identity satisfies μ' (if no machine's identity satisfies μ', then the message is not delivered). This mechanism allows for delivery of messages without knowing the receiver's full identity (by setting $f = 0$), as well as invocation of new machines (by setting $f = 1$). Finally, if $r = 1$ and the message is delivered, then the sender's identity is sent together with the message.

In an execution of a protocol, the environment $\mathcal{E}$ must set the forced-write flag $f = 1$, i.e., $\mathcal{E}$ must know the identities of all machines it communicates with, and is free to invoke new machines; the adversary $\mathcal{A}$ must set $f = 0$, i.e., $\mathcal{A}$ can only communicate with existing machines; the protocol parties must set $f = 0$ while sending backdoor messages, i.e., they cannot invoke a new adversary, and they must reveal their identities while sending messages by setting $r = 1$.[11]

Protocol sessions and ITIs. The model in Appendix B.1 implicitly assumes that the total number of protocol sessions is fixed in advance (where each session of each party is an ITM). In order to take into account the dynamic generation of protocol sessions, the full UC framework instead lets the entire protocol to be a single ITM, and assigns multiple session IDs within this ITM; every combination of protocol code/party ID/session ID, called an *extended identity*, determines an *instance of ITM (ITI)*. A *session s* of protocol π (or (π, s)) is simply the set of ITIs with code π and session ID s. As discussed above, ITIs can be dynamically generated, which allows for dynamic generation of protocol sessions. An *extended session* (π, s) is the transitive closure of the invocation relation starting from the ITIs in session (π, s). ITIs in session (π, s) are also called the *main parties* of s; ITIs in extended session (π, s) (but not in s) are also called the *sub-parties* of (π, s).

To provide some form of "separation" among protocol sessions, we require that the protocol be *subroutine respecting*. Roughly speaking, this means that the only messages sent to any ITI in an extended session from the outside are input messages to some main party, and the only messages sent from any ITI in an extended session to an *existing* outside ITI are subroutine-output messages from some main party.

Compatibility requirements. In the simplified framework, we assume that the identities of all protocol machines are public. However, in the dynamic setting where new ITIs can be invoked by existing ITIs (who also picks the identities of the new ITIs), this is not necessarily the case, causing potential difficulties in simulation (e.g., the simulator in the proof of Theorem 2 needs to check if there is a machine in protocol ϕ with identity ID). To resolve this, we only consider protocols which are *identity exposing*; that is, each session s of protocol π provides to the adversary an interface in which, given an extended identity α, answers with a bit indicating whether α is the extended identity of some ITI in extended session (π, s).

All the modifications to the simplified UC framework discussed above are orthogonal to our UC-AGM framework, so our treatment naturally extends to the full UC framework. The only exception is the running time, since our framework requires the environment and the adversary to additionally send (or output) the algebraic representations of group elements. However, notice that at any state of protocol execution, the length of algebraic representations sent/output by the environment/adversary cannot exceed the overall import it receives so far multiplies the length of the message (without the algebraic representations) it outputs this time. Therefore, if we replace all polynomials $T(\cdot)$ upper-bounding the running time with $T(\cdot)^2$ (and let the environment assign imports appropriately), then the conclusions in the simple framework can be transferred to the full framework.

B.3   Proof of Theorem 1

Proof. For the "only if" direction, simply take Definition 2 and let $\mathcal{A}$ be the $(\mathcal{G}, \pi)$-algebraically dummy adversary (which is clearly efficient assuming that the environment and all protocol machines are efficient).

---

[11] Formally, these rules are enforced via a *control function*, which on input an execution up-to-date, outputs a bit indicating allow/disallow. If the output is 1 then the message is delivered; otherwise the environment is activated. The control function has similar effects to those of the communication sets in the simplified model.

For the "if" direction, let $\mathcal{D}$ be the $(\mathcal{G}, \pi)$-algebraically dummy adversary. Since π $\mathcal{G}$-AGM emulates ϕ w.r.t. the dummy adversary, we know that there is an efficient simulator $\mathcal{S}_{\mathcal{D}}$ such that $(\mathcal{E}', \mathcal{S}_{\mathcal{D}})$ is $(\mathcal{G}, \phi)$-algebraic, and

$$\mathrm{EXEC}_{\phi, \mathcal{S}_{\mathcal{D}}, \mathcal{E}'} \approx \mathrm{EXEC}_{\pi, \mathcal{D}, \mathcal{E}'}$$

for any efficient $(\mathcal{G}, \pi)$-algebraic environment $\mathcal{E}'$.

Construction of simulator $\mathcal{S}$. Given an efficient adversary $\mathcal{A}$ (in π's execution), we construct a simulator $\mathcal{S}$ (in ϕ's execution) which simulates $\mathcal{E}$'s view for any efficient environment $\mathcal{E}$ such that $(\mathcal{E}, \mathcal{A})$ is $(\mathcal{G}, \pi)$-algebraic. $\mathcal{S}$ simulates instances of both $\mathcal{A}$ and $\mathcal{S}_{\mathcal{D}}$, and essentially "combines" them. (Recall that both $\mathcal{A}$ and $\mathcal{S}_{\mathcal{D}}$ communicate with the environment and protocol parties, and $\mathcal{S}$ needs to simulate all these interfaces.) Concretely, $\mathcal{S}$ works as follows:

1. On message (input, m_0) from identity 0 (i.e., from $\mathcal{E}$), $\mathcal{S}$ (locally) activates $\mathcal{A}$ with message (input, m_0).
2. When $\mathcal{A}$ completes this activation,
   (a) If $\mathcal{A}$ does not generate any outgoing message, then $\mathcal{S}$ jumps to step 1.
   (b) If $\mathcal{A}$ generates an outgoing message $(\mathsf{backdoor}, m_1)$ to identity 0 (i.e., to the environment), then $\mathcal{S}$ sends $(\mathsf{backdoor}, m_1)$ to identity 0 (i.e., to $\mathcal{E}$) and jumps to step 1.
   (c) If $\mathcal{A}$ generates an outgoing message $(\mathsf{backdoor}, m_1)$ to some identity $ID \neq 0$ (i.e., to a protocol machine), then $\mathcal{S}$ extracts Λ_1 as the concatenation of the algebraic representations of all group elements in m_1, from either $\mathcal{A}$'s algebraic tape or messages previously sent to $\mathcal{A}$. Then $\mathcal{S}$ (locally) activates $\mathcal{S}_{\mathcal{D}}$ with message $(\mathsf{input}, (ID, m_1 \| \Lambda_1))$.
3. When $\mathcal{S}_{\mathcal{D}}$ completes this activation,
   (a) If $\mathcal{S}_{\mathcal{D}}$ generates an outgoing message $(\mathsf{backdoor}, (ID, m_2))$ to identity 0 (i.e., to the environment), then $\mathcal{S}$ sends $(\mathsf{backdoor}, (ID, m_2))$ to identity 0 and jumps to step 1. (Note that $\mathcal{A}$ is "bypassed" in this case.)
   (b) If $\mathcal{S}_{\mathcal{D}}$ generates an outgoing message $(\mathsf{backdoor}, m_2)$ to some identity $ID \neq 0$ (i.e., to a protocol machine), then $\mathcal{S}$ extracts Λ_2 as the concatenation of the algebraic representations of all group elements in m_2, from either $\mathcal{S}_{\mathcal{D}}$'s algebraic tape or messages previously sent to $\mathcal{S}_{\mathcal{D}}$. Then $\mathcal{S}$ writes Λ_2 on its own algebraic tape and sends $(\mathsf{backdoor}, m_2)$ to identity ID.
4. On message $(\mathsf{backdoor}, m_3)$ from identity $ID \neq 0$ (i.e., from a protocol machine), $\mathcal{S}$ activates $\mathcal{S}_{\mathcal{D}}$ with message $(\mathsf{backdoor}, m_3)$.
5. When $\mathcal{S}_{\mathcal{D}}$ completes this activation,
   (a) If $\mathcal{S}_{\mathcal{D}}$ generates an outgoing message $(\mathsf{backdoor}, (ID, m_2))$ to identity 0, then $\mathcal{S}$ activates $\mathcal{A}$ with message $(\mathsf{backdoor}, (ID, m_2))$ and jumps to step 2.
   (b) If $\mathcal{S}_{\mathcal{D}}$ generates an outgoing message $(\mathsf{backdoor}, m_2)$ to some identity $ID \neq 0$ (i.e., to a protocol machine), then $\mathcal{S}$ extracts Λ_2 as the concatenation of the algebraic representations of all group elements in m_2, from either $\mathcal{S}_{\mathcal{D}}$'s algebraic tape or messages previously sent to $\mathcal{S}_{\mathcal{D}}$. Then $\mathcal{S}$ writes Λ_2 on its own algebraic tape and sends $(\mathsf{backdoor}, m_2)$ to identity ID.
   (Note that the only difference with step 3 is that in case (a) $\mathcal{A}$ is not "bypassed.")

Analysis of simulator $\mathcal{S}$. It is straightforward to see that if $\mathcal{A}$ and $\mathcal{S}_{\mathcal{D}}$ are both efficient, then $\mathcal{S}$ is also efficient. We now show that $(\mathcal{E}, \mathcal{S})$ is $(\mathcal{G}, \phi)$-algebraic in two steps:

– Recall that while interacting with $\mathcal{S}_{\mathcal{D}}$, $\mathcal{S}$ generates the interfaces of both the environment and protocol parties in the execution of ϕ. Let $\mathcal{E}'$ denote such environment in $\mathcal{S}_{\mathcal{D}}$'s view; we show that $\mathcal{E}'$ *is $(\mathcal{G}, \pi)$-algebraic.* $\mathcal{E}'$ sends an input message to identity 0 (i.e., the adversary) in step 2(c) only (let m_1 be the message payload), in which case an algebraic representation Λ_1 is attached to m_1. Such an algebraic representation can be extracted because $(\mathcal{E}, \mathcal{A})$ is $(\mathcal{G}, \pi)$-algebraic, so (1) if Λ_1 is on $\mathcal{A}$'s algebraic tape, then it can be read by $\mathcal{S}$; (2) otherwise Λ_1 must have been previously sent to $\mathcal{A}$ by $\mathcal{S}$ (who simulates $\mathcal{E}$ in $\mathcal{A}$'s view), so $\mathcal{S}$ can extract Λ_1 from its own transcript.

29

– Since $\mathcal{E}'$ is $(\mathcal{G}, \pi)$-algebraic, by definition of $\mathcal{S}_\mathcal{D}$, we have that $(\mathcal{E}', \mathcal{S}_\mathcal{D})$ is $(\mathcal{G}, \phi)$-algebraic. Now we claim that $(\mathcal{E}, \mathcal{S})$ *is* $(\mathcal{G}, \phi)$-*algebraic*. $\mathcal{S}$ sends (backdoor messages to identity $ID \neq 1$ in steps 3(b) and 5(b) only (let m_2 be the message payload); in both cases $\mathcal{S}$ writes an algebraic representation Λ_2 on its algebraic tape. Such an algebraic representation can be extracted because (1) if Λ_2 is on $\mathcal{S}_\mathcal{D}$'s algebraic tape, then it can be read by $\mathcal{S}$; (2) otherwise Λ_2 must have been previously sent to $\mathcal{S}_\mathcal{D}$ by $\mathcal{E}'$, which in turn is part of $\mathcal{S}$, so $\mathcal{S}$ can extract Λ_2 from its own transcript.

Next we show that $\mathcal{S}$ generates an indistinguishable view for $\mathcal{E}$. We construct another environment $\mathcal{E}_\mathcal{D}$, which simulates instances of both $\mathcal{E}$ and $\mathcal{A}$, and essentially "combines" them. Concretely, $\mathcal{E}_\mathcal{D}$, on initial input z, activates $\mathcal{E}$ with input z. Then $\mathcal{E}_\mathcal{D}$ works as follows:

1. When $\mathcal{E}$ completes this activation,
   (a) If $\mathcal{E}$ halts with some output, then $\mathcal{E}_\mathcal{D}$ also halts with the same output.
   (b) If $\mathcal{E}$ generates an outgoing message (input, m_0) to some identity $ID \neq 0$ (i.e., to a protocol machine), then $\mathcal{E}_\mathcal{D}$ sends (input, m_0) to identity ID.
   (c) If $\mathcal{E}$ generates an outgoing message (input, m_0) to identity 1 (i.e., to the adversary), then $\mathcal{E}_\mathcal{D}$ activates $\mathcal{A}$ with message (input, m_0).
2. When $\mathcal{A}$ completes this activation,
   (a) If $\mathcal{A}$ does not generate any outgoing message, then $\mathcal{E}_\mathcal{D}$ jumps to step 1.
   (b) If $\mathcal{A}$ generates an outgoing message $(\mathsf{backdoor}, m_1)$ to identity 0 (i.e., to the environment), then $\mathcal{E}_\mathcal{D}$ enters a special state called bypass.
   (c) If $\mathcal{A}$ generates an outgoing message $(\mathsf{backdoor}, m_1)$ to some identity $ID \neq 0$ (i.e., to a protocol machine), then $\mathcal{E}_\mathcal{D}$ extracts Λ as the concatenation of the algebraic representations of all group elements in m_1, from either $\mathcal{A}$'s algebraic tape or messages previously sent from $\mathcal{E}$ to $\mathcal{A}$. Then $\mathcal{E}_\mathcal{D}$ sends $(\mathsf{input}, (ID, m_1 \| \Lambda))$ to identity 1 (i.e., to the adversary).[12]
3. On message $(\mathsf{backdoor}, (ID, m_2))$ from identity 1 (i.e., from the adversary),
   (a) If $\mathcal{E}_\mathcal{D}$ is in the bypass state, then it exits this state, activates $\mathcal{E}$ with message $(\mathsf{subroutine\text{-}output}, (ID, m_2))$ (as from identity 1), and jumps to step 1.
   (b) Otherwise $\mathcal{E}_\mathcal{D}$ activates $\mathcal{A}$ with message (input, m_2) (as from identity ID) and jumps to step 2, except that in case 2(b) $\mathcal{E}_\mathcal{D}$ activates $\mathcal{E}$ with message $(\mathsf{backdoor}, m_1)$ (rather than entering the bypass state).
4. On message $(\mathsf{subroutine\text{-}output}, m_3)$ from identity $ID \neq 1$ (i.e., from a protocol machine), $\mathcal{E}_\mathcal{D}$ activates $\mathcal{E}$ with message $(\mathsf{subroutine\text{-}output}, m_3)$ and jumps to step 1.

It is straightforward to see that if $\mathcal{E}$ and $\mathcal{A}$ are both efficient, then $\mathcal{E}_\mathcal{D}$ is also efficient. Also, $\mathcal{E}_\mathcal{D}$ perfectly simulates an instance of $\mathcal{A}$ in π's execution, and an instance of $\mathcal{S}$ in ϕ's execution, i.e.,

$$\mathrm{EXEC}_{\pi, \mathcal{D}, \mathcal{E}_\mathcal{D}} = \mathrm{EXEC}_{\pi, \mathcal{A}, \mathcal{E}},$$

and

$$\mathrm{EXEC}_{\phi, \mathcal{S}_\mathcal{D}, \mathcal{E}_\mathcal{D}} = \mathrm{EXEC}_{\phi, \mathcal{S}, \mathcal{E}}.$$

Next we claim that if $(\mathcal{E}, \mathcal{A})$ is $(\mathcal{G}, \pi)$-algebraic, then $\mathcal{E}_\mathcal{D}$ is also $(\mathcal{G}, \pi)$-algebraic. According to the description of $\mathcal{E}_\mathcal{D}$ above, $\mathcal{E}_\mathcal{D}$ sends input messages to identity 1 in step 2(c) only (let m_1 be the message payload), in which case an algebraic representation Λ is attached to m_1. Such an algebraic representation can be extracted because $(\mathcal{E}, \mathcal{A})$ is $(\mathcal{G}, \pi)$-algebraic, so (1) if Λ is on $\mathcal{A}$'s algebraic tape, then it can be read by $\mathcal{E}_\mathcal{D}$; (2) otherwise Λ must have been previously sent to $\mathcal{A}$ by $\mathcal{E}$, so $\mathcal{E}_\mathcal{D}$, who runs both $\mathcal{E}$ and $\mathcal{A}$, can extract Λ from the message sent from $\mathcal{E}$ to $\mathcal{A}$.

Since $\mathcal{E}_\mathcal{D}$ is both efficient and algebraic, by the definition of $\mathcal{S}_\mathcal{D}$, we have that

$$\mathrm{EXEC}_{\phi, \mathcal{S}_\mathcal{D}, \mathcal{E}_\mathcal{D}} \approx \mathrm{EXEC}_{\pi, \mathcal{D}, \mathcal{E}_\mathcal{D}}.$$

Combining the three results above, we conclude that

$$\mathrm{EXEC}_{\phi, \mathcal{S}, \mathcal{E}} \approx \mathrm{EXEC}_{\pi, \mathcal{A}, \mathcal{E}},$$

completing the proof.   □

---

[12] Note that this adversary is not $\mathcal{A}$: both this adversary and $\mathcal{E}_\mathcal{D}$ are machines in protocol execution (of π or ϕ), whereas $\mathcal{E}_\mathcal{D}$ runs $\mathcal{A}$ as a subroutine. In other words, from $\mathcal{E}_\mathcal{D}$'s point of view, this adversary is "external," whereas $\mathcal{A}$ is "internal."

C Detailed Proof for SPAKE2 (Theorem 6)

Simulator $\mathcal{S}$ is shown (again) in Fig. 10, side by side with the code for uncorrupted parties in the real-world. Recall that, whenever the dummy adversary is instructed to deliver a group element to an uncorrupted party, it will output in its auxiliary tape the algebraic representation of that element with respect to group elements appear in the view of the environment. In this case, the bases for such representations include $\mathbf{M}$, $\mathbf{N}$ and any messages $\mathbf{X}^\star$ or $\mathbf{Y}^\star$ produced by an uncorrupted party.

Simulation strategy. The simulator generates all messages of uncorrupted parties by simpling raising either $\mathbf{M}$ or $\mathbf{N}$ to a random exponent. It does so because it does not know the corresponding password. The distribution of such messages is identical to those produced by honest parties in the real world, which are of the form $g^x\mathbf{M}^{\mathsf{pw}}$ or $g^y\mathbf{N}^{\mathsf{pw}}$. The simulator then keeps track of whether the adversary is launching a passive attack or an active attack, where the former means that there exists another simulated session with a consistent view. All passively attacked sessions are not interrupted by the simulator, which means that $\mathcal{F}_{\mathsf{pwKE}}$ will choose independent keys at the associated dummy parties' outputs. For actively attacked sessions, the simulator checks if the delivered message was constructed as per the protocol and, if so, it extracts the password. All malformed messages cause the simulator to interrupt the session in the functionality by calling TestPw with $\mathsf{pw} =\perp$. For well formed messages, the simulator queries TestPw on the extracted password and, if the password is correct, computes the correct key: this is possible because, even though the simulator does not know the correct exponent implicit in the simulated honest party's state, it knows the algebraic decomposition of the delivered message and this is well formed (this means it can compute the key as the adversary would). If the password is incorrect, the simulator generates a totally random key (this is ignored by the functionality if there are no corrupt parties involved in the session, but it is relevant otherwise as we discuss below). The simulation is perfect for all sessions with well-formed messages and correct password guesses. It looks perfect for all other sessions, unless the attacker can query the random oracle on the group element that such a session would compute in the real world. Our proof shows that any such query can, with overwhelming probability, be used to solve the SqDH problem. Two important observations for the proof: i. the simulator never uses the random exponents it generates for the honest party messages to perform any computation; and ii. the simulator never constructs any group element for which it cannot provide an algebraic decomposition to bases g, $\mathbf{M}$ and $\mathbf{N}$. The second observation guarantees that our simulator is an algebraic adversary as required by the composition theorem in Section 2.

Corrupt parties. Fig. 10 does not show explicitly the simulator's handling of sessions involving corrupt parties. In this case, the environment tells the simulator what the corrupt party should be doing, and the simulator does not keep the state of the corrupt party. Moreover, any group elements transmitted by the corrupt party come with their algebraic decomposition as above.

In particular, this means that any output key produced by a corrupt party can be easily fixed by the simulator at the ideal functionality output, since its intended real-world output will be explicitly provided by the environment. More subtle is the case of sessions accepted by uncorrupted parties when interacting with corrupt parties: in this case the simulator must also fix the output of the uncorrupted party via the ideal functionality consistently with the real world.

Our simulator is structured to handle this case identically to the setting where the uncorrupted party is actively attacked while interacting with another uncorrupted party:

- such a session will never be interpreted as passively attacked by the simulator (as it is not controlling the corrupt party state), so it will always test the password;
- if the password is correctly guessed, our simulator programs the functionality with the correct key;
- otherwise, it programs a completely random key (mimicking actively attacked sessions);
- in both cases the functionality outputs what the simulator provides because the session is corrupted.

The proof below covers this scenario as a particular case.

Proof of simulator correctness. From this point on we consider only interactions involving uncorrupted parties. The first observation we make is that the distribution of the protocol messages produced by the

UNCORRUPTED PARTIES	SIMULATOR $\mathcal{S}$
proc INITIALIZE( ) $\overline{\text{Get CRS}=(\mathbf{M},\mathbf{N})}$	proc INITIALIZE( ) $\overline{\text{Get CRS}=(\mathbf{M},\mathbf{N})}$
On input (NewSession, $sid, P, P', \mathsf{pw}, \text{role}$) from $\mathcal{E}$ $\overline{\text{At most one instance for a given } sid \text{ at } P.}$ If role = init $\quad x \twoheadleftarrow \mathbb{Z}_q; \mathbf{X}^{\star} \leftarrow g^x \cdot \mathbf{M}^{\mathsf{pw}}$ $\quad \pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}^{\star}, \perp), \perp, \mathsf{pw}, \text{init})$ $\quad$ Send SENDINIT$(P, P', sid, \mathbf{X}^{\star})$ to $\mathcal{E}$ via $\mathcal{D}$ Else $\quad y \twoheadleftarrow \mathbb{Z}_q; \mathbf{Y}^{\star} \leftarrow g^y \cdot \mathbf{N}^{\mathsf{pw}}$ $\quad \pi_P^{sid} \leftarrow (y, (P', P, sid, \perp, \mathbf{Y}^{\star}), \perp, \mathsf{pw}, \text{resp})$	On input (NewSession, sid, P, P', role) from $\mathcal{F}_{\mathsf{pwKE}}$ $\overline{\text{If } \neg(\pi_P^{sid} = \perp) \text{ discard input.}}$ If role = init $\quad x \twoheadleftarrow \mathbb{Z}_q; \mathbf{X}^{\star} \leftarrow \mathbf{M}^x$ $\quad \pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}^{\star}, \perp), \perp, \text{init})$ $\quad$ Send SENDINIT$(P, P', sid, \mathbf{X}^{\star})$ to $\mathcal{E}$ Else $\quad y \twoheadleftarrow \mathbb{Z}_q; \mathbf{Y}^{\star} \leftarrow \mathbf{N}^y$ $\quad \pi_P^{sid} \leftarrow (y, (P', P, sid, \perp, \mathbf{Y}^{\star}), \perp, \text{resp})$
On message SENDINIT$(P, P', sid, \mathbf{X}^{\star})$ from $\mathcal{E}$ via $\mathcal{D}$ $\overline{\text{Ignore if } \pi_{P'}^{sid} \neq (y, (P, P', sid, \perp, \mathbf{Y}^{\star}), \perp, \mathsf{pw}, \text{resp})}$ (A unique $\pi_{P'}^{sid}$ satisfies the above check) $X \leftarrow \mathbf{X}^{\star} / \mathbf{M}^{\mathsf{pw}}$ $K \leftarrow \text{H}(P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}, \mathsf{pw}, X^y)$ Send SENDRESP$(P', P, sid, \mathbf{Y}^{\star})$ to $\mathcal{E}$ via $\mathcal{D}$ $\pi_{P'}^{sid} \leftarrow (y, (P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}), K, \text{resp})$ Output K	On message SENDINIT$(P, P', sid, (\mathbf{X}^{\star}, \mathsf{alg}))$ from $\mathcal{E}$ $\overline{\text{Ignore if } \pi_{P'}^{sid} \neq (y, (P, P', sid, \perp, \mathbf{Y}^{\star}), \perp, \text{resp})}$ (A unique $\pi_{P'}^{sid}$ satisfies the above check) $K \twoheadleftarrow \mathcal{K}$ (Can't interrupt passive sessions so $\mathcal{F}$ sets $=K$ at output) If $\pi_P^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^{\star}, \perp), \perp, \text{init})$ Jump to COMPLETE (First check whether $\mathbf{X}^{\star}$ was constructed as per protocol) If $\mathsf{alg} = [(g, x); (\mathbf{M}, \mathsf{pw})]$ $\quad$ Query (TestPwd, sid, P', pw) to $\mathcal{F}_{\mathsf{pwKE}}$ $\quad$ If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" $\quad\quad Y \leftarrow \mathbf{Y}^{\star} / \mathbf{N}^{\mathsf{pw}}$ $\quad\quad K \leftarrow \text{H}(P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}, \mathsf{pw}, Y^x)$ (Interrupt all other sessions so independent key is set) Else Query (TestPwd, $sid, P', \perp$) to $\mathcal{F}_{\mathsf{pwKE}}$ COMPLETE: Send SENDRESP$(P', P, sid, \mathbf{Y}^{\star})$ to $\mathcal{E}$ $\quad\quad\quad \pi_{P'}^{sid} \leftarrow (y, (P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}), K, \text{resp})$ $\quad\quad\quad$ Query (NewKey, sid, P', K) to $\mathcal{F}_{\mathsf{pwKE}}$
On message SENDRESP$(P', P, sid, \mathbf{Y}^{\star})$ from $\mathcal{E}$ via $\mathcal{D}$ $\overline{\text{Ignore if } \pi_P^{sid} \neq (x, (P, P', sid, \mathbf{X}^{\star}, \perp), \perp, \mathsf{pw}, \text{init})}$ (A unique π_P^{sid} satisfies the above check) $Y \leftarrow \mathbf{Y}^{\star} / \mathbf{N}^{\mathsf{pw}}$ $K \leftarrow \text{H}(P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}, \mathsf{pw}, Y^x)$ $\pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}), K, \text{init})$ Output K	On message SENDRESP$(P', P, sid, (\mathbf{Y}^{\star}, \mathsf{alg}))$ from $\mathcal{E}$ $\overline{\text{Ignore if } \pi_P^{sid} \neq (x, (P, P', sid, \mathbf{X}^{\star}, \perp), \perp, \text{init})}$ (A unique π_P^{sid} satisfies the above check) $K \twoheadleftarrow \mathcal{K}$ (Can't interrupt passive sessions so $\mathcal{F}$ sets $=K$ at output) If $\pi_{P'}^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}), \,\cdot\,, \text{resp})$ Jump to COMPLETE (First check whether $\mathbf{Y}^{\star}$ was constructed as per protocol) If $\mathsf{alg} = [(g, y); (\mathbf{N}, \mathsf{pw})]$ $\quad$ Query (TestPwd, sid, P, pw) to $\mathcal{F}_{\mathsf{pwKE}}$ $\quad$ If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" $\quad\quad X \leftarrow \mathbf{X}^{\star} / \mathbf{M}^{\mathsf{pw}}$ $\quad\quad K \leftarrow \text{H}(P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}, \mathsf{pw}, X^y)$ (Interrupt all other sessions so independent key is set) Else Query (TestPwd, $sid, P, \perp$) to $\mathcal{F}_{\mathsf{pwKE}}$ COMPLETE: $\pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}^{\star}, \mathbf{Y}^{\star}), K, \text{init})$ $\quad\quad\quad$ Query (NewKey, sid, P, K) to $\mathcal{F}_{\mathsf{pwKE}}$

Fig. 10. The operation of uncorrupted SPAKE2 parties in the real world (left) and the corresponding simulator (right). The simulator does not need to observe adversarial random oracle queries nor program either of the random oracle or the CRS.

simulator is identical to that occurring in the real world, even though they are constructed differently. It therefore remains to prove that the outputs of the ideal functionality match the distribution of the parties' outputs in the real world. We observe that the real and ideal worlds are identical until bad, where *bad* is defined as the event that a secret key that is selected uniformly at random by the functionality at the output of an uncorrupted party is inconsistent with the answer given by H to the adversary. This is because in all other cases the simulator programs the output of the ideal functionality consistently with the real world. This means formally that, for $\epsilon = \Pr[\text{EXEC}_{\text{pwKE},\mathcal{S},\mathcal{E}} \Rightarrow \text{bad}]$, we have

$$\text{EXEC}_{\text{pwKE},\mathcal{S},\mathcal{E}} \approx_\epsilon \text{EXEC}_{\text{SPAKE2},\mathcal{D},\mathcal{E}}$$

More precisely, we define event bad as the existence within the set of queries placed by $\mathcal{E}$ to the random oracle of a query $(sid, P, P', \mathbf{X}^\star, \mathbf{Y}^\star, \text{pw}, \mathbf{Z})$ that is *consistent* with the trace of a session at an uncorrupted party, which accepted after a passive attack or after an active attack where the simulator did not place a correct TestPw query. We define now these conditions formally.

We say a random oracle query $(sid, P, P', \mathbf{X}^\star, \mathbf{Y}^\star, \text{pw}, \mathbf{Z})$ is *consistent* with an initiator session π_P^{sid} if this instance was created following a NewSession query by $\mathcal{E}$ using pw and $\pi_P^{sid} = (\cdot, (P, P', \mathbf{X}^\star, \mathbf{Y}^\star), \cdot, \text{init})$. Similarly, the condition for responder session $\pi_{P'}^{sid}$ is $\pi_{P'}^{sid} = (\cdot, (P, P', \mathbf{X}^\star, \mathbf{Y}^\star), \cdot, \text{resp})$. Note that the order of party identities in the trace determines the role of the party.

We say an initiator session π_P^{sid} *accepted after a passive attack* if it completed following a SENDRESP(P', $P, sid, (\mathbf{Y}^\star, \text{alg})$) message from $\mathcal{E}$, when $\pi_{P'}^{sid} = (\cdot, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), \cdot, \text{resp})$. We say a responder session $\pi_{P'}^{sid}$ *accepted after a passive attack* if it completed following a SENDINIT($P, P', sid, (\mathbf{X}^\star, \text{alg})$) message from $\mathcal{E}$, when $\pi_P^{sid} = (\cdot, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), \cdot, \text{init})$. All other sessions are considered to be *under active attack*.

We reduce the probability of bad to SqDH. Intuitively, our reduction embeds the SqDH challenge $\mathbf{A}$ in the global parameters $(\mathbf{M} = \mathbf{A}, \mathbf{N} = \mathbf{A}^\delta)$ for δ sampled uniformly at random from $\mathbb{Z}_q^\star$ (this accounts for the $1/q$ term in the theorem statement). Suppose bad is first set for a random oracle entry that is consistent with a session accepted by an initiator session. The attacker delivered a message $\mathbf{Y}^\star$ and an algebraic representation that we can see as $[(a, g), (b, \mathbf{M}), (c, \mathbf{N})]$. The reduction can transform this algebraic representation into $[(a, g), (b + \delta c, \mathbf{A})]$. This means that any problematic random oracle query will include a group element of the form

$$\mathbf{Z} = \text{CDH}(\mathbf{A}^{x-\text{pw}}, g^a \mathbf{A}^{b+\delta(c-\text{pw})}) = \text{CDH}(\mathbf{A}^{x-\text{pw}}, g^a) \cdot \text{CDH}(\mathbf{A}^{x-\text{pw}}, \mathbf{A}^{b+\delta(c-\text{pw})})$$

where x was chosen by the reduction following the simulator code. Since the reduction can compute the first factor, we can recover

$$\text{CDH}(\mathbf{A}, \mathbf{A})^{(x-\text{pw})(b+\delta(c-\text{pw}))}$$

which means that the required SqDH solution can be recovered when $(x - \text{pw})(b + \delta(c - \text{pw})) \neq 0$. The case of responders is similar, but we can only recover the SqDH result provided $\delta(y - \text{pw})(b + \delta c - \text{pw}) \neq 0$.

We therefore split the bad event into the following cases:

- bad_I: bad is first detected on an initiator session but $(x - \text{pw})(b + \delta(c - \text{pw})) = 0$
- bad_R: bad is first detected on a responder session but $\delta(y - \text{pw})(b + \delta c - \text{pw}) = 0$
- $\text{bad}_1 = \text{bad}_I \vee \text{bad}_R$
- $\text{bad}_2 = \text{bad} \wedge \neg\text{bad}_1$

Clearly we have

$$\Pr[\text{EXEC}_{\text{pwKE},\mathcal{S},\mathcal{E}} \Rightarrow \text{bad}] = \Pr[\text{EXEC}_{\text{pwKE},\mathcal{S},\mathcal{E}} \Rightarrow \text{bad}_1] + \Pr[\text{EXEC}_{\text{pwKE},\mathcal{S},\mathcal{E}} \Rightarrow \text{bad}_2].$$

Bounding bad_1: Reductions to Discrete Log We analyze bad_1 by splitting it into five disjoint cases:

- $\text{bad}_{I,R}^1$: bad_1 first occurs because the value x (resp. y) sampled in an initiator (resp. responder) session matches the password pw that was included by $\mathcal{E}$ in the NEWSESSION query that created that session. This event is easy to bound, as pw is fixed when the value is sampled, and it accounts for the q_S/q term in the theorem statement.

- bad_I^2: bad_I first occurs because $\mathsf{bad}_{I,R}^1$ did not occur and $c = \mathsf{pw}$. In the case of passive attacks, this means that the environment extracted the value $y = c$ sampled by the simulator in the matching session. In active attacks, this can *never* occur. To see this, note that if bad occurs and $c = \mathsf{pw}$, then it must be the case that $b \neq 0$, as otherwise the simulator would have made a correct password guess and computed the correct key. But then, the condition $(b + \delta(c - \mathsf{pw})) = 0$ cannot be satisfied.
- bad_R^2: bad_R first occurs because none of the previous variants occurred and $b = \mathsf{pw}$. In the case of passive attacks, this means that the environment extracted the value $x = b$ sampled by the simulator in the matching session. In active attacks, this can *never* occur. To see this, note that if bad occurs and $b = \mathsf{pw}$, then it must be the case that $c \neq 0$, as otherwise the simulator would have made a correct password guess and computed the correct key. But then, the condition $(b + \delta c - \mathsf{pw}) = 0$ cannot be satisfied because we use $\delta \neq 0$.
- bad_I^3: bad_I first occurs because none of the previous variants occurred and $(b + \delta(c - \mathsf{pw})) = 0$. This event can only occur in actively attacked sessions, as passively attacked sessions all have $b = 0$ and bad_1^2 has not occurred.
- bad_R^3: bad_R first occurs because none of the previous variants occurred and $(b + \delta c - \mathsf{pw}) = 0$. This event can only occur in actively attacked sessions, as passively attacked sessions all have $c = 0$ and bad_1^3 has not occurred.

It is straightforward to bound $\mathsf{bad}_I^2 \vee \mathsf{bad}_R^2$ by reducing the probability of these events occurring to the discrete logarithm problem. Let us without loss of generality assume we get q_S discrete logarithm challenges and use them as the messages $\mathbf{X}^\star$ and $\mathbf{Y}^\star$ produced by the simulator. The reduction defines $(\mathbf{M} = g^\lambda, \mathbf{N} = g^\delta)$, for random λ and δ. Then, if bad_I^2 occurs, we obtain the discrete logarithm c of some $\mathbf{Y}^\star$ wrp to $\mathbf{N}$, and we can compute the discrete logarithm to the base g as $\lambda \cdot c$. The reasoning for the responder side is identical. We call this reduction $\mathcal{B}_1^1$ in the theorem statement.

We now consider the last two cases. We note that bad_I^3 occurs if and only if $\delta = b/(\mathsf{pw} - c)$, as we know from bad_I^2 that $c \neq \mathsf{pw}$. Similarly, bad_R^3 occurs if and only if $c \neq 0 \wedge \delta = (\mathsf{pw} - b)/c$, as we know from bad_R^2 that $b \neq \mathsf{pw}$. These two events can be jointly reduced to the Discrete Logarithm problem by setting $(\mathbf{M} = g^\lambda, \mathbf{N} = \mathbf{R})$ where $\mathbf{R}$ is the discrete logarithm challenge and λ is sampled uniformly at random. Indeed, if either of these events occurs, then the reduction can recover δ, and consequently the discrete logarithm $\lambda \cdot \delta$ (recall we have defined $\mathbf{N} = \mathbf{M}^\delta$). We call this reduction $\mathcal{B}_1^2$ in the theorem statement.

Bounding bad_2: reduction to SqDH. Fig. 11 gives the detailed reduction to SqDH. The reduction, as presented, would fail in the cases we have excluded with bad_1, but here we can assume that bad_1 has not occurred. The reduction follows the strategy we outlined above. When the experiment terminates, it samples a random oracle query uniformly at random[13] and looks for a consistent session. It could find one or two consistent sessions, where the latter case corresponds to a passive attack with matching passwords on both sides. In any case, it computes a candidate SqDH value using the appropriate initiator or responder-side formula we described above. If the randomly selected random oracle entry was the first to cause the bad event, the algorithm solves SqDH. This accounts for the q_H multiplicative loss in the theorem statement. $\qquad\square$

D Detailed Proof for CPace (Theorem 7)

Simulator $\mathcal{S}$ is shown (again) in Fig. 12, side by side with the code for uncorrupted parties in the real-world. Recall that, whenever the dummy adversary is instructed to deliver a group element to an uncorrupted party, it will output in its auxiliary tape the algebraic representation of that element with respect to group elements appear in the view of the environment. In this case the bases for such representations include the outputs of the random oracle H_1 and any messages $\mathbf{X}$ or $\mathbf{Y}$ produced by an uncorrupted party.

Simulation strategy and handling of corrupt parties. The simulation strategy here is identical to that we adopt for the SPAKE2 proof, with the caveat that the simulator must learn the environments' queries to

---

[13] This step could be replaced with a search for a consistent entry using a DDH oracle to the fixed basis $\mathbf{A}$, resulting in a tighter reduction to Strong SqDH where the q_H factor disappears.

REDUCTION $\mathcal{B}_2(\mathbf{A})$

<u>proc Init( )</u>
$\delta \twoheadleftarrow \mathbb{Z}_q^*$
Program $\mathsf{CRS} = (\mathbf{M}, \mathbf{N}) = (\mathbf{A}, \mathbf{A}^\delta)$

<u>proc H($sid, U, S, \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, \mathbf{Z}$))   (non-repeat queries)</u>
$K \twoheadleftarrow \mathcal{K}$; $T[sid, U, S, \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, \mathbf{Z})] \leftarrow K$; return K

<u>If $\mathcal{E}$ causes $\mathcal{F}_{\mathsf{pwKE}}$ to send (NewSession, $sid, P, P', \mathrm{role}$) to $\mathcal{S}$</u>
(Note reduction gets pw in the triggering NEWSESSION query)
If $\neg(\pi_P^{sid} = \perp)$ discard input
If role $=$ init
    $\hat{x} \twoheadleftarrow \mathbb{Z}_q$; $\mathbf{X}^\star \leftarrow \mathbf{M}^{\hat{x}}$
    $\pi_P^{sid} \leftarrow (\hat{x}, (P, P', sid, \mathbf{X}^\star, \perp), \perp, \mathrm{init})$
    Send SENDINIT($P, P', sid, \mathbf{X}^\star$) to $\mathcal{D}$
Else
    $\hat{y} \twoheadleftarrow \mathbb{Z}_q$; $\mathbf{Y}^\star \leftarrow \mathbf{N}^{\hat{y}}$
    $\pi_P^{sid} \leftarrow (\hat{y}, (P', P, sid, \perp, \mathbf{Y}^\star), \perp, \mathrm{resp})$

<u>On message SENDINIT($P, P', sid, (\mathbf{X}^\star, \mathsf{alg})$) from $\mathcal{E}$</u>
If $\pi_{P'}^{sid} \neq (\hat{y}, (P, P', sid, \perp, \mathbf{Y}^\star), \perp, \mathrm{resp})$ discard input
$K \twoheadleftarrow \mathcal{K}$
If $\pi_P^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^\star, \perp), \perp, \mathrm{init})$ Jump to COMPLETE
If $\mathsf{alg} = [(g, x), (\mathbf{M}, \mathsf{pw})]$
    Compute (TestPwd, sid, P', pw) as $\mathcal{F}_{\mathsf{pwKE}}$
    If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" Then $K \leftarrow \mathrm{H}(sid, \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, (\mathbf{Y}^\star / \mathbf{N}^{\mathsf{pw}})^x)$
    Else Compute (TestPwd, $sid, P', \perp$) as $\mathcal{F}_{\mathsf{pwKE}}$
Else Query (TestPwd, $sid, P', \perp$) to $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: Send SENDRESP($P', P, sid, \mathbf{Y}^\star$) to $\mathcal{E}$
        $\pi_{P'}^{sid} \leftarrow (\hat{y}, (P, P', sid, (\mathbf{X}^\star, \mathsf{alg}), \mathbf{Y}^\star), K, \mathrm{resp})$
        Compute (NewKey, sid, P', K) as $\mathcal{F}_{\mathsf{pwKE}}$; send output to $\mathcal{E}$

<u>On message SENDRESP($P', P, sid, (\mathbf{Y}^\star, \mathsf{alg})$) from $\mathcal{E}$</u>
If $\pi_P^{sid} \neq (\hat{x}, (P, P', sid, \mathbf{X}^\star, \perp), \perp, \mathrm{init})$ discard input
$K \twoheadleftarrow \mathcal{K}$
If $\pi_{P'}^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^\star, \mathbf{Y}^\star), \,\cdot\,, \mathrm{resp})$ Jump to COMPLETE
If $\mathsf{alg} = [(g, y), (\mathbf{N}, \mathsf{pw})]$
    Compute (TestPwd, sid, P, pw) as $\mathcal{F}_{\mathsf{pwKE}}$
    If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" Then $K \leftarrow \mathrm{H}(sid, \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, (\mathbf{X}^\star / \mathbf{M}^{\mathsf{pw}})^y)$
Else Compute (TestPwd, $sid, P, \perp$) as $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: $\pi_P^{sid} \leftarrow (\hat{x}, (P, P', sid, \mathbf{X}^\star, (\mathbf{Y}^\star, \mathsf{alg})), K, \mathrm{init})$
        Compute (NewKey, sid, P, K) as $\mathcal{F}_{\mathsf{pwKE}}$; send output to $\mathcal{E}$

<u>On termination</u>
Sample $(sid, P, P', \mathbf{X}^\star, \mathbf{Y}^\star, \mathsf{pw}, \mathbf{Z}) \twoheadleftarrow T$
If $\exists \pi_P^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}^\star, (\mathbf{Y}^\star, \mathsf{alg})), \,\cdot\,, \mathrm{init})$ for which $\mathcal{F}_{\mathsf{pwKE}}$ holds pw:
    Rewrite $\mathsf{alg} = [(a, g), (b + \delta c, \mathbf{A})]$
    Terminate outputting $(\mathbf{Z} / \mathbf{A}^{\alpha(x - \mathsf{pw})})^{\frac{1}{(x - \mathsf{pw})(b + \delta(c - \mathsf{pw}))}}$
If $\exists \pi_{P'}^{sid} = (\,\cdot\,, (P, P', sid, (\mathbf{X}^\star, \mathsf{alg}), \mathbf{Y}^\star), \,\cdot\,, \mathrm{resp})$ for which $\mathcal{F}_{\mathsf{pwKE}}$ holds pw:
    Rewrite $\mathsf{alg} = [(a, g), (b + \delta c, \mathbf{A})]$
    Terminate outputting $(\mathbf{Z} / \mathbf{A}^{\alpha\delta(y - \mathsf{pw})})^{\frac{1}{\delta(y - \mathsf{pw})(b + \delta c - \mathsf{pw})}}$

Fig. 11. Reduction $\mathcal{B}_2$ computes $\mathsf{SqDH}(\mathbf{A})$. T is initially empty. $\mathcal{B}_2$ runs $\mathcal{F}_{\mathsf{pwKE}}$ internally, so it can check password guesses and answer corrupt queries as in the ideal world. It can also check whether random oracle entries are consistent with the session passwords on termination.

UNCORRUPTED PARTIES	SIMULATOR $\mathcal{S}$
On input $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \mathsf{role})$ from $\mathcal{E}$	$\underline{\text{proc } \mathrm{H}_1(sid, P, P', \mathsf{pw}) \qquad \text{(non-repeat queries)}}$
At most one instance for a given sid at P.	$r \twoheadleftarrow \mathbb{Z}_q; \mathbf{G} \leftarrow g^r; T_1[sid, P, P', \mathsf{pw}] \leftarrow \mathbf{G}; \text{return } \mathbf{G}$
If $\mathsf{role} = \mathsf{init}$	Simulator aborts if at any point T_1 is non-injective.
$\quad \mathbf{G} \leftarrow \mathrm{H}_1(sid, P, P', \mathsf{pw})$	
$\quad x \twoheadleftarrow \mathbb{Z}_q; \mathbf{X} \leftarrow \mathbf{G}^x$	On input $(\mathsf{NewSession}, sid, P, P', \mathsf{role})$ from $\mathcal{F}_{\mathsf{pwKE}}$
$\quad \pi_P^{sid} \leftarrow (x, (P, P', sid, \mathbf{X}, \perp), \perp, \mathsf{init})$	If $\neg(\pi_P^{sid} = \perp)$ discard input.
$\quad$ Send $\mathrm{SENDINIT}(P, P', sid, \mathbf{X})$ to $\mathcal{D}$	If $\mathsf{role} = \mathsf{init}$
Else	$\quad \hat{x} \twoheadleftarrow \mathbb{Z}_q^*; \mathbf{X} \leftarrow g^{\hat{x}}$
$\quad \mathbf{G} \leftarrow \mathrm{H}_1(sid, P', P, \mathsf{pw})$	$\quad \pi_P^{sid} \leftarrow (\hat{x}, (P, P', sid, \mathbf{X}, \perp), \perp, \mathsf{init})$
$\quad y \twoheadleftarrow \mathbb{Z}_q; \mathbf{Y} \leftarrow \mathbf{G}^y$	$\quad$ Send $\mathrm{SENDINIT}(P, P', sid, \mathbf{X})$ to $\mathcal{D}$
$\quad \pi_P^{sid} \leftarrow (y, (P', P, sid, \perp, \mathbf{Y}), \perp, \mathsf{resp})$	Else
	$\quad \hat{y} \twoheadleftarrow \mathbb{Z}_q^*; \mathbf{Y} \leftarrow g^{\hat{y}}$
	$\quad \pi_P^{sid} \leftarrow (\hat{y}, (P', P, sid, \perp, \mathbf{Y}), \perp, \mathsf{resp})$
On message $\mathrm{SENDINIT}(P, P', sid, \mathbf{X} \neq \mathbf{1})$ from $\mathcal{E}$ via $\mathcal{D}$	On message $\mathrm{SENDINIT}(P, P', sid, (\mathbf{X}, \mathsf{alg}) \neq \mathbf{1})$ from $\mathcal{E}$ via $\mathcal{D}$
Ignore if $\pi_{P'}^{sid} \neq (y, (P, P', sid, \perp, \mathbf{Y}), \perp, \mathsf{resp})$	Ignore if $\pi_{P'}^{sid} \neq (\hat{y}, (P, P', sid, \perp, \mathbf{Y}), \perp, \mathsf{resp})$
(A unique $\pi_{P'}^{sid}$ satisfies the above check)	(A unique $\pi_{P'}^{sid}$ satisfies the above check)
$K \leftarrow \mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, X^y)$	$K \twoheadleftarrow \mathcal{K}$
Send $\mathrm{SENDRESP}(P', P, sid, \mathbf{Y})$ to $\mathcal{D}$	(Can't interrupt passive sessions so $\mathcal{F}$ sets $= K$ at output)
$\pi_{P'}^{sid} \leftarrow (\perp, (P, P', sid, \mathbf{X}, \mathbf{Y}), K, \mathsf{resp})$	If $\pi_P^{sid} = (\cdot, (P, P', sid, \mathbf{X}, \perp), \perp, \mathsf{init})$ Jump to COMPLETE
Output K	(First check whether $\mathbf{X}$ was constructed as per protocol)
	If $\mathsf{alg} = [(\mathbf{G}, x)] \wedge (sid, P, P', \mathsf{pw}, \mathbf{G}) \in T_1$
	$\quad$ Query $(\mathsf{TestPwd}, sid, P', \mathsf{pw})$ to $\mathcal{F}_{\mathsf{pwKE}}$
	$\quad$ If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess"
	$\quad\quad K \leftarrow \mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Y}^x)$
	(Interrupt all other non-passive sessions.)
	Else Query $(\mathsf{TestPwd}, sid, P', \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$
	COMPLETE: Send $\mathrm{SENDRESP}(P', P, sid, \mathbf{Y})$ to $\mathcal{D}$
	$\quad\quad \pi_{P'}^{sid} \leftarrow (\perp, (P, P', sid, \mathbf{X}, \mathbf{Y}), K, \mathsf{resp})$
	$\quad\quad$ Query $(\mathsf{NewKey}, sid, P', K)$ to $\mathcal{F}_{\mathsf{pwKE}}$
On message $\mathrm{SENDRESP}(P', P, sid, \mathbf{Y} \neq \mathbf{1})$ from $\mathcal{E}$ via $\mathcal{D}$	On message $\mathrm{SENDRESP}(P', P, sid, (\mathbf{Y}, \mathsf{alg}) \neq \mathbf{1})$ from $\mathcal{E}$ via $\mathcal{D}$
Ignore if $\pi_P^{sid} \neq (x, (P, P', sid, \mathbf{X}, \perp), \perp, \mathsf{init})$	Ignore if $\pi_P^{sid} \neq (\hat{x}, (P, P', sid, \mathbf{X}, \perp), \perp, \mathsf{init})$
(A unique π_P^{sid} satisfies the above check)	(A unique π_P^{sid} satisfies the above check)
$K \leftarrow \mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, Y^x)$	$K \twoheadleftarrow \mathcal{K}$
$\pi_P^{sid} \leftarrow (\perp, (P, P', sid, \mathbf{X}, \mathbf{Y}), K, \mathsf{init})$	(Can't interrupt passive sessions so $\mathcal{F}$ sets $= K$ at output)
Output K	If $\pi_{P'}^{sid} = (\cdot, (P, P', sid, \mathbf{X}, \mathbf{Y}), \cdot, \mathsf{resp})$ Jump to COMPLETE
	(First check whether $\mathbf{Y}$ was constructed as per protocol)
	If $\mathsf{alg} = [(\mathbf{G}, y)] \wedge (sid, P, P', \mathsf{pw}, \mathbf{G}) \in T_1$
	$\quad$ Query $(\mathsf{TestPwd}, sid, P, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{pwKE}}$
	$\quad$ If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess"
	$\quad\quad K \leftarrow \mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{X}^y)$
	(Interrupt all other non-passive sessions.)
	Else Query $(\mathsf{TestPwd}, sid, P, \perp)$ to $\mathcal{F}_{\mathsf{pwKE}}$
	COMPLETE: $\pi_P^{sid} \leftarrow (\perp, (P, P', sid, \mathbf{X}, \mathbf{Y}), K, \mathsf{init})$
	$\quad\quad$ Query $(\mathsf{NewKey}, sid, P, K)$ to $\mathcal{F}_{\mathsf{pwKE}}$

Fig. 12. The operation of uncorrupted CPace parties in the real world (left) and the corresponding simulator (right). The simulator needs to observe adversarial random oracle queries on H_1 but not on H_2, and it does not need to program either of the random oracles. T_1 is initially empty.

H_1 in order to extract the password in an active attack. The handling of corrupt parties is also the same. Also here the simulator never generates any group element for which it cannot give an algebraic decomposition with respect to base g, and hence it is an algebraic adversary. The handling of corrupt parties is also the same.

Proof of simulator correctness. From this point on we consider only interactions involving uncorrupted parties. The first observation we make is that the distribution of the protocol messages produced by the simulator is statistically close to that occurring in the real world, even though they are constructed differently. The only difference is that we exclude the case where $\hat{x}$ or $\hat{y}$ are zero. We therefore first modify the real-world so that the distribution of the protocol messages matches that of the ideal world. This explains the statistical term q_S/q in the theorem statement. Moreover, the simulator will abort if at some point it cannot invert outputs of H_1 due to ambiguity in T_1. The probability of abort explains the statistical factor of $q_{H_1}^2/q$ in the theorem statement.

It remains to prove that the outputs of the ideal functionality match the distribution of the parties' outputs in this slightly modified version of the real world. We observe that the real and ideal worlds are now identical until bad, where *bad* is defined as the event that a secret key that is selected uniformly at random by the functionality at the output of an uncorrupted party is inconsistent with the answer given by H_2 to the adversary. This is because in all other cases the simulator programs the output of the ideal functionality consistently with the real world. This means formally that, for $\epsilon = \Pr[\text{EXEC}_{\mathsf{pwKE},\mathcal{S},\mathcal{E}} \Rightarrow \mathsf{bad}]$, we have $\text{EXEC}_{\mathsf{pwKE},\mathcal{S},\mathcal{E}} \approx_\epsilon \text{EXEC}_{\mathsf{CPACE},\mathcal{D},\mathcal{E}}$.

More precisely, we define event bad as the existence within the set of queries placed by $\mathcal{E}$ to H_2 of a query $(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$ that is *consistent* with the trace of a session at an uncorrupted party, which accepted after a passive attack or after an active attack where the simulator did not place a correct TestPw query. We define now these conditions formally.

We say an H_2 query $(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$ is consistent with an initiator session π_P^{sid} if $\pi_P^{sid} = (\,\cdot\,, (\,\cdot\,, \cdot\,, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{init})$. Similarly, the condition for responder session $\pi_{P'}^{sid}$ is $\pi_{P'}^{sid} = (\,\cdot\,, (\,\cdot\,, \cdot\,, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{resp})$. We say an initiator session π_P^{sid} accepted after a passive attack if it completed following a $\text{SENDRESP}(P', P, sid, (\mathbf{Y}, \mathsf{alg}))$ message from $\mathcal{E}$, when $\pi_{P'}^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{resp})$. Responder session $\pi_{P'}^{sid}$ accepted after a passive attack if it completed after a $\text{SENDINIT}(P, P', sid, (\mathbf{X}, \mathsf{alg}))$ message from $\mathcal{E}$, when $\pi_P^{sid} = (\,\cdot\,, (P, P', sid, \mathbf{X}, \mathbf{Y}), \,\cdot\,, \mathsf{init})$. All other sessions are considered to be under active attack. Finally, we say a T_1 entry of the form $(sid, P, P', \mathsf{pw})$ is consistent with an initiator (resp. responder) instance, if that instance was initialized by the environment in a NEWSESSION query with $(sid, P, P', \mathsf{pw}, \mathsf{init})$ (resp. $(sid, P', P, \mathsf{pw}, \mathsf{resp})$).

We bound the probability of bad in the ideal world using a simple sequence of games.

Guessing the RO entries that cause bad. We modify ideal world as follows: sample (ℓ_1, ℓ_2) uniformly at random in $[q_{H_1}] \times [q_{H_2}]$. Then, if bad first occurs due to the i-th H_2 query such that $i \neq \ell_2$, abort. Furthermore, if the offending T_1 entry (i.e., the T_1 unique entry consistent with the session where the bad event was detected) is not the ℓ_1-th one, abort. Clearly, we can still bound the probability of bad in the previous game with the pessimistic bound $q_{H_1} \cdot q_{H_2} \cdot \Pr[\mathsf{bad}]$, where we only check for bad if the experiment has not aborted. We give in Fig. 13 a reduction $\mathcal{B}_{\ell_1, \ell_2}$ that solves the InvCDH problem whenever bad occurs in this modified game.

Final reduction. The reduction strategy is as follows. The generator returned by H_1 for the problematic session associated with the ℓ_2-th password query is programmed to be $\mathbf{A}$, the InvCDH problem instance. All messages generated by uncorrupted parties are generated as $g^{\hat{x}}$ or $g^{\hat{y}}$. All random oracle queries consistent with a session with trace $(\mathbf{X}, \mathbf{Y})$ and generator $\mathbf{A}$ include the key element $\mathbf{Z}$ satisfying the following equation:

$$\mathbf{Z} = \mathbf{X}^{\mathsf{dlog}_{\mathbf{A}}(\mathbf{Y})} = \mathbf{Y}^{\mathsf{dlog}_{\mathbf{A}}(\mathbf{X})} = \mathbf{A}^{(\mathsf{dlog}_{\mathbf{A}}(\mathbf{X}) \cdot \mathsf{dlog}_{\mathbf{A}}(\mathbf{Y}))}$$

Observing that $\mathsf{dlog}_{\mathbf{A}}(\mathbf{X}) = \mathsf{dlog}_g(\mathbf{X})/\mathsf{dlog}_g(\mathbf{A})$ the equation can be re-written as:

$$\mathbf{Z} = g^{\frac{\mathsf{dlog}_g(\mathbf{X}) \cdot \mathsf{dlog}_g(\mathbf{Y})}{\mathsf{dlog}_g \mathbf{A}}}$$

In the simplest case of a passive attack, it is immediate that we recover the solution to the InvCDH problem if $\hat{x} \cdot \hat{y} \neq 0$, which we know to be the case.

Now let us suppose the problematic case occurs with an actively attacked initiator session. Then we know that $\mathbf{Y} = g^\alpha \mathbf{A}^\beta$ and $\alpha \neq 0$; otherwise this would be a correct password guess and the bad event could never have occurred for this session—recall the experiment would have aborted if H_1 did not program $\mathbf{A}$ as the output for the password associated with this session. We can therefore refine the equation above to:

$$\mathbf{Z} = g^{\frac{\hat{x}\cdot(\alpha+\beta\cdot\mathrm{dlog}_g(\mathbf{A}))}{\mathrm{dlog}_g \mathbf{A}}} = g^{\frac{\hat{x}\cdot\alpha}{\mathrm{dlog}_g \mathbf{A}}+\hat{x}\cdot\beta}$$

Again, the InvCDH solution can be recovered, as long as $\hat{x} \neq 0$. The responder session case is identical. ☐

E   Implications of UC AGM composability

In this section we give a more detailed discussion of the UC AGM model and the implications of our theorems. The UC-AGM model deviates from the UC model in two ways:

– it extends the execution model with one or more CRS functionalities for group definitions; and
– it constrains adversaries, environments and simulators to behave algebraically w.r.t. to said group descriptions.

We argue that any UC proof in the standard model implies, as expected, a proof in the UC-AGM model. This means that we can compose UC-secure protocols with UC-AGM protocols using the UC-AGM composition theorem. Moreover, we argue that UC-AGM emulation w.r.t. to different $\mathcal{F}_{\mathrm{CRS}}$ functionalities (possibly using the same mathematical group) also composes. These results create a hierarchy of UC emulation notions, all of which can be composed with each-other by further restricting the adversarial entities to be algebraic with respect to all groups used by the different protocols. We reinforce this last point: composition comes at the cost of weakening the overall claim to holding over algebraically restricted adversarial entities—combining results results in a more restrictive setting—which is the natural consequence of considering algebraic adversaries. We first give the high-level details and then demonstrate with examples.

ALGEBRAIC CONTEXTS. Let us first distinguish *adversarial machines* from *non-adversarial machines*. The former include simulators, real-world adversaries, environments and corrupted protocol participants. The latter include ideal functionalities and honest protocol participants.

The notion of an *algebraic adversarial machine* depends on the context in which this machine is executed. Fix a functionality $\mathcal{F}_{\mathrm{CRS}}$, defining a group description $\mathbb{G}$. Adversarial machines must behave algebraically w.r.t. to $\mathbb{G}$ whenever the bit strings they produce are *interpreted* (typed) as elements in $\mathbb{G}$ by the target non-adversarial machines—note that this applies only to group elements delivered via backdoor interfaces, but not to protocol inputs. Furthermore, these algorithms can only use as bases for their representations group elements that they read directly from $\mathcal{F}_{\mathrm{CRS}}$ or that they receive *with this interpretation* (type) from a non-adversarial machine. These interpretations must be fixed because proofs will rely on the restrictions that come as a consequence.

We call non-adversarial machines that never interpret an incoming or outgoing bit-string as an element in $\mathbb{G}$ as $\mathbb{G}$-*agnostic*. More precisely, in a context where $\mathcal{F}_{\mathrm{CRS}}$ is present and provides a description of $\mathbb{G}$, $\mathbb{G}$-agnostic machines do not restrict adversarial machines into providing algebraic representations for group elements in $\mathbb{G}$, nor do they produce group elements that can be used any adversarial entity as a basis for a representation of a group in $\mathbb{G}$.

It is crucial that, for all machines in the context of a claim and all groups $\mathbb{G}$ in the same context, one defines a priori whether a machine is adversarial and, if not, whether it is $\mathbb{G}$-agnostic. In the particular case of an ideal-world adversary $\mathcal{S}$ interacting with an ideal functionality, there is only one target non-adversarial machine. Intuitively, if $\mathcal{S}$ is interacting with an $\mathcal{F}$ that is $\mathbb{G}$-agnostic, then $\mathcal{S}$ is trivially algebraic w.r.t. to $\mathbb{G}$. For the particular case of $\mathcal{F}_{\mathrm{CRS}}$ defining $\mathbb{G}$, we have that $\mathcal{F}_{\mathrm{CRS}}$ is agnostic to all other groups and it is not agnostic w.r.t. to $\mathbb{G}$, as it provides the basis for group element representations.

UC EMULATION IMPLIES UC AGM EMULATION. Our first claim is that standard UC emulation w.r.t. a $\mathbb{G}$-agnostic functionality implies UC emulation in the AGM w.r.t. to $\mathbb{G}$.

REDUCTION $\mathcal{B}_{\ell_1,\ell_2}(\mathbf{A})$

---

proc $\mathrm{H}_1(sid, P, P', \mathsf{pw})$ (non-repeat i-th query)
$\overline{\text{Sample } k \twoheadleftarrow \mathbb{Z}_q^* \text{ s.t. } B^k \notin T_1 \cup \{\mathbf{A}\}}$
If $i = \ell_1$ Then $T_1[sid, P, P', \mathsf{pw}] \leftarrow (\mathbf{A}, \perp)$; Return $\mathbf{A}$
$T_1[sid, P, P', \mathsf{pw}] \leftarrow (g^k, k)$; Return g^k

proc $\mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z}))$ (non-repeat queries)
$\overline{K \twoheadleftarrow \mathcal{K}; \; T_2[sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z})] \leftarrow K; \; \text{return } K}$

If $\mathcal{E}$ causes $\mathcal{F}_{\mathsf{pwKE}}$ to send $(\mathsf{NewSession}, sid, P, P', \mathrm{role})$ to simulator
$\overline{(\text{Note reduction gets } \mathsf{pw} \text{ in the triggering NewSession query})}$
If $\neg(\pi_P^{sid} = \perp)$ discard input.
If $\mathrm{role} = \mathrm{init}$
    $\hat{x} \twoheadleftarrow \mathbb{Z}_q^*; \; \mathbf{X} \leftarrow g^{\hat{x}}$
    $\pi_P^{sid} \leftarrow (\hat{x}, (P, P', sid, \mathbf{X}, \perp), \perp, \mathrm{init})$
    Send $\textsc{SendInit}(P, P', sid, \mathbf{X})$ to $\mathcal{D}$
Else
    $\hat{y} \twoheadleftarrow \mathbb{Z}_q^*; \; \mathbf{Y} \leftarrow g^{\hat{y}}$
    $\pi_P^{sid} \leftarrow (\hat{y}, (P', P, sid, \perp, \mathbf{Y}), \perp, \mathrm{resp})$

On message $\textsc{SendInit}(P, P', sid, (\mathbf{X}, \mathsf{alg}) \neq \mathbf{1})$ from $\mathcal{E}$
$\overline{\text{If } \pi_{P'}^{sid} \neq (\hat{y}, (P, P', sid, \perp, \mathbf{Y}), \perp, \mathrm{resp}) \text{ discard input}}$
$K \twoheadleftarrow \mathcal{K}$
If $\pi_P^{sid} = (\cdot, (P, P', sid, \mathbf{X}, \perp), \perp, \mathrm{init})$ Jump to COMPLETE
If $\mathsf{alg} = [(\mathbf{G}, x)] \wedge (sid, P, P', \mathsf{pw}, \mathbf{G}, \cdot, \cdot) \in T_1$
    Compute $(\mathsf{TestPwd}, sid, P', \mathsf{pw})$ as $\mathcal{F}_{\mathsf{pwKE}}$
    If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" Then $K \leftarrow \mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Y}^x)$
Else Compute $(\mathsf{TestPwd}, sid, P', \perp)$ as $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: Send $\textsc{SendResp}(P', P, sid, \mathbf{Y})$ to $\mathcal{D}$
        $\pi_{P'}^{sid} \leftarrow (\hat{y}, (P, P', sid, (\mathbf{X}, \mathsf{alg}), \mathbf{Y}), K, \mathrm{resp})$
        Compute $(\mathsf{NewKey}, sid, P', K)$ as $\mathcal{F}_{\mathsf{pwKE}}$; send output to $\mathcal{E}$

On message $\textsc{SendResp}(P', P, sid, (\mathbf{Y}, \mathsf{alg}) \neq \mathbf{1})$ from $\mathcal{E}$
$\overline{\text{If } \pi_P^{sid} \neq (\hat{x}, (P, P', sid, \mathbf{X}, \perp), \perp, \mathrm{init}) \text{ discard input}}$
$K \twoheadleftarrow \mathcal{K}$
If $\pi_{P'}^{sid} = (\cdot, (P, P', sid, \mathbf{X}, \mathbf{Y}), \cdot, \mathrm{resp})$ Jump to COMPLETE
If $\mathsf{alg} = [(\mathbf{G}, y)] \wedge (sid, P, P', \mathsf{pw}, \mathbf{G}, \cdot, \cdot) \in T_1$
    Compute $(\mathsf{TestPwd}, sid, P, \mathsf{pw})$ as $\mathcal{F}_{\mathsf{pwKE}}$
    If $\mathcal{F}_{\mathsf{pwKE}}$ responds with "correct guess" Then $K \leftarrow \mathrm{H}_2(sid, \mathbf{X}, \mathbf{Y}, \mathbf{X}^y)$
Else Compute $(\mathsf{TestPwd}, sid, P, \perp)$ as $\mathcal{F}_{\mathsf{pwKE}}$
COMPLETE: $\pi_P^{sid} \leftarrow (\hat{x}, (P, P', sid, \mathbf{X}, (\mathbf{Y}, \mathsf{alg})), K, \mathrm{init})$
        Compute $(\mathsf{NewKey}, sid, P, K)$ as $\mathcal{F}_{\mathsf{pwKE}}$; send output to $\mathcal{E}$

On termination
$\overline{\text{Get the } \ell_1\text{-th query from } T_1 \text{: } (sid, P, P', \mathsf{pw})}$
Get the ℓ_2-th query from T_2: $(sid, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$
If $\exists \pi_P^{sid} = (\hat{x}, (P, P', sid, \mathbf{X}, (\mathbf{Y}, \mathsf{alg})), \cdot, \mathrm{init})$ for which $\mathcal{F}_{\mathsf{pwKE}}$ holds pw:
    Rewrite $\mathsf{alg} = [(\alpha, g), (\beta, \mathbf{A})]$ // (for passive attacks $(\alpha, \beta) = (\hat{y}, 0)$)
    Terminate outputting $(\mathbf{Z}/g^{\beta \cdot \hat{x}})^{\frac{1}{\alpha \cdot \hat{x}}}$
If $\exists \pi_{P'}^{sid} = (\hat{y}, (P, P', sid, (\mathbf{X}, \mathsf{alg}), \mathbf{Y}), \cdot, \mathrm{resp})$ for which $\mathcal{F}_{\mathsf{pwKE}}$ holds pw:
    Rewrite $\mathsf{alg} = [(\alpha, g), (\beta, \mathbf{A})]$ // (for passive attacks $(\alpha, \beta) = (\hat{y}, 0)$)
    Terminate outputting $(\mathbf{Z}/g^{\beta \cdot \hat{y}})^{\frac{1}{\alpha \cdot \hat{y}}}$

Fig. 13. Reduction $\mathcal{B}_{\ell_1,\ell_2}$ computes $\mathsf{InvCDH}(\mathbf{A})$. T_1 and T_2 are initially empty. $\mathcal{B}_{\ell_1,\ell_2}$ runs $\mathcal{F}_{\mathsf{pwKE}}$ internally, so it can check password guesses and answer corrupt queries as in the ideal world. It can also check whether random oracle queries are consistent with the session passwords on termination.

Theorem 9 (Informal). *Fix a protocol π that UC emulates $\mathbb{G}$-agnostic functionality $\mathcal{F}$. Then π UC emulates $\mathcal{F}$ w.r.t. $\mathbb{G}$-algebraic adversarial machines.*

Proof. (Sketch) Algebraic adversaries and environments are subclasses of general UC adversaries and environments, so an UC-AGM simulator $\mathcal{S}$ is guaranteed to exist for all algebraic environments and the dummy algebraic adversary; furthermore $\mathcal{S}$ is trivially algebraic because $\mathcal{F}$ is $\mathbb{G}$-agnostic. $\square$

The same argument applies when we consider UC-AGM emulation with respect to two different groups. Suppose $\mathcal{F}$ is not $\mathbb{G}_1$-agnostic but it is $\mathbb{G}_2$ agnostic. Suppose further than $\mathcal{S}$ is $\mathbb{G}_1$-algebraic when it interacts with $\mathcal{F}$. Then $\mathcal{S}$ is trivially $(\mathbb{G}_1, \mathbb{G}_2)$-algebraic when it interacts with $\mathcal{F}$. From here we obtain the following result.

Theorem 10 (Informal). *Fix a protocol π that UC emulates a $\mathbb{G}_2$-agnostic functionality $\mathcal{F}$ w.r.t. $\mathbb{G}_1$-algebraic adversarial machines. Then π UC emulates $\mathcal{F}$ w.r.t. $(\mathbb{G}_1, \mathbb{G}_2)$-algebraic adversarial machines.*

Combining these results with the AGM-UC composition theorem yields a comprehensive compositional framework.