J. Cryptol. (2011) 24: 720–760
DOI: 10.1007/s00145-010-9075-9

Journal of
CRYPTOLOGY

Secure Computation Without Authentication*

Boaz Barak†
Microsoft Research New England, Cambridge, MA 02142, USA
boaz@microsoft.com

Ran Canetti‡
Department of Computer Science, Tel-Aviv University, P.O. Box 39040, Tel-Aviv 69978, Israel
canetti@tau.ac.il

Yehuda Lindell§
Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel
lindell@cs.biu.ac.il

Rafael Pass¶
Cornell University, Ithaca, NY 14853, USA
rafael@cs.cornell.edu

Tal Rabin
IBM T.J. Watson, New York, USA
talr@watson.ibm.com

Communicated by Ivan Damgaard

Received 5 March 2008
Online publication 15 September 2010

Abstract. Research on secure multiparty computation has mainly concentrated on the case where the parties can authenticate each other and the communication between them. This work addresses the question of what security can be guaranteed when authentication is not available. We consider a completely unauthenticated setting, where *all* messages sent by the parties may be tampered with and modified by the adversary without the uncorrupted parties being able to detect this fact. In this model, it is not possible to achieve the same level of security as in the authenticated-channel setting. Nevertheless, we show that meaningful security guarantees *can* be provided: Essentially, all the adversary can do is to partition the network into disjoint sets, where in each set the computation is secure in of itself, and also *independent* of the computation

* An extended abstract of this paper appeared in the proceedings of *CRYPTO* 2005.

† Work partially carried out while at IBM T.J. Watson.

‡ Work carried out while at IBM T.J. Watson.

§ Work partially carried out while at IBM T.J. Watson.

¶ Work partially carried out while at IBM T.J. Watson, and partially supported by an Akamai Presidential Fellowship.

in the other sets. In this setting we provide, for the first time, nontrivial security guarantees in a model with *no setup assumptions whatsoever*. We also obtain similar results while guaranteeing universal composability, in some variants of the common reference string model. Finally, our protocols can be used to provide conceptually simple and unified solutions to a number of problems that were studied separately in the past, including *password-based authenticated key exchange* and *nonmalleable commitments*. As an application of our results, we study the question of constructing secure protocols in partially authenticated networks, where some of the links are authenticated, and some are not (as is the case in most networks today).

Key words. Multiparty computations, Unauthenticated channels, Man-in-the-middle attacks, Universal composability (UC), Password authentication, Partially-authenticated networks.

1. Introduction

In the setting of secure multiparty computation, a set of parties with private inputs wish to jointly compute some function of their inputs in a secure way. Loosely speaking, the security requirements are that nothing is learned from the protocol other than the output (*privacy*) and that the output is distributed according to the prescribed functionality (*correctness*). These security properties must be guaranteed even when some subset of the parties and/or an external adversary maliciously and actively attack the protocol with the aim of compromising the uncorrupted parties' security. Since its introduction in the 1980s [1,12,20,28], the research in this area has not subsided. The area has produced a rich body of work that deals with many aspects of the problem, including definitional issues, general feasibility results, protocols with low round and communication complexity, protocols that rely on a wide variety of computational assumptions, lower bounds, and security under composition.

A common approach in this body of work is to treat the *authenticated communication* aspect of the problem as extraneous to the actual protocol design. That is, practically all the existing protocols assume that the parties can communicate in an ideally authenticated way: the adversary is assumed to be unable to send messages in the name of uncorrupted parties or modify messages that the uncorrupted parties send to each other. This means that authentication must be provided by some mechanism that is external to the protocol itself.

The "ideally authenticated channels" abstraction has proven to be a powerful and useful tool in the design and analysis of protocols. Indeed, authenticated communication can be obtained in multiple very different ways that need not affect the design of higher-layer protocols. However, this abstraction is also somewhat limiting, since it makes full authentication a prerequisite to any meaningful solution to a multiparty computation problem. In contrast, it may be useful to provide some security guarantees even when full authentication is not available. That desire is highlighted by the fact that authentication is a nontrivial and intricate guarantee to obtain (or to even precisely specify) in some salient settings. In particular, any authentication solution (perhaps short of dedicated physically authenticated channels) relies in an essential way on some sort of trust in external computational entities, such as certification authorities or Kerberos-like authentication servers. This leads us to the following question:

What security can be obtained in a network without any authentication mechanism?

In addition to being a natural and interesting question on its own, studying this question enhances our understanding of the role of authentication in secure computation, even when authentication *is* possible. In fact, as seen below, our answer provides a general methodology for incorporating different authentication mechanisms in protocols. This methodology is based on *incorporating the authentication within the protocol itself,* rather than confining it to a separate module. Using this methodology, we construct conceptually simple protocols for seemingly unrelated tasks such as password-based key exchange and nonmalleable commitment.

Security Without Authentication: The Approach at a Glance. For simplicity, we begin by considering the special case of *two-party* protocols in an unauthenticated network. An immediate but important observation is that an adversary in such a network can simply "disconnect" the uncorrupted parties completely and engage in *completely separate* executions with each one of the two parties, where in each execution the adversary plays the role of the other party. Such an attack is unavoidable since, without authentication, the parties have no way of distinguishing the case where they interact with each other from the case that they each interact separately with a third party (in this case, the adversary). Our aim is to guarantee that this is the *only* attack that the adversary can carry out. More specifically, our notion of security guarantees that the adversary is limited to pursuing one of the two following strategies:

1. *Message relaying:* In this strategy, the adversary honestly relays the communication between the two parties, allowing them to perform their computation as if they were communicating over an authenticated channel. (As always, the adversary may still drop messages at will.)
2. *Independent executions:* In this strategy, the adversary intercepts the communication between the parties and engages in "independent" executions with each of them. That is, for parties A and B, the adversary can run an execution of a secure protocol with A while playing B's role and an execution of a secure protocol with B while playing A's role. Furthermore, the two executions are independent, except for potentially determining the inputs to one execution as a function of the inputs and outputs of the other execution. A bit more precisely, if we think of each execution as emulating an ideal interaction with a trusted party for the task at hand, then it is guaranteed that the overall interaction emulates an ideal model where A and B interact with *two separate trusted parties*.

This guarantee is extended in a natural way to the case of multiparty protocols. Specifically, an adversary can always partition the uncorrupted parties into *disjoint* subsets. Then, given this partition, the adversary can run separate (and independent) executions with each subset in the partition, where in an execution with a given subset H of the parties, the adversary plays the roles of all the parties outside of H. We guarantee that this is the only attack the adversary can carry out. That is, we consider an adversary who interacts with a set of parties who are each willing to run a single execution with each other. Then, although the adversary can actually run a different execution with each subset of the parties, we guarantee that:

1. Once a subset of uncorrupted parties is chosen, it is fixed for the duration of the protocol,
2. The subsets of uncorrupted parties are disjoint, and
3. The overall interaction emulates an ideal model where each subset H interacts with its own separate trusted party, where the parties not in H are played by the adversary. In particular, this implies that the only dependence between the subsets is due to the capability of the adversary to choose its inputs as a function of the outputs that were already generated by other executions. (In essence, the interaction within each subset of parties is the same as when there are authenticated channels.)

We note that the basic idea of limiting the adversary to either "message relaying" or "running independent sessions" originates in the notion of *nonmalleability* [13]. Indeed, the protocols in [13] essentially provide these guarantees for the specific tasks of commitments and zero-knowledge proofs. Viewed this way, the contribution of the present work is in extending this idea to general two-party and multiparty computation. This contribution is twofold: first, we *formalize* this concept for general computation; second, we show how realize it via a natural (yet somewhat surprising) combination of techniques.

1.1. *Our Results in More Detail*

We instantiate the above approach in three different settings. The first setting provides the parties with *no trusted set-up whatsoever.* Here we obtain a notion of security which we call bounded UC. This notion is strictly stronger than the basic notion of security of [3,16] (but still weaker than full-fledged UC). Essentially, bounded-UC security guarantees that security is preserved under unlimited nonconcurrent composition, plus a limited form of concurrent composition. To the best of our knowledge, this is the first work to formulate this notion, which seems to be of general interest.

The second setting assumes the *common reference string* (*CRS*) model as in [4,6,10]. Here each protocol instance is guaranteed to have a dedicated reference string that is taken from predetermined distribution. In this setting we obtain full-fledged *universally composable* (*UC*) secure computation [4]; namely the protocols are guaranteed to remain secure when concurrently composed with any other set of protocols or network activity.

The third setting assumes the *augmented CRS* (*ACRS*) model as in [5]. Here a single reference string is used by all protocol instances in the system; in addition, adversarial parties are allowed to obtain some personalized trapdoor information on the reference string. Here too we obtain full-fledged UC secure computation. (To allow modeling a global reference string, we use the *generalized* UC framework [5].) See more discussion on the CRS and ACRS models at the end of this section.

Both the definitional approach and the high-level structure of the construction are similar for all three settings. We thus present them together. We first provide more details on the new notion of security. Next, we review the construction (which is actually quite simple, conceptually).

The Notion of Security. As sketched earlier, the basic definitional idea is to require that running the protocol will amount to "emulating" the following ideal process: The adversary chooses (in an adaptive way throughout the computation) disjoint subsets of parties. Once a subset is fixed, the uncorrupted parties in the subset interact with a trusted party that is "dedicated" to this subset, where the rest of the parties (i.e., the parties that complete this subset to the full set of parties) are played by the adversary. The idea here is that the ideal process limits the adversary to the "trivial" and unavoidable behavior of "honestly" running different executions with different subsets, while determining future inputs to the various execution based on the legitimate outputs it received so far from all executions.

One way to formalize this idea is to formulate a special-purpose ideal process that captures the above process. However, using such a formalism may make it harder to compose protocols analyzed in this model with other protocols (such as partial authentication protocols) in order to guarantee some sort of combined security properties. We thus use a somewhat different approach that allows us to keep the ideal process the same as in standard definitions and thus use existing composability results. Specifically, instead of letting the adversary invoke multiple instances of the trusted party, we keep to the standard formulation where all parties interact with a single trusted party. We then modify the program of the (single) trusted party, namely the *ideal functionality,* to mimic an interaction with multiple separate trusted parties, consistently with the adversarially defined subsets.

A bit more precisely, for any functionality $\mathcal{F}$ to be realized by two or more parties, we define a relaxed version of $\mathcal{F}$ called split-$\mathcal{F}$, or s$\mathcal{F}$, which is an *interactive functionality* and works as follows. Functionality s$\mathcal{F}$ lets the adversary define disjoint sets of parties, called authentication sets. Then, a separate and independent instance of the original functionality $\mathcal{F}$ is locally invoked for each authentication set. In an ideal execution of an instance of $\mathcal{F}$ for a given authentication set, functionality s$\mathcal{F}$ allows the adversary to play the roles of all the parties not in the set; i.e., the adversary provides the inputs of these parties and receive their outputs.

As an example, consider the two-party case. Here the adversary can either choose a *single* authentication set containing both parties, or it can choose two authentication sets, each containing a single party. In the first case it cannot do anything more than in the authenticated channels model. In the second case it must run an independent and separate execution with each party.

We can now informally state our two main theorems. Following the theorem statements, we discuss some properties of the results. We note that both theorems hold only for functionalities satisfying certain syntactic conditions, as defined in [10].

Theorem 1 (Unauthenticated Bounded-UC Computation). *Assume the existence of collision-resistant hash functions and enhanced trapdoor permutations* (*see* [16, *Appendix* C.1]) *and consider the standalone model with no setup whatsoever. Then, for any probabilistic polynomial-time multiparty functionality* $\mathcal{F}$, *there exists a protocol that securely realizes the split functionality* s$\mathcal{F}$, *with bounded-UC security in the presence of static, malicious adversaries.*

Theorem 2 (Unauthenticated UC and gUC Computation). *Assume the existence of enhanced trapdoor permutations. Then for any ideal functionality* $\mathcal{F}$, *there exist pro-*

tocols that realize the split functionality $\mathsf{s}\mathcal{F}$ *with UC security in the CRS model, or alternatively with gUC security in the ACRS model, in the presence of static, malicious adversaries. Assuming the existence of enhanced noncommitting encryption (as in* [10]*), this holds even with respect to adaptive adversaries and without data erasures.*

Before turning to the constructions, let us point out that an effort was made to present the proofs of these two theorems (i.e., the different constructions and their analyses) using a single framework that allows expressing within it different composability guarantees and use different trust models. Such combined analysis is both simpler and clearer than proving each result in a different formal framework. In particular, substantial parts of the constructions and analyses are the same in both proofs. Using a single framework allows us to avoid redoing these parts anew for each proof.

The Construction. The protocol is comprised of two stages. In the first stage, an attempt is made to establish "pseudo-authenticated" channels. In the second stage, a concurrently composable secure protocol is run on top of these "pseudo-authenticated" channels. The crux of the analysis is to show that the combination of the pseudo-authenticated channels with concurrently composable protocols that make cryptographic use of some public information on the channels suffices for guaranteeing our notion of security. More precisely, assume that there is a set of parties where the parties know each other's identity and wish to engage in a single protocol execution with each other over an unauthenticated network. The protocol proceeds as follows.

Stage 1—Link initialization: In this stage, each party P_i generates a pair of signing and verification keys (s_i, v_i) of a standard signature scheme and sends the verification key v_i to the parties in the set. Next, after receiving verification keys v_j from all parties, P_i signs (using its secret key s_i) the sequence of all keys received, together with the names of the corresponding senders, and sends the signature σ_i to all other parties. Finally, each P_i waits to receive signatures from all parties and checks that all the signatures refer to the same set of verification keys. If all checks verify, then P_i uses the signature keys to set up an authenticated channel with each other party and continues to the next stage, where all messages are sent over these authenticated channels.

The idea behind this step is as follows. Let P_i and P_j be uncorrupted parties, let v_i be the verification key sent by P_i to P_j, and let v_j be the key sent by P_j to P_i. Since these keys are sent over unauthenticated channels, there is no guarantee that P_j will actually receive v_i rather than some $v_i' \neq v_i$ generated by the adversary (and likewise for P_i). However, it is guaranteed that *if* P_i and P_j receive each other's real keys, *then* the subsequent communication between them will be authenticated.

Furthermore, at the end of this stage, the parties hold public values that correspond to the subsets of parties among which the authentication succeeded. That is, let S_i denote the sequence of public keys that P_i received from the other parties, including itself. (If P_i did not successfully complete the first stage, then $S_i = \perp$.) Then, $S_i = S_j \neq \perp$ if and only if both P_i and P_j completed the first state, and they have each other's correct public keys. This means that the set of uncorrupted parties which completed the first stage is partitioned into disjoint subsets (called authentication subsets), where $S_i =$

S_j if and only if P_i and P_j are in the same set, and where in each set all the parties can communicate in an authenticated way. We call the S_i values session identifiers (SIDs). The SIDs will be used in a crucial way in the second stage of the computation, to provide the necessary "binding" between the authentication and the computation stages.

Stage 2—Secure computation using the generated links: In this stage, the parties essentially run a generic multiparty computation protocol *that assumes authenticated channels,* on top of the pseudo-authenticated channels generated in the first stage. There are two important caveats though: First, standard, standalone security is not enough. The protocol must be *concurrently composable*. Second, each party uses the SID generated in the first stage as its session identifier in the second stage.

The rationale here is as follows: Recall that in this stage the uncorrupted parties are partitioned to *authentication subsets*, identified by the SIDs, where the communication within each subset is essentially authenticated. Consequently, one can regard the entire interaction as concurrent execution of several instances of the protocol, where each instance is associated with a different SID. However, since the protocol is concurrently composable, each one of these instances maintains its own security.

We use different protocols for the bounded-UC, UC, or generalized UC results. For the bounded-UC case, we use the general construction of Pass [25], which guarantees concurrent composition of a predefined number of protocol executions. (This *bounded concurrency* guarantee suffices in our case, since the number of executions that can run concurrently is bounded by the number of parties.) This protocol can run in the bare model, namely on top of the pseudo-authenticated channels provided by the first stage, with no additional setup.

To obtain UC security in the common reference string (CRS) model, we use the construction of [10]. To obtain generalized UC in the ACRS model, we use the construction of [5]. The protocol that uses the CRS model is guaranteed to compose securely with any other set of arbitrary protocols or network activity *that does not use the same reference string.* The protocol that uses the ACRS model guarantees secure composition even with protocols that use the *same* (*augmented*) *reference string*.

Note that the three constructions differ only in one building block, namely in how the zero-knowledge functionality is realized. In particular, the three constructions use the SIDs generated in the first stage in a similar way, in order to guarantee "independence" between executions; this "independence" is crucial for secure concurrent composability.

We remark that many of the techniques used here have appeared in some form or other in the literature. The idea of using "self-signed messages" with ephemeral public keys in order to bind between various components of a protocol or a message has been used multiple times, often in the context of guaranteeing nonmalleability (see, e.g., [13]). Also the idea of exchanging ephemeral verification keys in order to reach a weak flavor of authentication or agreement in a multiparty setting has appeared in multiple places, e.g., [7] and [15] (albeit without signing the sequence of public keys or using this sequence in a subsequent protocol). On the other hand, the present use of concurrent composability, together with the method of assigning unique SIDs that can also be used to enforce authenticity within sessions, is new to the best of our knowledge.

Before proceeding to sketch some applications and additional results, we discuss the following properties of the main results.

Agreement is not Obtained. We stress that although Theorems 1 and 2 constitute "general feasibility results," the security guarantee obtained is *far weaker* than that of the authenticated channels model. For example, agreement-type problems cannot be solved in this model. Indeed the split-functionality formalization explicitly removes all flavor of agreement, since uncorrupted parties in different authentication subsets run independent executions and so clearly are not guaranteed to agree on anything.

Honest Majority is Meaningless. We note that, unlike the setting of authenticated channels, here it does not help to assume that a large fraction of the parties are uncorrupted. This is due to the fact that the adversary can always choose all the subsets to be small, thereby ensuring an uncorrupted minority in each execution. In particular, it is impossible to prevent an adversarial early abort, and neither fairness nor output delivery can be guaranteed in this setting. Indeed, Theorems 1 and 2 hold irrespective of the number of corrupted parties and do not guarantee fairness or reliable output delivery.

Separating Entity Authentication from Session Authentication. One aspect of our results is a more explicit distinction between *entity authentication* and *session authentication* in secure computation. Entity authentication provides a party A with a guarantee that the messages that it received in the name of party B were indeed sent by B. In contrast, session authentication only provides A with the guarantee that it interacts with the same entity throughout the protocol execution (the "session"). Party A does not know the identity of the party with whom it holds the channel; however, it knows that if the party is uncorrupted, then the adversary cannot interfere with any messages that are sent on the channel. While this distinction appears in certain forms in the literature (e.g., [13,27]), here it is made more explicit. Indeed, our results can be viewed as explicitly formalizing and achieving session authentication without entity authentication. This separation may be quite useful, since it allows entity authentication methods to be carried out *separately from* session authentication, and in many different ways. Specifically, within the same execution, different parties may use different authentication mechanisms like passwords, digital signatures, interactive authentication protocols, group-based authentication (potentially with anonymity guarantees), and so on. In particular, the application to password-based key exchange, described in the next subsection, can be regarded as an instantiation of this methodology.

Concurrency in a Standalone World. From the above informal description of our protocol we see that, in the unauthenticated channels setting, obtaining even *standalone security* inherently requires withstanding adversaries that run *concurrent executions* of the protocol with different sets of uncorrupted parties.

However, an important observation here is that when there are n uncorrupted parties, the adversary can force at most n concurrent executions (because the sets are disjoint and each uncorrupted party runs only once). It therefore follows that we only need security under *bounded* concurrency, which is fortunately much easier to achieve. (See [23] for impossibility results for the setting of unbounded concurrency in the plain model, in contrast to the feasibility results of [22,25,26] for bounded concurrency.)

On the CRS and ACRS Models. Recall that the CRS and ACRS models postulate existence of a string that is available to all parties and is guaranteed to be sampled from some predetermined distribution. Still, the guarantees provided by the two models are incomparable: On the one hand, in the CRS model it is assumed that each protocol instance has its own dedicated reference string, which was chosen independently of the strings used in other instances. In contrast, in the ACRS model only a single string is used for all the protocol instances in the system. On the other hand, The ACRS model provides an additional "interface," whereby corrupted parties may obtain some "trapdoor information" on the string. We stress though that this trapdoor information only comes up in the analysis and is not available to protocols designed in this mode. In particular, any protocol designed for either the CRS or the ACRS model can be analyzed in either model.

To justify the use of the CRS and ACRS models, we recall that obtaining UC security for a wide array of tasks is impossible in the plain model [4,6,9], even if authenticated communication is provided. Similarly, obtaining UC security with respect to globally available trusted information is impossible when the information is strictly public, as in the CRS model [5].

Another justification for the CRS and ACRS models is that they are essentially "orthogonal" to the trust models needed for authentication: On the one hand, the CRS and ACRS models require all parties to have access to and trust a single entity, or a string. On the other hand, they do not require individual parties to register themselves in any way. In contrast, setting up authenticated communication essentially requires each party to register individually (say, to obtain a certificate for its public key) and maintain some related secret information known only to itself. In particular, it is impossible to construct authenticated channels in either the CRS or the ACRS models without additional trust.

1.2. *Additional Results and Applications*

We list some additional results and applications of our main results. Very recently, Camenisch et al. have further extended the results here (especially for the case of password and credential-based identification and key exchange), providing significantly more efficient protocols than the ones here [2].

Password-Based Authenticated Key Exchange. Password-based key exchange is the problem where two parties, who share only a short secret string with limited entropy (e.g., a password), wish to agree on a full-fledged cryptographic key over an unauthenticated channel. This problem proved quite tricky to formalize. Furthermore, provably secure password-based key-exchange protocols have been relatively involved, using multiple underlying primitives in special ways. See, e.g., [8] and the references within for various definitional efforts and constructions.

The core of the problem here is that since the password is a "weak secret" that can be guessed with noticeable probability, it is impossible to guarantee full authentication. Still, one wants to capture the fact that some weak form of authentication is indeed achieved. Informally, the standard requirement here is that the adversary will be essentially limited to the trivial and unavoidable attack of running the program of an uncorrupted party with a randomly guessed password. This attack works only in the event that the password guess happened to be correct; the probability of this event depends only

on the entropy of the password, rather than on the protocol itself. In particular, verifying each additional password guess will necessitate another execution of the protocol.

The literature contains a number of quite different ways for formalizing the above requirement, with varying degrees of restrictiveness and security guarantees. The definitional framework developed here can be used to provide a natural and strong formalization of this requirement. Furthermore, our general constructions immediately yield conceptually simple password-based key-exchange protocols, based on general assumptions, with strong security and composability properties.

In more detail, consider the following functionality $\mathcal{F}_{pw}$. Each party provides an input (a "password"); if the inputs are equal, then $\mathcal{F}_{pw}$ provides each party with a long random value (a "cryptographic key"); if the inputs are not equal, then $\mathcal{F}_{pw}$ hands $\perp$ to each party. Of course, the inputs we are referring to here are the parties' secret passwords. $\mathcal{F}_{pw}$ guarantees both agreement and secrecy for the generated key; thus any protocol that realizes it would be sufficiently secure. However, $\mathcal{F}_{pw}$ is too strong and is not realizable in our unauthenticated setting. (In particular it does not enable the adversary to make "online password guesses", which are inherently possible in password based key-exchange schemes.) We thus resort to realizing the split functionality $\mathsf{s}\mathcal{F}_{pw}$. Indeed, a protocol that securely realizes $\mathsf{s}\mathcal{F}_{pw}$ guarantees a strong notion of security: The adversary is limited to either including both parties in the same authentication set (which corresponds to providing them with a random and secret joint key) or having each party be in its own authentication set (in which case the adversary is limited to a single password guess in the name of each party).

We note that this definition is essentially the same as that proposed in [8]. It is interesting to note that the definition of [8], which was developed specifically for password-based key exchange, can be obtained by applying the general methodology of split functionalities to a variant of the classic "equality testing" function.

Our general constructions solves the password-based key-exchange problem in a conceptually simple way. They improve on previous ones as follows. Applying Theorem 1, we obtain secure password-based authenticated key exchange in a setting with no setup assumptions. The only previously known protocols to achieve this (without using random oracles) are [17,24]. Comparing our result to [17,24], we have the following advantages. First, we obtain a stronger security guarantee for the parties. Specifically, we guarantee exactly two password guesses per execution, rather than a constant or even polynomial number of guesses. Furthermore, these guesses are *explicit* (see [8] for a discussion about why this is advantageous). Second, our solution directly generalizes to password authentication protocols for *multiple* parties (whereas previous solutions only work for two parties).

Applying Theorem 2, we obtain UC-secure password-based authenticated key exchange in the common reference string model. This problem was previously considered by [8], who present highly optimized protocols based on specific assumptions. In contrast, we obtain less efficient protocols, based on general assumptions. However, our protocol is secure even in the case of *adaptive corruptions* and is the first to achieve this. That is:

Theorem 3 (Password-Based Key Exchange). *Assume the existence of collision-resistant hash functions and enhanced trapdoor permutation and consider the stand-alone model with no setup whatsoever. Then there exists a protocol that realizes the*

password-based key-exchange functionality of [8] *with bounded-UC security in the presence of static corruptions.*

Furthermore, in the CRS and ACRS models there exist protocols that realize the password-based key-exchange functionality of [8] *with UC and gUC security, respectively, in the presence of static corruptions. Assuming the existence of enhanced noncommitting encryption, this holds even with respect to adaptive adversaries and without data erasures.*

Partially Authenticated Networks. So far, we have discussed a completely unauthenticated setting and have contrasted it to the standard completely authenticated setting. However, the most realistic setting in today's Internet is that of a *partially authenticated network*, where some of the parties are connected via authenticated links, and others are not. In addition, the authentication on these links may be unidirectional or bidirectional. For example, the most common case in the Internet today is one where servers have digital certificates for public (signature) keys as part of an implemented public-key infrastructure, whereas clients do not (but sometimes do have passwords). While this setting is addressed by some key exchange protocols (e.g., [21,27]), the option of running protocols for general secure computation in this setting has not been considered. For instance, we should be able to use a secure auction protocol, even if the only party who has a certificate is the auctioneer. Our results extend this setting. In particular, it allows one to obtain secure computation in partially authenticated networks while utilizing the authenticated links that do exist. This is a much more realistic setting than the standard one that requires that all pairs of parties are connected via authenticated channels.

Alternative Authentication Mechanisms. Passwords are just one mechanism for authenticating parties. The same methodology can be used to obtain secure protocols for other, nonstandard ways for parties to authenticate each other. The only requirement for accommodating these methods is that they can be described by an efficient functionality (and thus can be incorporated into stage 2 of the protocol). For example, we can accommodate "fuzzy" authentication where parties are authenticated if they pass at least k out of n "authentication tests," such as remembering the names of at least three of your childhood friends. Our solutions can also work in the case where parties are considered authenticated if they can perform some nontrivial computational task, like the "proof of work" in the anti-spam work of [14]. Finally, our protocols can be used to address cases where two or more parties wish to authenticate themselves to each other based on useful data which they hold, as in the case of peer-to-peer and overlay networks. This last type of authentication may also be used to guarantee anonymity, say when the same piece of identifying information is available to a set parties, and the participant does not wish to reveal its individual identity within that set.

Nonmalleable Commitments. Finally we remark that nonmalleable commitments [13] can be obtained using our results in a similarly simple manner. Namely, define $\mathcal{F}$ to be a two-party functionality that on input m from one party outputs $f(x)$ to the other party, where f is some noninteractive (and potentially malleable) commitment function. Then, a protocol that securely realizes $\mathsf{s}\mathcal{F}$ constitutes a nonmalleable commitment. This

protocol does not improve on other known protocols. Nonetheless, it demonstrates the power of our general framework.

Organization. The rest of the paper is organized as follows. Section 2 overviews the models of computation considered (bounded-UC, standard UC, and generalized-UC). Section 3 formalizes the notion of split functionalities described above. The main theorems are then stated and proved in Sect. 4. Section 5 concludes with our results regarding partially authenticated networks.

2. The Model of Computation

This section briefly summarizes the basic model of unauthenticated, asynchronous computation, presents the notions of UC, gUC, and bounded UC security, and reviews the composition theorem. We also present the notion of multiemulating protocols and multirealizing ideal functionalities. This notion plays a central role in our security analysis of Sect. 4.

We consider a completely unauthenticated, asynchronous, and unreliable communication network. This is modeled by postulating that the adversary gets hold of all messages sent by the parties. It is then up to the adversary to deliver whatever messages it wishes to parties; the delivered messages need not be related to the messages sent in any way. This of course means that messages can be duplicated, reordered, dropped, and modified to the adversary's liking. Also the real source of a message is not known to the recipient. This modeling is essentially the same as in [4]. (In contrast, the models in [3] and [16, Chap. 7] concentrate on synchronous and authenticated communication.)

This section contains a sketch of the model and definitions. Full details appear in [3,4,16]. We first present the model of computation, ideal protocols, and the general definition of securely realizing an ideal functionality. Next we present hybrid protocols and the composition theorem. See [4] for a more detailed description of all these notions.

2.1. *The Basic Model*

Protocol Syntax. Following [19], both protocols and adversarial entities are represented as interactive Turing machines (ITMs). Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model inputs from other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes model messages received from the network. In addition, an ITM has an identity tape that can be written to only once upon creation. The identity tape contains the program of the ITM (in some canonical representation) plus an identity string. Writing into a tape of another ITM is done via a special external write operation, whose effect is described below.

Systems of ITMs. As a preliminary step toward defining security of protocols, we sketch the "mechanics" of a joint execution of multiple communicating ITMs. A basic concept here is an ITM instance (ITI) which represents a process in a distributed system. An instance is captured by the program it is running and by an identity that distinguishes it from other processes (ITIs) that run the same program. Formally, an ITI of an ITM M is a pair (M, ID), where $ID \in \{0, 1\}$ is the contents of the identity tape.

A system of ITMs is defined via a pair (M, C), where M is an ITM, and $C : \{0,1\}^* \to \{0,1\}^*$ is a control function that determines the effect of the external write commands. An execution of a system (M, C) of ITMs, on input x, consists of a sequence of activations of ITIs, as follows. Initially, the system consists of a single ITI with program M, some fixed identity (say, $ID = (0,0)$), and some input value x written on the input tape. This ITI, called the initial ITI, is then activated.

Each activation of an ITI (M, ID) is a sequence of configurations of M, until an external write command is executed or an inactive state is reached. The execution ends when the initial ITI halts. The output of an execution is the output of the initial ITI.

It remains to specify the effect of the external-write operation. This operation specifies a target ITI (namely, program and identity), a tape out of {input, communication, subroutine output}, and data to be written. When an external-write operation is carried out, the control function C is applied to the sequence of external write requests in the execution so far. Then:[1]

1. If C returns 1, then:
   (a) If an ITI with the same identity as the target ITI does not exist in the system, then a new ITI with the specified program and identity is created and added to the system. This allows dynamic generation of new ITIs during the execution.
   (b) If the target tape is the input or subroutine output tape, then the identity and program of the writing ITI is added to the message. This allows an ITM to verify the true identity and program of its subroutines and callers.
   (c) If an ITI with the same identity and program as those specified in the external write command exists in the system, then the specified data is written to the specified tape of this (unique) ITI.
   (d) If an ITI with the specified identity exists, but this ITI is running a different program than the one specified, then: If the specified tape is the communication tape, then the specified data is written to the communication tape of the target ITI in the same way as in item 1(c). If the specified tape is the input or subroutine tape, then no data is written, and the writing ITI transitions to a special error state.

   In either case, the active ITI becomes inactive, and the target ITI is activated.
2. If C returns 0, or the active ITI entered an inactive state, then the initial ITI is activated.
3. If C returns another value, then this value is interpreted as a description of an ITM M. The effect is the same as in Case 1, except that the program of the target ITI is taken to be M rather than the value specified in the external write command. (This technical detail is needed for the notion of protocol emulation to make sense.)

Subroutines. An ITI μ is a subroutine of ITI μ' in an execution if μ wrote to the input tape of μ or μ' wrote to the subroutine output tape of μ'. μ is a descendant of μ' if it is a subroutine of μ' or of another descendant of μ'.

[1] A more formal description in terms of sequences of configurations can be extracted from this description.

Protocols and Protocol Instances (*Sessions*). To define protocol instances, we let the identity of each ITI to consist of two fields: the session-identifier (SID) field, and a party-identifier (PID) field. This way, a protocol instance is determined by a program (ITM) and a session identifier. That is, the instance s of protocol M in a given configuration of a system of ITMs is the set of ITIs that have program M and SID s. The party identifier of an I is often associated with a physical computer or a trust domain within which the ITI is running; however, the model does not enforce this convention.

Polynomial-Time ITMs. We consider ITMs that run in probabilistic polynomial time (PPT), where PPT is defined as follows: An ITM M is PPT if there exists a constant $c > 0$ such that, for any ITI μ with program M, at any point during its run, and for any contents of the random tape, the overall number of steps taken is at most n^c, where n is the overall number of bits written to the input tape of μ minus the overall number of bits written by μ to input tapes of other ITIs. (The purpose of this definition is to have syntactically verifiable conditions which guarantee that running a system of ITMs does not consume "super-polynomial resources." In particular, it can be seen that an execution of a system of ITMs, where the initial ITM is PPT, and the control function is polytime computable, can be simulated on a standard PPT Turing machine.)

2.2. *Defining Security of Protocols*

Protocols that securely carry out a given task are defined via comparison with an ideal process for carrying out the task. Formalizing this concept consists of three elements: defining the process of executing a protocol in the presence of an adversarial environment; defining what it means for one protocol to "emulate" another protocol; and defining the "ideal process" for carrying out the task in terms of a special idealized protocol. A protocol is said to securely carry out the task if it emulates the idealized protocol for that task.

We provide three different formalizations of this idea, resulting in our three different measures of security: global UC, UC, and bounded UC. The three formalizations differ only in the first element described above, namely in how the process of protocol execution is formalized. We first fully define UC security. Next we describe the other two notions.

The Model for Protocol Execution (*UC Security*). The model for executing a protocol π is parameterized by a security parameter $k \in \mathbf{N}$ and three ITMs: the ITM π, an ITM $\mathcal{A}$ called the adversary, which represents the adversarial activity against a single instance of π, and an ITM $\mathcal{Z}$, called the environment, which represents the rest of the system. Specifically, to run protocol π on input x, execute the system of ITMs $(\mathcal{Z}, C_{\mathcal{A},\pi})$. It remains to describe the control function $C_{\mathcal{A},\pi}$, namely the external write capabilities of each ITI.

In essence, the definition of the control function captures a model where a *single instance* of π interacts with $\mathcal{Z}$ and $\mathcal{A}$. $\mathcal{Z}$ controls the inputs to parties and reads the outputs. All communication (via the communication tapes) must pass through $\mathcal{A}$. In addition, the ITIs running π can create subroutine ITIs, can write to the input tapes of the subroutines, and receive outputs from the subroutine on the subroutine output tapes of the calling parties. More precisely:

External writes by the environment: The environment can write only to input tapes of other ITIs. The program of the first ITI invoked by the environment is set (by the control function) to be the program of the adversary $\mathcal{A}$. The programs of all the other ITIs that the environment writes to are set to be the protocol π.[2] In addition, the session IDs of all the ITIs invoked by the environment (other than the adversary) must be the same. That is, let s denote the SID of the first ITI to be invoked with program π. Then all the remaining ITIs invoked by the environment must have SID s. Consequently, all the ITIs invoked by the environment, except for the adversary, belong to the same instance of π.

External writes by the adversary: The adversary can write only to the communication tapes of ITIs.

External writes by other ITIs: An ITI μ other than the environment and the adversary can write to the input and subroutine output tapes of any ITI other than the environment. ITIs whose SID is the same as the SID s chosen by the environment may also write to the subroutine output tape of the environment. In addition, μ can write to the communication tape of the adversary.

For measuring runtime, we use the convention that the first input to be written to an ITI must start with 1^k, where k is the security parameter. (Note that an ITI might be invoked by writing to a tape other than the input tape, e.g., it may be invoked by the adversary. In this case the ITI may still have some constant runtime, depending on its bounding polynomial.)

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the output distribution of environment $\mathcal{Z}$ when interacting with parties running protocol π on security parameter k and input z. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k,z\in\{0,1\}^*}$.

Protocol Emulation. Informally, we say that a protocol π emulates protocol ϕ with UC security if for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with nonnegligible probability whether it is interacting with $\mathcal{S}$ and parties running π, or it is interacting with $\mathcal{S}$ and parties running ϕ. This means that, from the point of view of the environment, running protocol π is "just as good" as interacting with ϕ. This notion is formalized as follows. A distribution ensemble is called binary if it consists of distributions over $\{0,1\}$. That is:

Definition 4. Two binary distribution ensembles $\{X(k,a)\}_{k\in,a\in\{0,1\}^*}$ and $\{Y(k,a)\}_{k\in,a\in\{0,1\}^*}$ are called indistinguishable (written $X \stackrel{c}{\equiv} Y$) if for any $c,d\in\mathbf{N}$, there exists $k_0\in\mathbf{N}$ such that for all $k>k_0$ and for all $a\in\{0,1\}^{k^d}$, we have

$$\left|\Pr\big(X(k,a)=1\big)-\Pr\big(Y(k,a)=1\big)\right|<k^{-c}.$$

Definition 5 (Protocol Emulation). Let π and ϕ be protocols. We say that π emulates ϕ with UC security if for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{S}$ such that for any

[2] The reason for having the control function (rather than the environment) determine the program π is to allow a situation where the program π is changed into another program π' without having the environment being necessarily aware of the change. This detail is necessary for the definition of protocol emulation to make sense.

environment $\mathcal{Z}$ that outputs a value in $\{0, 1\}$, we have

$$\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}.$$

Ideal Functionalities and Ideal Protocols. A key ingredient in the ideal process for a given task is the ideal functionality that captures the desired behavior or, in other words, the specification of that task. An ideal functionality is modeled as an ITM (representing a "trusted party") that interacts with the ITIs of some protocol instance and also with the adversary.

For convenience, the process for realizing an ideal functionality is represented as a special type of protocol, called an ideal protocol. In the ideal protocol $I_{\mathcal{F}}$ for ideal functionality $\mathcal{F}$, all parties simply hand their inputs to an ITI with program $\mathcal{F}$, session ID that is equal to the local session ID, and party ID set to some fixed value, say $\perp$. Whenever a party in $I_{\mathcal{F}}$ receives a value from $\mathcal{F}$ on its subroutine output tape, it immediately copies this value to the subroutine output tape of the ITI that invoked it. We call the parties of the ideal protocol dummy parties. We use the convention that the PID of a dummy party includes the entire identity of its calling ITI. (This convention allows ideal functionalities to makes decisions based on the true identities of the ITIs in the protocol instance that called the current instance of $I_{\mathcal{F}}$.)

Definition 6 (Realizing Functionalities). Let π be a protocol, and let $\mathcal{F}$ be an ideal functionality. We say that π realizes $\mathcal{F}$ with UC security if π emulates $I_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$, with UC security.

Generalized UC (gUC) Security. The model of protocol execution in the definition of UC security allows a protocol instance to communicate with the rest of the system only via the inputs from the environment and outputs to the environment. This does not capture situations where many (or all) protocol instances access some common, global entity that is trusted to run some predefined program. gUC allows modeling such situations. This is done by lifting the restriction on the identities and protocols that the environment may invoke. That is, for gUC security, the environment can invoke ITIs with arbitrary (PPT) program and identities.

To have a meaningful notion of protocol emulation, we allow the environment to single out a single protocol instance out of the environment's subroutines and leave the program of this instance empty, say by replacing it with a special symbol $\perp$. The control function $C_{\mathcal{A}}, \pi$ will then let the program of the ITIs in this instance be π. Let $\text{EXEC}^{gUC}_{\pi,\mathcal{A},\mathcal{Z}}$ be the same as $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$, with the exception that the execution model is modified as described here.

Bounded UC Security. Bounded-UC security is a restricted version of UC security, where the environment can pass to the adversary only an a priori bounded amount of information during the execution. Specifically, after the initial input that the environment sends to the adversary, the overall length of all later inputs to the adversary is upper bounded by $l(k)$, where k is the security parameter, and $l()$ is some polynomial, which is a parameter of the definition. There is no bound on the amount of information that the

adversary may provide the environment. Let $\text{EXEC}^{p}_{\pi,\mathcal{A},\mathcal{Z}}$ be the same as $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$, with the exception that the environment is bounded as described here.

We note that bounded UC models a type of bounded concurrent composition and, among other things, implies security in the ordinary model of standalone computation. For sake of concreteness, we concentrate on $l(n) = O(n \log n)$, where n is the number of parties in the protocol instance. (This choice of the bounding function is rather arbitrary and is derived from the properties of the actual protocol described we analyze in this work.)

Definition 7 (Protocol Emulation: gUC and Bounded-UC). Let π and ϕ be protocols. We say that π emulates ϕ with gUC security (resp., with p-security) if for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that outputs a value in $\{0, 1\}$, we have $\text{EXEC}^{gUC}_{\phi,\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} \text{EXEC}^{gUC}_{\pi,\mathcal{A},\mathcal{Z}}$ (resp., $\text{EXEC}^{p}_{\phi,\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} \text{EXEC}^{p}_{\pi,\mathcal{A},\mathcal{Z}}$).

The definition of realizing ideal functionalities with UC security is generalized analogously.

Party Corruptions: The above model does not provide an explicit party corruption operation. Instead, we model corruption via a special `(corrupt)` message written by the adversary on the incoming communication tape of the corrupted ITI. The ITI's response is, in general, up to the program of the ITI. For the purpose of this work, we use the following conventions, which are intended to capture the standard Byzantine corruptions model:

1. Except for ideal protocols, an ITI that receives a `(corrupt)` message first writes a `(corrupted)` output on the subroutine output tapes of all the ITIs of which the corrupted ITI is a subroutine and then sends all its current state to the adversary. All future received inputs and subroutine outputs are reported to the adversary. All incoming communication is interpreted as instructions by the adversary to generate input or subroutine output to other ITIs; these instructions are carried out.
2. For ideal protocols, we instruct dummy ITIs to ignore `(corrupt)` messages, with the intention that these messages should be sent directly to the ideal functionality. There are no restrictions on the behavior of ideal functionalities upon receiving corrupt inputs; this behavior is part of the security specification of the task being captured.

Discussion:

- Clearly, if π gUC emulates ϕ, then it UC emulates ϕ. Similarly, if π UC emulates ϕ, then it p-UC emulates ϕ for any polynomial $p()$.
- While the dummy adversary remains universal in the case of gUC emulation, it does *not* in general hold in the case of bounded UC security.
- Say that a protocol instance is subroutine respecting if no descendant μ of an ITI of this protocol is also a descendant of an ITI μ' where μ' is not an ITI of this instance or a descendant thereof. We note that, as long as the analyzed protocol is subroutine respecting, gUC security is identical to UC security.

- For sake of concreteness, we assume that the function p bounding the communication of the environment in bounded-UC security is $l(n) = O(n \log n)$, where n is the number of parties. (This choice of $l(n)$ is rather arbitrary and is derived from the properties of the actual protocol in use.)

Universal Composition. Let ρ be a protocol that uses one or more instances of some protocol ϕ as a subroutine, and let π be a protocol that emulates ϕ with UC security. The composed protocol $\rho^{\pi/\phi}$ is constructed by modifying the program of ρ so that calls to ϕ are replaced by calls to π. Similarly, subroutine outputs coming from π are treated as subroutine outputs coming from ϕ. The universal composition theorem says that protocol $\rho^{\pi/\phi}$ behaves essentially the same as the original protocol ρ.

In the case of UC security, the theorem applies only to subroutine respecting protocols. In the case of gUC security, the restriction is dropped, and the theorem holds with respect to any protocol. In the case of bounded UC security, only limited forms of composability (such as nonconcurrent composability) are known. We do not state them here. That is:

Theorem 8 (Universal Composition [4]). *Let π, ϕ, ρ be protocols such that π emulates ϕ with gUC security* (*resp.*, *with UC security, and both π and ϕ are subroutine respecting*). *Then the protocol $\rho^{\pi/\phi}$ emulates ρ with gUC security* (*resp.*, *with UC security*). *In particular, if ρ realizes an ideal functionality $\mathcal{F}$ with gUC security* (*resp.*, *UC security*), *then so does $\rho^{\pi/\phi}$.*

2.3. *Multi-Realizing Ideal Functionalities*

Informally, a protocol π n-emulates protocol ϕ if running n concurrent instances of π, on potentially different inputs and by potentially different parties, amounts to emulating n independent instances of ϕ.

More precisely, we first concentrate on defining UC n-emulation. The cases of n-emulation with gUC and bounded UC security are dealt with later. As a first step, we consider environments $\mathcal{Z}$ that can interact with some number, n, of concurrent instances of the protocol in question. We call such environments n-instance environments. That is, an n-instance environment can invoke ITIs with up to n different SIDs. To facilitate the analysis, we also put the following syntactic restriction on the SIDs: All SIDs should be of the form (*sid*, *ssid*), where the first component *sid* is the same for all instances. We then modify the model of executing a protocol π so that the program of *all* the ITIs invoked by the environment, other than the adversary, is fixed (by the control function) to be π.

We now say that protocol π n-emulates protocol ϕ with UC security if for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{S}$ such that for any n-instance environment $\mathcal{Z}$, we have $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

If π n-emulates ϕ for any $n = \text{poly}(k)$, then we say that π poly-emulates ϕ. π n-realizes (poly-realizes an ideal functionality $\mathcal{F}$ if π n-emulates (poly-emulates) the ideal protocol $I_{\mathcal{F}}$.

n-emulation with gUC security is defined analogously. That is, we say that an ITI invoked (by the environment) is *special* if its program is set to be $\perp$. Then, we modify the gUC model of protocol execution so that the environment invokes special ITIs with

up to at most n different SIDs, and furthermore these SIDs comply with the technical restriction described above. The model of executing π then replaces the program of all these ITIs with π. The rest of the definition is the same as for the case of UC security.

n-emulation with bounded UC security is the same as n-emulation with UC security, with the exception that the communication from the environment to the adversary is restricted as in the case of plain emulation with bounded UC security.

We note that, as long as no two ITIs in different instances of protocol π have the same ITI as a subroutine, the UC theorem asserts that if π emulates protocol ϕ with UC security then π also poly-emulates ϕ. The same holds for gUC security.

3. Split Functionalities

We define what it means to realize an ideal functionality in an unauthenticated network without any setup. As sketched in the Introduction, realizing a functionality in the unauthenticated model is defined by requiring that the protocol actually realizes a relaxed variant of the functionality, called a "split functionality." (More motivational discussion on this choice appears there.) Recall that split functionalities enable the ideal-model adversary to split the uncorrupted parties into disjoint sets, called authentication sets, in an adaptive way. The parties in each authentication set H then run a separate ideal execution with the trusted party where in each execution the adversary plays the roles of all the parties not in H.

A little more precisely, we assume that the parties know in advance the set U of parties with which they wish to interact. In each execution of an authentication set H, the adversary plays the roles of the parties in $U - H$. (Throughout, we use the term "party P" to mean "an ITI with PID P", where the SID is clear from the context.) We wish to make the following guarantees:

1. An authentication set must be fixed before any computation in the set begins (and thus an authentication set cannot be chosen on the basis of the inputs of the uncorrupted parties in that set);
2. The authentication sets are disjoint. This guarantees that all the parties in an authentication set have consistent views of the interaction. In particular, each party participates in only one execution;
3. The computation within each set is secure in the standard sense, i.e., as in the case that authenticated channels are assumed. In particular it is independent of the computations in other sets, except for the inputs provided by the adversary, which can be correlated to the outputs that it has received from computations with other authentication sets that have already been completed.

For simplicity, we assume that the authentication sets given by the adversary do not include corrupted parties. Alternatively, we modify the disjointment requirement to say that the intersection of any two authentication sets contains only corrupted parties.

The Split Functionality $\mathsf{s}\mathcal{F}$. We now proceed to formalize the above. Let $\mathcal{F}$ be an ideal functionality. We define the relaxation of $\mathcal{F}$, called split-$\mathcal{F}$ or $\mathsf{s}\mathcal{F}$, in Fig. 1. For convenience, we let the SID of $\mathsf{s}\mathcal{F}$ explicitly contain the list of parties (i.e., PIDs) that the initiator of this instance wished to interact with.

Functionality $\mathsf{s}\mathcal{F}$

Given functionality $\mathcal{F}$, functionality $\mathsf{s}\mathcal{F}$ proceeds as follows:

Initialization:

1. Upon receiving an input $(\texttt{Init}, sid)$ from a party P (i.e., from an ITI with SID sid and PID P), interpret $sid = (U, sid')$, where U is a set of IDs that includes P. Send $(\texttt{Init}, sid, P)$ to the adversary.
2. Upon receiving a message $(\texttt{Init}, sid, P', H, sid_H)$ from the adversary, verify that $H \subseteq U$, that $P' \in H$, and that for all previously recorded sets H', it holds that either **(1)** $H \cap H'$ contains only corrupted parties and $sid_H \neq sid_{H'}$, or **(2)** $H = H'$ and $sid_H = sid_{H'}$. If any condition fails, then do nothing. Otherwise, record the pair (H, sid_H), output $(\texttt{Init}, sid, sid_H)$ to P', locally initialize a new instance of the original functionality $\mathcal{F}$ with session identifier sid_H, and provide this instance of $\mathcal{F}$, denoted $\mathcal{F}_H$, with the current set of corrupted parties. From now on let the adversary play the role of the parties in $U - H$ in $\mathcal{F}_H$.

Computation:

1. Upon receiving an input $(\texttt{Input}, sid, v)$ from party P', find the set H such that $P' \in H$ and forward the message v from P' to $\mathcal{F}_H$. If no such H is found, then ignore the input.
2. Upon receiving a message $(\texttt{Input}, sid, H, P', v)$ from the adversary, if $\mathcal{F}_H$ is initialized and $P' \in U - H$, then forward v to $\mathcal{F}_H$ as if coming from party P'. Otherwise, ignore the message.
3. When an instance $\mathcal{F}_H$ generates an output v for party $P \in H$, output v to P. When the output is for a party $P' \in U - H$ or for the adversary, send the output to the adversary.

Corruption:

1. Upon receiving a $(\texttt{corrupt}, P)$ message from the adversary, record P as corrupted and forward this message to all currently active instances of $\mathcal{F}$.

Fig. 1. The split version of ideal functionality $\mathcal{F}$.

In the initialization stage of $\mathsf{s}\mathcal{F}$, the adversary adaptively chooses subsets of uncorrupted parties. (The adaptivity relates to the fact that an authentication set can be chosen and even a full execution completed, before the next authentication set is chosen.) The adversary can choose any subsets that it wishes, as long as they are *disjoint* (or, rather, as long as the intersection of any two authentication sets contains only corrupted parties).

$\mathsf{s}\mathcal{F}$ requires the adversary to provide a unique identifier, sid_H, for each authentication set H. This identifier is used to differentiate between the various copies of $\mathcal{F}$. Furthermore, this identifier is outputted explicitly to all the parties in this set. This is an important security guarantee: while the parties do not know, of course, which are the authentication sets, they have "evidence" of the set they are in. In particular, a global entity that sees the outputs of all parties can determine the authentication sets from the outputs alone. (From a technical point of view, this provision forces the adversary in the ideal process to mimic the same partitioning to authentication sets as the adversary in the real protocol execution.)

In the computation stage of the functionality, each set H is provided with a different and independent copy of $\mathcal{F}$. This means that each set H essentially runs a separate ideal execution of $\mathcal{F}$. In each such execution, the parties $P \in H$ provide their own inputs, and the adversary provides the inputs for all $P' \in U - H$. This reflects the fact that in each execution, the roles of the parties *outside* of the authentication set are played by the adversary. Similarly, the parties $P \in H$ all receive their specified outputs as computed by their copy of $\mathcal{F}$. The adversary receives all of its own outputs, as well as the outputs of the parties $P \in U - H$. We stress that there is no sharing of state between the different copies of $\mathcal{F}$ run by $\mathsf{s}\mathcal{F}$.

Note that each instance of $\mathsf{s}\mathcal{F}$ has a session identifier, *sid*, which is separate from the identifiers of the various authentication sets. This convention makes sure that all participants agree on the set U of intended participants. It is also consistent with the UC framework, which requires each instance of a functionality or a protocol to have its own unique identifier.

Also, when initializing a new copy of $\mathcal{F}$, functionality $\mathsf{s}\mathcal{F}$ lets this copy know the current set of corrupted parties. This too is done for consistency with the UC framework, which models party corruption in the ideal process via special messages sent from the adversary to the ideal functionality.

4. Unauthenticated Secure Computation

We restate more formally the two main theorems from the Introduction. Here the term "well formed" is taken from [10]. It indicates some mild syntactic restrictions that rule out unrealistic functionalities that "abuse the model".[3] The remainder of this section is dedicated to proving the two theorems.

Theorem 9 (Theorem 1, Restated). *Assume existence of collision-resistant hash functions and enhanced trapdoor permutations. Then, for any well-formed probabilistic polynomial-time multiparty functionality $\mathcal{F}$, there exists a protocol that realizes the split functionality $\mathsf{s}\mathcal{F}$ with bounded-UC security, in the presence of static, malicious adversaries and with no setup whatsoever.*

Theorem 10 (Theorem 2, Restated). *Assume existence of enhanced trapdoor permutations. Then for any well-formed ideal functionality $\mathcal{F}$, there exists a protocol that realizes the split functionality $\mathsf{s}\mathcal{F}$ with UC security in the CRS model or, alternatively, with gUC security in the ACRS model, in the presence of static, malicious adversaries. Assuming existence of enhanced noncommitting encryption, this holds even with respect to adaptive adversaries and without data erasures.*

The rest of this section is dedicated to proving these two theorems.

[3] The restrictions are of three types: One type rules out functionalities that pass information from one party to another without allowing the adversary to delay the process. A second type forces revealing certain information to the adversary upon party corruption. A third type forces revealing some information on the inputs of the parties (such as their lengths) to the adversary.

4.1. *Proof Overview*

We first provide an overview of the proofs. The proofs consist of a number of steps, where all steps but one are joint for the two theorems.

Obtaining Link Initialization. The First step in the proof is to securely implement the link initialization stage sketched in the introduction. We start by defining the abstract properties needed from this link initialization protocol by means of an appropriate ideal functionality. This ideal functionality, called the split authentication functionality, $\mathcal{F}_{\mathrm{SA}}$, is in fact the split version of the standard authenticated message transmission functionality, $\mathcal{F}_{\mathrm{AUTH}}$. We show that the link initialization protocol sketched in the introduction realizes the split authentication functionality with UC security and no setup.

Multisession Security. The Second step in our proof asserts that, when *authenticated communication* is available, any multiparty functionality can be n-realized, where n is the number of parties that the functionality interacts with. This is done thrice: First we observe that, with no additional setup, for any well-formed functionality and any n, the [25] protocol n-realizes the functionality with bounded-UC security. This result follows directly from the proof in [25]. Next, we observe that the construction and proof in [5] imply that any well-formed functionality can be poly-realized in the ACRS model with gUC security. Finally, we demonstrate that the [10] protocol poly-realizes any well-formed functionality with UC security. To obtain this result, we somewhat strengthen the analysis there. (The analysis there does not apply to our setting since the [10] protocol uses a trusted common reference string (CRS), and therefore in a setting where multiple protocols sessions run concurrently, we have multiple sessions of the protocol that use the same joint subroutine (the reference string). This means that the UC composition theorem cannot be immediately used to argue multisession security.)

We note that this is the only step in the proof which is different for the cases of bounded-UC , UC, and gUC security. The remainder of the proofs are identical to each other.

Transformation from Multisession Protocols to Unauthenticated Protocols. The third step in the proof is a general transformation from any protocol π that poly- (resp., n-) realizes a functionality $\mathcal{F}$ with UC (resp., gUC, bounded-UC) security given authenticated communication, to a protocol Π_F that realizes $\mathsf{s}\mathcal{F}$ with UC (resp., gUC, bounded-UC) security given access to $\mathcal{F}_{\mathrm{SA}}$. On a high level, protocol Π_F proceeds as follows: Each party first runs $\mathcal{F}_{\mathrm{SA}}$ to obtain an identifier s, called an subsession identifier (SSID); next it runs protocol π with s as SID. Each outgoing message of π is forwarded to $\mathcal{F}_{\mathrm{SA}}$, and each incoming message from $\mathcal{F}_{\mathrm{SA}}$ is forwarded to π.

Putting it All Together. We combine the protocol Π_F and the link initialization protocol into a protocol that realizes $\mathsf{s}\mathcal{F}$ with UC (resp., gUC, bounded-UC) security in the CRS (resp., plain, ACRS) model. Security of the combined protocol is deduced using the universal composition theorem.

Functionality $\mathcal{F}_{\text{MAUTH}}$

1. Upon activation with input $(\texttt{Init}, sid)$, where $sid = (\mathcal{P}, sid')$ and $\mathcal{P}$ is a set of PIDs, record $\mathcal{P}$ and forward it to the adversary.
2. Upon receiving $(\texttt{send}, sid, P, P', m)$ from P, where $P, P' \in \mathcal{P}$, send (P, P', m) to the adversary and add (P, P', m) to an (initially empty) list $\mathcal{W}$ of waiting messages. Note that the same triple (P, P', m) can appear multiple times in the list.
3. Upon receiving $(\texttt{deliver}, sid, P, P', m)$ from the adversary, if P is in $\mathcal{P}$ and is currently corrupted, then output $(\texttt{received}, sid, P, P', m)$ to P'. Else, if there is a triple $(P, P', m) \in \mathcal{W}$, then remove one appearance of it from $\mathcal{W}$ and output $(\texttt{received}, sid, P, P', m)$ to P'. Otherwise do nothing.

Fig. 2. The multimessage authentication functionality $\mathcal{F}_{\text{MAUTH}}$.

4.2. *Realizing Split Authentication*

This section shows how to securely implement the link initialization stage, as sketched in the introduction. In order to present the construction in a modular way, we capture the requirements from the initialization stage via an ideal functionality. This ideal functionality is in fact the split functionality of the standard authenticated message transmission functionality. We thus call it the split authentication functionality, $\mathcal{F}_{\text{SA}}$.

We first present $\mathcal{F}_{\text{SA}}$. Next, we present a simple protocol that realizes $\mathcal{F}_{\text{SA}}$ with UC security *in the bare model,* without any setup.

4.2.1. *The Split Authentication Functionality $\mathcal{F}_{\text{SA}}$*

The split authentication functionality $\mathcal{F}_{\text{SA}}$ enables parties in the same authentication set to communicate in an authenticated way. That is, $\mathcal{F}_{\text{SA}}$ first lets the adversary define disjoint authentications sets H. Next, if the adversary wishes to deliver a message m to a party P' with an alleged sender P, then, roughly speaking, $\mathcal{F}_{\text{SA}}$ proceeds as follows:

1. If the authentication set H of P' is not yet determined (i.e., P' does not appear in any set H), then the delivery request is ignored. Otherwise:
2. If P is *not* in the same authentication set as P', then m is delivered as requested, regardless of whether it was actually sent by P.
3. If P and P' *are* in the same authentication set, then the message is delivered to P' only if it was sent by P, and has not yet been delivered.

More precisely, $\mathcal{F}_{\text{SA}}$ is the split version of the authentication functionality, $\mathcal{F}_{\text{MAUTH}}$, defined in Fig. 2. In other words, we define $\mathcal{F}_{\text{SA}} = \text{s}\mathcal{F}_{\text{MAUTH}}$. $\mathcal{F}_{\text{MAUTH}}$ is essentially the "multiple-session extension" of the standard authenticated message transmissions functionality, $\mathcal{F}_{\text{AUTH}}$, defined in [4]. That is, here a single instance of $\mathcal{F}_{\text{MAUTH}}$ handles multiple messages among parties in a given set $\mathcal{P}$ of PIDs. For concreteness, we assume that the set $\mathcal{P}$ is included in the SID of $\mathcal{F}_{\text{MAUTH}}$ and that each PID $P \in \mathcal{P}$ represents a full identity (i.e., SID and PID) of an ITI. (These ITIs are taken to be members of the calling protocol instance and will eventually receive output from $\mathcal{F}_{\text{SA}}$.)

For ease of reference, we also give an explicit formulation of $\mathcal{F}_{\text{SA}}$ in Fig. 3.

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

Initialization:

1. Upon activation with input $(\texttt{Init}, sid)$, where $sid = (\mathcal{P}, sid')$ and $\mathcal{P}$ is a set of PIDs, record $\mathcal{P}$ and forward it to the adversary.
2. When receiving $(\texttt{Init}, sid, P, H, sid_H)$ from the adversary, verify that party $P \in \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either $H \cap H'$ contains only corrupted parties and $sid_H \neq sid_{H'}$, or $H' = H$ and $sid_H = sid_{H'}$. If so, then output $(\texttt{Init}, sid, sid_H)$ to P and record (H, sid_H) if not yet recorded. Else, do nothing.

Message authentication:

1. When receiving $(\texttt{Send}, sid, P, P', m)$ from P, where $P' \in \mathcal{P}$, send (P, P', m) to the adversary and add (P, P', m) to an (initially empty) list $\mathcal{W}$ of waiting messages. Note that the same triple can appear multiple times in the list.
2. When receiving $(\texttt{Deliver}, sid, P, P', m)$ from the adversary, proceed as follows:
   (a) If P' did not previously receive an $(\texttt{Init}, sid, sid_H)$ output, then do nothing.
   (b) Else, if P is in the authentication set H of P', and P is uncorrupted, then: If there is a triple $(P, P', m) \in \mathcal{W}$, then remove one appearance of the triple from $\mathcal{W}$ and output $(\texttt{Received}, sid, P, P', m)$ to P'. Otherwise do nothing.
   (c) Else (i.e., P' received $(\texttt{Init}, sid, sid_H)$, and either P is corrupted or $P \notin H$), then output $(\texttt{Received}, sid, P, P', m)$ to P', regardless of $\mathcal{W}$.

Fig. 3. The split authentication functionality, $\mathcal{F}_{\mathrm{SA}}$.

4.2.2. *Realizing* $\mathcal{F}_{\mathrm{SA}}$

In this section, we present a simple protocol for realizing $\mathcal{F}_{\mathrm{SA}}$ in the bare model without any setup. The protocol that we present is actually UC-secure. This will be important in our final protocol, where the protocol for computing $\mathcal{F}_{\mathrm{SA}}$ runs alongside a general protocol for realizing some ideal functionality.

Our protocol uses a signature scheme that is existentially unforgeable against chosen-message attacks as in [18]. It does so in a way that is somewhat reminiscent of the technique used in [13] to construct nonmalleable encryption. On a high level, our protocol also resembles the weak Byzantine agreement protocol of [15] (although both the goal and the actual protocol are very different). The main idea of the protocol has already been described in the introduction. We therefore proceed directly to its description.

Protocol 1.
I. Link initialization:

1. Upon activation with input $(\texttt{Init}, P, sid)$, where P is the local PID, interpret $sid = (\mathcal{P}, sid')$ where $\mathcal{P}$ is a set of identities, and verify that $P \in \mathcal{P}$. If so, send $(\texttt{Init}, sid)$ to each party in $\mathcal{P}$. For notational simplicity, we let $\mathcal{P} = P_1, \ldots, P_n$, where $n = |\mathcal{P}|$.
2. Having received an incoming message $(\texttt{Init}, sid)$, party P_i interprets $sid = (\mathcal{P}, sid')$. If $P_i \notin \mathcal{P}$, then P_i aborts. Else:

(a) P_i chooses a key pair (VK_i, SK_i) for the signature scheme.
(b) P_i sends VK_i to all parties $P_j \in \mathcal{P}$ (recall that in an unauthenticated network, sending m to P_j only means that the message (P_i, P_j, m) is given to the adversary.)
(c) P_i waits until it receives keys from every $P_j \in \mathcal{P}$, $j \neq i$ (recall that these keys are actually received from the adversary and do not necessarily correspond to keys sent by other parties) and defines $sid_i = \langle (P_1, VK_1), \ldots, (P_n, VK_n) \rangle$.
(d) P_i computes $\sigma_i = \mathsf{Sign}_{SK_i}(sid_i)$ and sends $\alpha_i = (sid_i, \sigma_i)$ to all parties $P_j \in \mathcal{P}$.
(e) P_i waits until it receives an $\alpha_j = (sid_j, \sigma_j)$ message from every $P_j \in \mathcal{P}$, $j \neq i$. Then, P_i checks that for every j, $\mathsf{Verify}_{VK_j}(sid_j, \sigma_j) = 1$ and that $sid_1 = \cdots = sid_n$. If all of these checks pass, then P_i outputs $(\texttt{Init}, sid, sid_i)$.

II. Authenticating messages:

1. P_i initializes a counter c to zero.
2. When P_i has input $(\texttt{send}, sid, P_i, P_j, m)$, meaning that it wishes to send a message m to P_j, then it signs on m together with sid_i, the recipient identity, and the counter value. That is, P_i computes $\sigma = \mathsf{Sign}_{SK_i}(sid_i, m, P_j, c)$, sends (P_i, m, c, σ) to P_j, and increments c.
3. Upon receiving a message (P_j, m, c, σ) allegedly from P_j, party P_i first verifies that c did not appear in a message received from P_j in the past. It then verifies that σ is a valid signature on (sid_i, m, P_i, c), using the verification key VK_{i_j}. If the verification succeeds, then it outputs $(\texttt{received}, sid, P_j, P_i, m)$.

We have:

Theorem 11. *Assume that the signature scheme used in Protocol 1 is existentially secure against chosen-message attacks. Then, Protocol 1 realizes $\mathcal{F}_{\mathrm{SA}}$ with UC security in the presence of malicious, adaptive adversaries, and in the bare model with no setup whatsoever.*

Proof. We show that for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a probabilistic polynomial-time ideal-process adversary (i.e., a simulator) $\mathcal{S}$ such that no probabilistic polynomial-time environment $\mathcal{Z}$ can distinguish the case that it is interacting in the real world with adversary $\mathcal{A}$ and parties running Protocol 1, from the case that it is interacting in the ideal world with adversary $\mathcal{S}$ and the ideal functionality $\mathcal{F}_{\mathrm{SA}}$. The simulator $\mathcal{S}$ internally invokes $\mathcal{A}$ and perfectly simulates the uncorrupted parties interacting with $\mathcal{A}$. Then, when an uncorrupted party P_i in the internal simulation by $\mathcal{S}$ completes its link initialization phase and outputs $(\texttt{Init}, sid, sid_i)$, simulator $\mathcal{S}$ determines the set H of P_i to be the set of parties for which sid_i contain their "authentic" verification keys. Formally, we say that VK_j is P_j's authentic key if it is the key generated by P_j in the internal simulation by $\mathcal{S}$. Next, when $\mathcal{A}$ delivers a signed message to some P_i, simulator $\mathcal{S}$ asks $\mathcal{F}_{\mathrm{SA}}$ to deliver the message to P_i in the ideal process only if the internally simulated uncorrupted party would accept the signature, according to the protocol specification.

We now proceed to more formally specify $\mathcal{S}$. Simulator $\mathcal{S}$ internally invokes a copy of all the uncorrupted parties and runs an interaction between $\mathcal{A}$ and these simulated copies as follows:

1. All messages from the external $\mathcal{Z}$ to $\mathcal{S}$ are forwarded to the internal $\mathcal{A}$, and all messages that $\mathcal{A}$ wishes to send to $\mathcal{Z}$ are externally forwarded by $\mathcal{S}$ to $\mathcal{Z}$.
2. Whenever $\mathcal{A}$ delivers a message $(\texttt{Init}, sid)$ to an uncorrupted party $P_i \in \mathcal{P}$, $\mathcal{S}$ simulates the actions of P_i in the link initialization phase of Protocol 1.
3. Whenever an internally simulated uncorrupted party P_i completes the link initialization phase with output $(\texttt{Init}, sid, sid_i)$, simulator $\mathcal{S}$ determines the set H_i to be the set of uncorrupted parties P_j such that the authentic verification key sent by P_j is included in sid_i. (Recall that $\mathcal{S}$ internally runs all the uncorrupted parties, so it can determine whether or not the key VK_j that it chose for P_j is included in sid_i.) $\mathcal{S}$ then checks that for all previously computed sets H, it holds that either:

   - $H_i \cap H$ contains only corrupted parties, and $sid_{H_i} \neq sid_H$, or
   - $H_i = H$ and $sid_{H_i} = sid_H$.

   If this holds, then $\mathcal{S}$ sends $(\texttt{Init}, sid, P_i, H_i, sid_i)$ to $\mathcal{F}_{\text{SA}}$. Otherwise, $\mathcal{S}$ halts and outputs fail1.
4. Whenever $\mathcal{S}$ receives a message $(\mathsf{send}, sid, P_i, P_j, m)$ from $\mathcal{F}_{\text{SA}}$ where P_i is uncorrupted, simulator $\mathcal{S}$ simulates the actions of an uncorrupted P_i sending a message m in the authentication phase of Protocol 1.
5. Whenever an internally simulated party P_i outputs $(\mathsf{received}, sid, P_j, P_i, m)$ in the simulation, $\mathcal{S}$ works as follows. If P_j is corrupted, then $\mathcal{S}$ instructs P_j to send an appropriate send message to $\mathcal{F}_{\text{SA}}$. If P_j is uncorrupted but not in the same authentication set as P_i, then $\mathcal{S}$ sends the appropriate send message to $\mathcal{F}_{\text{SA}}$ itself. Then, $\mathcal{S}$ sends $\mathcal{F}_{\text{SA}}$ the message $(\mathsf{deliver}, P_j, P_i, m)$, instructing it to deliver m to P_i from P_j.[4] If the request is not fulfilled by $\mathcal{F}_{\text{SA}}$, then $\mathcal{S}$ halts and outputs fail2.
6. Whenever $\mathcal{A}$ corrupts a party P_i, simulator $\mathcal{S}$ hands $\mathcal{A}$ the state of the internally simulated uncorrupted party P_i.

It is straightforward to verify that as long as $\mathcal{S}$ does not output fail1 or fail2, the view of $\mathcal{Z}$ in the ideal model is identical to its view in a real execution of Protocol 1. This is due to the fact that unless a fail occurs, $\mathcal{S}$ just mimics the actions of the uncorrupted parties. In addition, the local outputs of the uncorrupted parties in the internal simulation correspond exactly to the outputs of the actual uncorrupted parties in the ideal model. It therefore suffices to show that $\mathcal{S}$ outputs a fail1 or fail2 message with at most negligible probability.

We first show that $\mathcal{S}$ outputs a fail1 message with at most negligible probability. Below, we refer only to uncorrupted parties in the authentication sets because $\mathcal{S}$ never includes corrupted parties in these sets. There are three events that could cause a fail1 message:

1. *There exist two uncorrupted parties P_i and P_j for whom $\mathcal{S}$ defines sets H_i and H_j such that $H_i = H_j$ and yet $sid_i \neq sid_j$:* In order to see that this event cannot occur with nonnegligible probability, notice that $\mathcal{S}$ only defines sets H_i and H_j for parties that conclude the Link Initialization portion of the protocol. Furthermore, it places P_i and P_j in the same set only if they received each others authentic

[4] Note that if P_j is uncorrupted and is in the same authentication set as P_i, then $\mathcal{S}$ does not begin with a send message, but rather immediately sends a deliver message.

verification keys. By this behavior of $\mathcal{S}$, we have that if H_i and H_j are defined, and $H_i = H_j$, then P_i received P_j's authentic key and vice versa. Now, P_j only concludes this phase with output if $sid_i = sid_j$ (where sid_j is the identifier it locally defined, and sid_i is the identifier it received from P_i) and if the verification of sid_i with the verification key of P_i passes. If the event we are considering here occurred with nonnegligible probability, then $sid_i \neq sid_j$, and so in the internal simulation by $\mathcal{S}$, we have that P_i never signed on the identifier that P_j received in the name of P_i. Thus, $\mathcal{A}$ must have forged a signature, and it can be used to break the signature scheme. (The formal reduction to the security of the signature scheme is straightforward and is therefore omitted.)

2. *There exist two sets $H_i \neq H_j$ whose intersection contain uncorrupted parties:* Let $P_i \in H_i \cap H_j$ be an uncorrupted party. Then, using the same arguments as above, except with negligible probability, P_i must have the same sid_i as all the uncorrupted parties in H_i *and* all the uncorrupted parties in H_j. Thus, all of the parties in $H_i \cup H_j$ have the same sid_i. Since this sid_i is comprised of the parties verification keys, it must hold that all parties in $H_i \cup H_j$ received each other's authentic verification keys. By the construction of $\mathcal{S}$, it therefore holds that $H_i = H_j$.
3. *There exist two sets $H_i \neq H_j$, and yet $sid_i = sid_j$:* We have already seen that by the construction of $\mathcal{S}$, if $sid_i = sid_j$, then $H_i = H_j$. This case therefore never occurs.

It remains to show that $\mathcal{S}$ outputs fail2 with at most negligible probability. This occurs if $\mathcal{S}$ sends a $(\texttt{deliver}, P_j, P_i, m)$ message to $\mathcal{F}_{\text{SA}}$ where P_i is uncorrupted, and the message is not actually delivered to P_i. By the definition of $\mathcal{F}_{\text{SA}}$ (and in general split functionalities), this can only occur if P_j is uncorrupted, and P_i and P_j are in the same authentication set H. (We ignore trivialities here like the case that H is not defined.) In order to see this, notice that if P_j is corrupted, then $\mathcal{S}$ first instructs it to send a send message to $\mathcal{F}_{\text{SA}}$, and so $\mathcal{S}$'s deliver message would not be ignored. The same is true in the case that P_i and P_j are *not* in the same authentication set (because then $\mathcal{S}$ first sends the send message itself). Now, if P_i and P_j *are* in the same authentication set, then they both hold each others "authentic" verification keys (as shown above). Furthermore, the deliver message of $\mathcal{S}$ is only ignored if P_j did not previously send an appropriate send message to $\mathcal{F}_{\text{SA}}$. Now, by the simulation strategy of $\mathcal{S}$, it only generates a signature on a message in the internal simulation if a send message was sent in the ideal world; see step 4 of $\mathcal{S}$. Furthermore, by step 5 of $\mathcal{S}$, it only issues a deliver message to $\mathcal{F}_{\text{SA}}$ if the internally simulated P_i outputs a received in the protocol simulation. Recall that this occurs only if P_i received a message together with a valid signature on that message.

Combining the above, we have that $\mathcal{S}$ can only output fail2 if:

1. No $(\text{send}, sid, P_j, P_i, m)$ message was issued in the ideal world, and thus $\mathcal{S}$ did not generate a signature (using P_j's authentic key) on the message (sid_i, m, P_i, c), *and*
2. $\mathcal{S}$ issues a $(\text{deliver}, sid, P_j, P_i, m)$ message, which only occurs if P_i received a valid signature on (sid_i, m, P_i, c) in the internal simulation.

It therefore follows that $\mathcal{A}$ must have forged a signature relative to the uncorrupted P_j's authentic key. As above, such an adversary can be used to break the signature, and the

actual reduction is straightforward. We note that in the case that the same message m is sent multiple times, the above argument remains the same while using the counter to argue that m cannot be delivered more times than it was sent. (Otherwise, there is a different message pair (m, c) that was never signed upon by $\mathcal{S}$.) We conclude that the views of $\mathcal{Z}$ in the two interactions are statistically close. □

4.3. *Realizing Multi-Session Functionalities with Authenticated Communication*

We show that, when authenticated communication is available, the "multisession version" of any ideal functionality can be realized by an appropriate "multisession protocol." Said otherwise, it is possible to multirealize any ideal functionality. This is proven thrice: once with bounded-UC security with no additional setup, once with UC security in the CRS model, and once with gUC security in the ACRS model.

We remark that, with respect to party corruptions, multisession security provides a somewhat stronger guarantee than needed: The multisession model allows the adversary to corrupt different sets of parties in different instances of the protocol under consideration. In contrast, for realizing split functionalities, we are only interested in the case where the same set of parties (i.e., the same set of PIDs) are corrupted in all sessions. Still, as shown in this section, known solutions do achieve this stronger property.

Multirealizing Functionalities with Bounded-UC Security. Although the formalism in [25] is quite different than the one here, the protocol there and its proof directly extend to give the following theorem. We note that this protocol can handle any polynomial bound on the amount of communication from the environment to the adversary, as long as the bound is known in advance; indeed, the protocol depends on this bound.

Theorem 12 [25]. *Assume the existence of collision-resistant hash functions and enhanced trapdoor permutations. Then, for any well-formed probabilistic polynomial-time ideal functionality $\mathcal{F}$ and for any n that is polynomial in the security parameter k, there exists a protocol π that n-realizes $\mathcal{F}$ with bounded-UC security, in the presence of static, malicious adversaries and with no setup other than authenticated communication.*

4.3.1. *Realizing Multisession Functionalities with UC and gUC Security*

Due to the impossibility of realizing large classes of ideal functionalities with UC (or gUC) security in the *plain model,* where the only available trusted set-up is authenticated communication [4,6,9], we resort to protocols that use some additional trusted set-up. Specifically, we consider two trusted set-up models: the common reference string (CRS) model and the augmented CRS (ACRS) model.

In the UC framework, the CRS model is formalized via an ideal functionality, $\mathcal{F}_{\text{CRS}}$, that proceeds as follows. $\mathcal{F}_{\text{CRS}}$ is parameterized by a probabilistic polynomial-time function D (which represents a sampling algorithm that induces the distribution of the reference string) and an SID that encodes the set of "legitimate users" of this instance of $\mathcal{F}_{\text{CRS}}$. Initially, $\mathcal{F}_{\text{CRS}}$ chooses a random value s, sets the reference string to be $S = D(s)$, and sends S to the adversary. It then returns S to any party that asks for the reference string *if the SID of the requesting party is a legitimate one.*

Restricting the set of legitimate recipients of the reference string plays a crucial role in the feasibility results in the CRS model. (Without it, protocols that use $\mathcal{F}_{\text{CRS}}$ would not be subroutine respecting, and thus the UC theorem should not apply to them.) However, this restriction results in a discrepancy between the formal modeling and the common view of a reference string as a "public piece of information" that is available to and usable by anyone, even by uncorrupted participants in arbitrary other protocols.

The ACRS model is geared toward avoiding this discrepancy. However, it does so at the price of some additional trust put in the ideal functionality. Specifically, the ACRS model is formalized via an ideal functionality, $\mathcal{F}_{\text{ACRS}}$, that interacts with *any* ITI (even ITIs wit ha different SID). It proceeds as follows. $\mathcal{F}_{\text{ACRS}}$ is parameterized by functions D and t. Similarly to $\mathcal{F}_{\text{CRS}}$, $\mathcal{F}_{\text{ACRS}}$ initially chooses a random secret s, sets the reference string to be $S = D(s)$, and sends S to the adversary. It then returns S to *any* party that asks for the reference string, regardless of the identity or program of that party. In addition, whenever a *corrupted* party with identity ID asks for its trapdoor, $\mathcal{F}_{\text{ACRS}}$ responds with $t(s, ID)$.

We give two general feasibility results, one using the $\mathcal{F}_{\text{CRS}}$ model, and one using the $\mathcal{F}_{\text{ACRS}}$ model. The results are incomparable: On the one hand, $\mathcal{F}_{\text{CRS}}$ is parameterized so that only the participants in a single instance of the analyzed protocol have access to the reference string, while $\mathcal{F}_{\text{ACRS}}$ allows any party in the system access to the reference string. On the other hand, $\mathcal{F}_{\text{ACRS}}$ is parameterized with a "trapdoor function" t that requires it to continue interacting with the adversary throughout the lifetime of the system. For notational simplicity, from now on the "CRS model" is intended to mean the model with access to $\mathcal{F}_{\text{CRS}}$ that gives the reference string only to the parties of a single instance of the analyzed protocol. The "ACRS model" means the model where all parties have access to a single instance of $\mathcal{F}_{\text{ACRS}}$.

Theorem 13. *Assume the existence of enhanced trapdoor permutations and noncommitting encryption. Then, for any well-formed probabilistic polynomial-time ideal functionality $\mathcal{F}$, there exists a protocol π that poly-realizes $\mathcal{F}$ with UC security, in the CRS model and assuming authenticated communication in the presence of adaptive, malicious adversaries. Furthermore, protocol π is such that all instances of π use only a single instance of $\mathcal{F}_{\text{CRS}}$.*

Theorem 14. *Assume the existence of enhanced trapdoor permutations and noncommitting encryption. Then, for any well-formed probabilistic polynomial-time ideal functionality $\mathcal{F}$, there exists a protocol π that poly-realizes $\mathcal{F}$ with gUC security, in the ACRS model and assuming authenticated communication in the presence of adaptive, malicious adversaries.*

Theorem 14 follows immediately from the general feasibility result of [5]. This is so since, by the UC theorem, n-realizing $\mathcal{F}$ with gUC security is equivalent to simply realizing $\mathcal{F}$ with gUC security. Proving Theorem 13 is somewhat more involved.

Proof of Theorem 13. We start from the general feasibility result in [10]:

Theorem 15 [10]. *Assume the existence of enhanced trapdoor permutations and noncommitting encryption. Then, for any well-formed ideal functionality $\mathcal{F}$, there exists a*

protocol that realizes $\mathcal{F}$ with UC security in the CRS model, assuming authenticated communication and in the presence of adaptive, malicious adversaries.

Let $\text{CLOS}_{\mathcal{F}}$ denote the protocol in [10] for realizing functionality $\mathcal{F}$. As mentioned above, here $\mathcal{F}_{\text{CRS}}$ provides the reference string only to parties of a single instance of $\text{CLOS}_{\mathcal{F}}$ (This is essential for guaranteeing that $\text{CLOS}_{\mathcal{F}}$ is subroutine respecting. In fact, as shown in [5], the result of Theorem 15 would be impossible if $\mathcal{F}_{\text{CRS}}$ were to provide the reference string to other parties.) Consequently, Theorem 13 cannot be derived via a simple application of Theorem 15. In particular, Theorem 15 provides no security guarantees for the case where multiple instances of $\text{CLOS}_{\mathcal{F}}$ use the same reference string (namely, the same instance of $\mathcal{F}_{\text{CRS}}$). Instead, we observe that the specific structure of $\text{CLOS}_{\mathcal{F}}$ guarantees that $\text{CLOS}_{\mathcal{F}}$ actually poly-realizes $\mathcal{F}$ (namely, multiple instances of $\text{CLOS}_{\mathcal{F}}$ emulate multiple independent instances of $\mathcal{F}$), even when all instances of $\text{CLOS}_{\mathcal{F}}$ use the same instance of $\mathcal{F}_{\text{CRS}}$.[5]

Specifically, recall that a key component in $\text{CLOS}_{\mathcal{F}}$ is the multi-instance commitment functionality, $\mathcal{F}_{\text{MCOM}}$. In fact, $\text{CLOS}_{\mathcal{F}}$ can be viewed as consisting of two separate parts:

Low-level component: A protocol π_c for realizing $\mathcal{F}_{\text{MCOM}}$ assuming access to $\mathcal{F}_{\text{CRS}}$. This protocol provides the functionality of multiple independent ideal commitments, by arbitrarily chosen sets of parties. This is done using a single instance of $\mathcal{F}_{\text{CRS}}$.

High-level component: A protocol Π_F for realizing any functionality assuming access to $\mathcal{F}_{\text{MCOM}}$. This protocol does not access $\mathcal{F}_{\text{CRS}}$ at all.

In other words, $\text{CLOS}_{\mathcal{F}} = \Pi_{\mathcal{F}}^{\pi_c}$. Furthermore, $\mathcal{F}_{\text{MCOM}}$ has the property that having access to multiple instances of $\mathcal{F}_{\text{MCOM}}$ has the same effect as having access to a single instance of $\mathcal{F}_{\text{MCOM}}$. Indeed, in both cases the provided functionality is that of multiple independent ideal commitments. Also, protocol π_c has the property that each commitment is handled by a separate subroutine such that the subroutines share no state other than the joint access to $\mathcal{F}_{\text{CRS}}$.

We conclude that for any n, running n independent instances of $\text{CLOS}_{\mathcal{F}}$, where each instance uses a different instance of $\mathcal{F}_{\text{MCOM}}$, has the same effect as running the same n instances of $\text{CLOS}_{\mathcal{F}}$, where all instances use the same joint instance of $\mathcal{F}_{\text{MCOM}}$. We can then use the UC theorem to replace this joint instance of $\mathcal{F}_{\text{MCOM}}$ by a (single instance of) a protocol that realizes $\mathcal{F}_{\text{MCOM}}$. We thus obtain a protocol that uses a single instance of $\mathcal{F}_{\text{CRS}}$ and has the same effect as n independent instances of $\text{CLOS}_{\mathcal{F}}$. In particular, this protocol realizes n independent instances of $\mathcal{F}$.

This argument can made more formal via the universal composition with joint state (JUC) theorem [11]. Recall that the multisession extension $\hat{\mathcal{F}}$ of an ideal functionality $\mathcal{F}$ is the functionality that provides the same interface as that of multiple independent instances of $\mathcal{F}$. (Technically, since all these instances of $\mathcal{F}$ are handled within the same instance of $\hat{\mathcal{F}}$, they all must use the same SID. They are differentiated using another identifier, called the subsession identifier, SSID.) The JUC theorem says the following:

[5] A natural way to determine whether a protocol π uses a CRS where each instance of π uses a different instance of the CRS, or alternatively whether all instances of π use the same CRS, is via the SID of $\mathcal{F}_{\text{CRS}}$. For instance, if π calls $\mathcal{F}_{\text{CRS}}$ with an some fixed SID, then this would result in having all instances of π use the same instance of $\mathcal{F}_{\text{CRS}}$. Alternatively, if π calls $\mathcal{F}_{\text{CRS}}$ with an SID that includes also the SID of the current instance of π, then this would result in invoking a new instance of $\mathcal{F}_{\text{CRS}}$ for each instance of π.

Let π be a protocol that uses multiple independent instances of functionality $\mathcal{F}$, and let ρ be a protocol that realizes $\hat{\mathcal{F}}$ with UC security. Then the protocol, denoted $\pi^{[\rho]}$, where all the calls of π to all the instances of $\mathcal{F}$ are replaced by calls to a *single instance* of ρ, emulates the original protocol π with access to multiple instances of $\mathcal{F}$.

To use the JUC theorem, we observe that the multisession extension of $\mathcal{F}_{\text{MCOM}}$ is essentially the same functionality as a single instance of $\mathcal{F}_{\text{MCOM}}$. (Technically, the multisession extension of $\mathcal{F}_{\text{MCOM}}$ has an additional SID, but this SID can be chosen to be some fixed known value, say 0.) It follows that protocol π_c defined above realizes also the multisession extension $\hat{\mathcal{F}}_{\text{MCOM}}$. It then follows from the JUC theorem that for any protocol α that uses multiple instances of $\mathcal{F}_{\text{MCOM}}$, the protocol $\alpha^{[\pi_c]}$ emulates α. Now, fix some ideal functionality $\mathcal{F}$, and let $\hat{\pi}_F$ denote the protocol that runs multiple instances of π_F, where each instance of π_F uses a dedicated instance of $\mathcal{F}_{\text{MCOM}}$. We now have:

1. Protocol $\hat{\pi}_F^{[\pi_c]}$ describes multiple instances of the [10] construction, where all instances use the same instance of $\mathcal{F}_{\text{MCOM}}$ and thus the same instance of $\mathcal{F}_{\text{CRS}}$.
2. From the JUC theorem we have that protocol $\hat{\pi}_F^{[\pi_c]}$ emulates protocol $\hat{\pi}_F$.
3. From the analysis of [10] we have that protocol $\hat{\pi}_F$ realizes $\hat{\mathcal{F}}$.
4. From the transitivity of emulation we have that $\hat{\pi}_F^{[\pi_c]}$ realizes $\hat{\mathcal{F}}$.

We conclude that protocol $\text{CLOS}_{\mathcal{F}}$ poly-realizes $\mathcal{F}$ with UC security, even when all instances of $\text{CLOS}_{\mathcal{F}}$ use the same instance of $\mathcal{F}_{\text{CRS}}$. □

4.4. *Realizing General Split Functionalities*

This section presents a general transformation from any protocol $\pi_{\mathcal{F}}$ that n-realizes $\mathcal{F}$ given authenticated communication to a protocol $\Pi_{\mathcal{F}}$ that realizes $\mathsf{s}\mathcal{F}$ given access to $\mathcal{F}_{\text{SA}}$. More precisely, we assume that all participants of each instance of $\pi_{\mathcal{F}}$ use a single instance of $\mathcal{F}_{\text{MAUTH}}$ for all communications. This instance has SID $sid \circ 0$, where sid is the SID of the current instance of $\pi_{\mathcal{F}}$.

The transformation preserves access to the underlying setup. That is, the transformed protocol $\Pi_{\mathcal{F}}$ uses an underlying ideal functionality $\mathcal{G}$ only if $\pi_{\mathcal{F}}$ uses $\mathcal{G}$. Furthermore, the transformation uses protocol $\pi_{\mathcal{F}}$ and its potential subroutines (except $\mathcal{F}_{\text{MAUTH}}$) in a "black-box way"; namely, Π_F accesses only the inputs and outputs of $\pi_{\mathcal{F}}$, as well as the interface of $\pi_{\mathcal{F}}$ with $\mathcal{F}_{\text{MAUTH}}$.

Protocol $\Pi_{\mathcal{F}}$ is quite simple: each party first runs $\mathcal{F}_{\text{SA}}$ to obtain an SSID $ssid$. Next it runs protocol $\pi_{\mathcal{F}}$ with the SID of $\pi_{\mathcal{F}}$ set to $ssid$. Each outgoing message of $\pi_{\mathcal{F}}$ is forwarded to $\mathcal{F}_{\text{SA}}$, and each incoming message from $\mathcal{F}_{\text{SA}}$ is forwarded to $\pi_{\mathcal{F}}$. For simplicity, we assume that $\mathcal{F}$ (and $\pi_{\mathcal{F}}$) comply with the convention that the set $\mathcal{P}$ of participating PIDs in an instance is included in the SID. (This in particular means that $\mathcal{P}$ is fixed at the onset of the protocol.) The protocol is given in Fig. 4.

Lemma 4.1. *Let $\mathcal{G}$ be a (setup) functionality, let $\mathcal{F}$ be an n-party ideal functionality, and let π_F be a protocol that securely n-realizes $\mathcal{F}$ with bounded-UC (resp. UC, gUC) security, using authenticated communication (i.e., $\mathcal{F}_{\text{MAUTH}}$) and an instance of $\mathcal{G}$. Then, protocol Π_F in Fig. 4 uses only an instance of $\mathcal{F}_{\text{SA}}$ and an instance of $\mathcal{G}$, and securely realizes the split functionality $\mathsf{s}\mathcal{F}$ with bounded-UC (resp. UC, gUC) security.*

Protocol $\Pi_{\mathcal{F}}$

Let $\pi_{\mathcal{F}}$ be a protocol as described above that has access to functionality $\mathcal{G}$ and assumes authenticated communication. Protocol $\Pi_{\mathcal{F}}$ proceeds as follows, using ideal access to $\mathcal{F}_{\text{SA}}$ and $\mathcal{G}$.

1. Upon receiving input $(\texttt{Init}, P, sid)$, where P is a PID, party P calls $\mathcal{F}_{\text{SA}}$ with input $(\texttt{Init}, P, sid \circ 1)$. (That is, P concatenates 1 to sid to create a different SID.)
2. Upon receiving output $(\texttt{Init}, sid \circ 1, sid_H)$ from $\mathcal{F}_{\text{SA}}$, store sid_H and output $(\texttt{Init}, sid, sid_H)$ to ITI P. (Here we use the convention that the PID P encodes a full identity (i.e., SID and PID) of an ITI.)
3. Upon receiving $(\texttt{Input}, sid, x)$, party P_i inputs x to the ITI running $\pi_{\mathcal{F}}$, with SID (sid, sid_H) and PID P_i.
4. Upon receiving output $(\texttt{received}, sid, P_j, P_i, m)$ from $\mathcal{F}_{\text{SA}}$, P_i forwards (P_j, P_i, m) to $(\pi_{\mathcal{F}}, (sid, sid_H), P_i)$, i.e., to the ITI running $\pi_{\mathcal{F}}$, with SID (sid, sid_H) and PID P_i, as if coming from $\mathcal{F}_{\text{MAUTH}}$.
5. When the ITI $(\pi_{\mathcal{F}}, (sid, sid_H), P_i)$ sends the message (P_i, P_j, m) to party P_j using $\mathcal{F}_{\text{MAUTH}}$, provide the input $(\texttt{send}, sid, P_i, P_j, m)$ to $\mathcal{F}_{\text{SA}}$.
6. Finally, whenever the ITI $(\pi_{\mathcal{F}}, (sid, sid_H), P_i)$ output a value y, outputs y.

Fig. 4. The protocol $\Pi_{\mathcal{F}}$ for realizing $\mathsf{s}\mathcal{F}$ in the unauthenticated model, given protocol $\pi_{\mathcal{F}}$ for multirealizing $\mathcal{F}$ with authenticated communication.

Proof. Assume that for any adversary $\mathcal{A}'$, there exists an ideal-process adversary (i.e., a simulator) $\mathcal{S}'$ such that no environment $\mathcal{Z}'$ that invokes up to n protocol instances, and complies with the restrictions of bounded-UC (resp., UC, gUC) security, can tell with nonnegligible probability whether it is interacting with parties running protocol π_F and adversary $\mathcal{A}'$, or with $\mathcal{F}$ and simulator $\mathcal{S}'$, i.e.,

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'} \stackrel{\text{c}}{\equiv} \text{EXEC}_{\pi_F,\mathcal{A}',\mathcal{Z}'}.$$

We show that $\Pi_{\mathcal{F}}$ realizes $\mathsf{s}\mathcal{F}$ with UC (resp, gUC, bounded-UC) security. That is, we show that for any adversary $\mathcal{A}$, there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ that complies with the restrictions of bounded-UC (resp., UC, gUC) security can tell with nonnegligible probability whether it is interacting with parties running protocol Π_F and adversary $\mathcal{A}$, or with $\mathsf{s}\mathcal{F}$ and simulator $\mathcal{S}$. That is, we show that

$$\text{IDEAL}_{\mathsf{s}\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{\text{c}}{\equiv} \text{EXEC}^{\mathcal{F}_{\text{SA}}}_{\Pi_F,\mathcal{A},\mathcal{Z}}.$$

The construction of $\mathcal{S}$ and its analysis proceed in a number of steps. We note that the same simulator $\mathcal{S}$ works for UC, gUC, and bounded UC security.

1. Given an adversary $\mathcal{A}$ that interacts with protocol Π_F, running on *unauthenticated* channels, we construct an adversary $\mathcal{A}'$ that interacts with protocol π_F running on *authenticated* channels. Furthermore, whereas $\mathcal{A}$ only interacts with a singe instance of protocol Π_F, $\mathcal{A}'$ might interact with up to n concurrent executions of the protocol π_F.
2. Since π_F securely n-realizes $\mathcal{F}$ (i.e., securely realizes $\mathcal{F}$ also under n concurrent executions), we deduce the existence of some ideal-model adversary $\mathcal{S}'$ such that the output of any bounded-UC (resp., UC, gUC), n-instance environment $\mathcal{Z}'$ that

interacts with $\mathcal{S}'$ and multiple instances of $\mathcal{F}$ is indistinguishable from the output of $\mathcal{Z}'$ when interacting with $\mathcal{A}'$ and π_F.

3. Next we transform the (concurrent) ideal model adversary $\mathcal{S}'$ into an ideal-model adversary $\mathcal{S}$ for the ideal execution of $\mathsf{s}\mathcal{F}$ (i.e., the ideal *unauthenticated execution*).
4. We show that the output of any bounded-UC (resp., UC, gUC), environment $\mathcal{Z}$ that interacts with $\mathcal{S}$ and $\mathsf{s}\mathcal{F}$ over unauthenticated communication is indistinguishable from the output of $\mathcal{Z}$ when interacting with $\mathcal{A}$ and Π_F. This is done as follows: Given an environment $\mathcal{Z}$ (that expects to interact with $\mathcal{A}$ and Π_F), we construct an n-instance environment $\mathcal{Z}'$ (that expects to interact with $\mathcal{A}'$ and π_F) such that $\mathcal{Z}'$ distinguishes between an interaction with $\mathcal{A}'$ in π_F and an interaction with $\mathcal{S}'$ and $\mathcal{F}$ with essentially the same probability that $\mathcal{Z}$ distinguishes between an interaction with $\mathcal{A}$ in Π_F and an interaction with $\mathcal{S}$ and $\mathsf{s}\mathcal{F}$. Furthermore, if $\mathcal{Z}$ complies with the restrictions of bounded-UC (resp., UC, gUC) security, then so does $\mathcal{Z}'$.

The Authenticated Concurrent Execution. We first describe the construction of adversary $\mathcal{A}'$, given adversary $\mathcal{A}$. To give a better intuitive picture of the proof, we also present the construction of the environment $\mathcal{Z}'$ along with the construction of $\mathcal{A}'$, in spite of the fact that $\mathcal{Z}'$ appears only later in the course of the proof (see the above outline).

The interaction of environment $\mathcal{Z}'$, adversary $\mathcal{A}'$, and protocol π_F perfectly mimics the interaction of $\mathcal{Z}$, $\mathcal{A}$, and protocol Π_F. In particular, $\mathcal{Z}'$ and $\mathcal{A}'$ internally run $\mathcal{Z}$ and $\mathcal{A}$ and internally emulate $\mathcal{F}_{\mathrm{SA}}$ for them. Note however that whereas $\mathcal{Z}$ only invokes *one* execution of the protocol Π_F, $\mathcal{Z}'$ might instead invoke up to n different concurrent execution.

In order to succeed, $\mathcal{Z}'$ and $\mathcal{A}'$ need to "coordinate their actions." They do so by sending messages between each other. As a convention, we let all such "internal" message sent between $\mathcal{Z}'$ and $\mathcal{A}'$ begin with the string `Internal`; for simplicity, we furthermore assume that $\mathcal{Z}$ and $\mathcal{A}$ never communicate using messages that begin with this prefix. Looking forward, these "internal messages" will also be useful to us for constructing the ideal unauthenticated adversary S. On an intuitive level, these messages correspond to the "interface" of $\mathsf{s}\mathcal{F}$. In order to ensure that this proof also holds for the setting of bounded-UC, we also need to make sure that there is a fixed upper bound on the length of all messages sent by $\mathcal{Z}'$ to $\mathcal{A}'$. We proceed with the description of $\mathcal{Z}'$ and $\mathcal{A}'$.

Constructing $\mathcal{Z}'$. For any given environment $\mathcal{Z}$, define the environment $\mathcal{Z}'$ that internally incorporates $\mathcal{Z}$ and proceeds as follows.

Outgoing communication from $\mathcal{Z}'$:

1. Whenever $\mathcal{Z}$ wishes to give input $(\texttt{Init}, sid)$ to party P of session sid of $\pi_{\mathcal{F}}$ (i.e., to an ITI running $\pi_{\mathcal{F}}$ with PID P_i and SID sid), $\mathcal{Z}'$ forwards the message $(\texttt{Internal-Init}, P, sid)$ to the adversary $\mathcal{A}'$.
2. Whenever $\mathcal{Z}$ wishes to give input $(\texttt{Input}, sid, x)$ to party P of session sid of $\pi_{\mathcal{F}}$, $\mathcal{Z}'$ verifies that it has previously stored a tuple (P, sid, sid_H, H) for some

sid_H. If so, then $\mathcal{Z}'$ inputs x to party P of session sid of $\pi_{\mathcal{F}}$. Otherwise $\mathcal{Z}'$ ignores and continues to run $\mathcal{Z}$.
3. Whenever $\mathcal{Z}$ wishes to input any other value to P, $\mathcal{Z}'$ ignores and continues to run $\mathcal{Z}$.
4. Whenever $\mathcal{Z}$ wishes to input any value to the adversary $\mathcal{A}$, $\mathcal{Z}'$ inputs this value unmodified to $\mathcal{A}'$.
5. Finally, whenever $\mathcal{Z}$ halts outputting z, $\mathcal{Z}'$ also halts outputting z.

Observe that the messages sent by $\mathcal{Z}'$ to $\mathcal{A}'$ are comprised of the `Internal-Init` messages and the messages that the original $\mathcal{Z}$ sends to $\mathcal{A}$. The overall length of the `Internal-Init` messages is at most $n \log n$, where n is the number of parties (this is due to the fact that for n executions, it suffices to take identifiers sid of length $\log n$). This implies that if the overall communication from the original $\mathcal{Z}$ to $\mathcal{A}$ is $O(n \log n)$, then so is the overall communication from $\mathcal{Z}'$ to $\mathcal{A}'$. Thus, if $\mathcal{Z}$ and $\mathcal{A}$ are in the setting of bounded-UC, then so are $\mathcal{Z}'$ and $\mathcal{A}'$.

Incoming message to $\mathcal{Z}'$:

1. Whenever $\mathcal{Z}'$ receives an output of the type $(\texttt{Internal-Init}, sid, P, H, sid_H)$ from $\mathcal{A}'$, it stores the tuple (P, sid, sid_H, H) and feeds the tuple $(\texttt{Init}, sid, sid_H)$ to $\mathcal{Z}$ as if it came from P with session id sid.
2. All other messages that $\mathcal{Z}'$ receives from $\mathcal{A}'$ are directly forwarded to $\mathcal{Z}$.
3. Any output received from party P with session id (sid, sid_H) is forwarded unmodified to $\mathcal{Z}$ as if it came from P with session id sid.

Constructing $\mathcal{A}'$. For any given adversary $\mathcal{A}$ that interacts with $\Pi_{\mathcal{F}}$, define the adversary $\mathcal{A}'$, which interacts with $\pi_{\mathcal{F}}$. (Recall that $\pi_{\mathcal{F}}$ uses $\mathcal{F}_{\text{MAUTH}}$, so $\mathcal{A}'$ can cause messages to be delivered only by interacting with $\mathcal{F}_{\text{MAUTH}}$.) $\mathcal{A}'$ internally runs an instance of $\mathcal{A}$, plus an instance of $\mathcal{F}_{\text{SA}}$. All messages that $\mathcal{F}_{\text{SA}}$ wishes to send to the adversary are directly forwarded to $\mathcal{A}$. Likewise, all messages that $\mathcal{A}$ wishes to send to $\mathcal{F}_{\text{SA}}$ are directly forwarded to this emulated version of $\mathcal{F}_{\text{SA}}$. More precisely:

1. Upon receiving $(\texttt{Internal-Init}, P, sid)$ from the environment, $\mathcal{A}'$ locally initializes an instance of $\mathcal{F}_{\text{SA}}$ and feeds it with input $(\texttt{Init}, sid \circ 1)$ as if it came from party (P, sid). Messages from $\mathcal{F}_{\text{SA}}$ to the adversary are delivered to $\mathcal{A}$.
2. Whenever $\mathcal{A}$ sends $(\texttt{Init}, sid \circ 1, P, H, sid_H)$ to $\mathcal{F}_{\text{SA}}$, $\mathcal{A}'$ forwards this message to the internal instance of $\mathcal{F}_{\text{SA}}$. When this local instance of $\mathcal{F}_{\text{SA}}$ outputs $(\texttt{Init}, sid \circ 1, H, sid_H)$ to P, $\mathcal{A}'$ outputs $(\texttt{Internal-Init}, sid, P, H, sid_H)$ to the environment.
3. Whenever $\mathcal{A}$ wishes to send $(\texttt{deliver}, sid \circ 1, P, P', m)$ to $\mathcal{F}_{\text{SA}}$, $\mathcal{A}'$ forwards this message to $\mathcal{F}_{\text{SA}}$. When $\mathcal{F}_{\text{SA}}$ outputs $(\texttt{Received}, sid \circ 1, P, P', m)$ to P', $\mathcal{A}$ sends the message $(\texttt{deliver}, sid \circ 0, P, P', m)$ to $\mathcal{F}_{\text{MAUTH}}$.
4. Whenever $\mathcal{A}$ wishes to corrupt a party (P, sid) in $\Pi_{\mathcal{F}}$, $\mathcal{A}'$ forwards this corruption instruction to the internal instance of $\mathcal{F}_{\text{SA}}$ and corrupts (P, sid) in $\pi_{\mathcal{F}}$.
5. Whenever $\mathcal{A}$ wishes to deliver any other message to an uncorrupted party, the message is ignored.
6. When receiving a message (P, P', m) from $\mathcal{F}_{\text{MAUTH}}$, $\mathcal{A}'$ feeds the local instance of $\mathcal{F}_{\text{SA}}$ with input $(\texttt{Send}, sid \circ 1, P, P', m)$ coming from P.

It directly follows from the construction of $\mathcal{A}'$ and $\mathcal{Z}'$ that the view of $\mathcal{Z}$ running within $\mathcal{Z}'$ is distributed identically to the view of $\mathcal{Z}$ in the execution of Π_F. That is,

$$\text{EXEC}^{\mathcal{F}_{\text{SA}}}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}} = \text{EXEC}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}'}. \tag{1}$$

The Ideal Concurrent Execution. Since protocol $\pi_{\mathcal{F}}$ n-realizes $\mathcal{F}$, it follows that for every adversary $\mathcal{A}'$, there exists some ideal-model adversary $\mathcal{S}'$ such that for every environment $\mathcal{Z}''$ that is allowed to open up at most n sessions of $\mathcal{F}$, it holds that

$$\text{EXEC}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}''} \stackrel{\text{c}}{\equiv} \text{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}''}.$$

This, in particular, implies that for every $\mathcal{Z}$ that opens at most one session of $\mathcal{F}$, it holds that

$$\text{EXEC}_{\pi_F,\mathcal{A}',\mathcal{Z}'} \stackrel{\text{c}}{\equiv} \text{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'}, \tag{2}$$

where $\mathcal{Z}'$ is as defined above.

The Ideal Unauthenticated Execution. We finally construct the simulator $\mathcal{S}$ out of the simulator $\mathcal{S}'$. Essentially, $\mathcal{S}$ mimics for $\mathcal{Z}$ an execution within $\mathcal{Z}'$, when $\mathcal{Z}'$ interacts with $\mathcal{S}'$ and multiple concurrent instances of $I_{\mathcal{F}}$. More precisely, consider the following simulator $\mathcal{S}$ for the ideal unauthenticated model. $\mathcal{S}$ internally incorporates $\mathcal{S}'$ and proceeds as follows.

1. Whenever $\mathcal{S}$ receives a message $(\texttt{Init}, sid, P)$ from $\mathsf{s}\mathcal{F}$, it forwards $(\texttt{Internal-Init}, P, sid)$ to $\mathcal{S}'$ (as if it came from the environment).
2. Whenever $\mathcal{S}'$ wishes to send the message $(\texttt{Internal-Init}, sid, P, H, sid_H)$ to the environment, $\mathcal{S}$ sends the message $(\texttt{Init}, sid, P, H, sid_H)$ to $\mathsf{s}\mathcal{F}$.
3. Whenever $\mathcal{S}'$ wishes to send message x to the instance of $\mathcal{F}$ with session id (sid, sid_H), $\mathcal{S}$ sends $(\texttt{Input}, sid, H, x)$ to $\mathsf{s}\mathcal{F}$. (The set H associated with sid_H can be determined from the previous $(\texttt{Internal-Init})$ outputs of $\mathcal{S}'$.)
4. When receiving from $\mathsf{s}\mathcal{F}$ an output v directed at party P in instance H, $\mathcal{S}$ forwards to $\mathcal{S}'$ an output v for party P, coming from $\mathcal{F}_H$. (Note that in this case P is corrupted.)
5. When $\mathcal{S}'$ corrupts party P (by sending $(\texttt{corrupt}, P)$ messages to the instances of $\mathcal{F}$), $\mathcal{S}$ sends a $(\texttt{corrupt}, P)$ message to $\mathsf{s}\mathcal{F}$. (Note that, by the validity of $\mathcal{S}'$, we have that $\mathcal{S}'$ must corrupt the same PID P in all instances of $\mathcal{F}$ together, except for negligible probability.) The response of $\mathsf{s}\mathcal{F}$ is then forwarded to $\mathcal{S}'$.

Analysis of $\mathcal{S}$. We claim that the output of $\mathcal{Z}, \mathcal{S}$ in the $\mathsf{s}\mathcal{F}$-hybrid model is statistically close to the output of $\mathcal{Z}', \mathcal{S}'$ in the concurrent $\mathcal{F}$-hybrid model. The only difference between those executions is that $\mathsf{s}\mathcal{F}$ might ignore certain inputs and not forward them to the respective instance of $\mathcal{F}$. This happens only if one of the following two events occur:

1. Upon receiving a message from an uncorrupted party that is not part of any specified set H.
2. Upon receiving a message $(\texttt{Init}, sid, P_i, H, sid_H)$ from the adversary, that is "mal-formed." This happens if:

(a) P_i has not previously sent an $(\texttt{Init}, sid)$ message,
(b) P_i is not part of the set H,
(c) $\mathsf{s}\mathcal{F}$ has previously received another pair $(sid_{H'}, H')$ such that either (1) $H \neq H'$, and their intersection contains uncorrupted parties, or (2) H and H' are equal, but $sid_H \neq sid_{H'}$.

Observe that inputs of the first type (i.e., messages from an uncorrupted party that is not part of any specified set H) are ignored also in the execution of $\mathcal{Z}'$ and $\mathcal{S}'$ in the concurrent $\mathcal{F}$-hybrid model. This follows since $\mathcal{Z}'$ ignores all inputs that $\mathcal{Z}$ wishes to send to parties $\mathcal{P}_i$ that are not part of a set H that has been previously recorded.

We conclude that the only difference between the execution of $\mathcal{Z}, \mathcal{S}$ in the $\mathsf{s}\mathcal{F}$-hybrid model and the execution of $\mathcal{Z}', \mathcal{S}'$ in the concurrent $\mathcal{F}$-hybrid model is that $\mathsf{s}\mathcal{F}$ ignores calls when the second event occurs. However, we show that the first event only happens with negligible probability. Recall that by construction of $\mathcal{A}'$, $\mathcal{A}'$ *never* outputs "mal-formed" messages of the form $(\texttt{Internal-Init}, sid, P, H, sid_H)$ to $\mathcal{Z}'$ (since $\mathcal{A}'$ only stores a pair (sid_H, H) if $\mathcal{F}_{\mathrm{SA}}$ would have done so, which by the definition of $\mathcal{F}_{\mathrm{SA}}$ implies that only "well-formed" messages $(\texttt{Internal-Init}, sid, P, H, sid_H)$ will be output to $\mathcal{Z}'$). Therefore, by the validity of $\mathcal{S}'$ (2), the probability that $\mathcal{S}'$ outputs a mal-formed message $(\texttt{Internal-Init}, sid, P, H, sid_H)$ in an execution with $\mathcal{Z}'$ is negligible. By the construction of S, it follows that the probability that S (when interacting with $\mathcal{Z}$) sends a malformed message $(\texttt{Init}, sid, P, H, sid_H)$ to $\mathsf{s}\mathcal{F}$ is negligible (recall that $\mathcal{S}$ only sends the message $(\texttt{Init}, sid, P, H, sid_H)$ if $\mathcal{S}'$ sends $(\texttt{Internal-Init}, sid, P, H, sid_H)$ to $\mathcal{Z}'$).

We conclude that the probability that the second event happens in an execution of $\mathcal{Z}, \mathcal{S}, \mathsf{s}\mathcal{F}$ is negligible. Since the execution of $\mathcal{S}', \mathcal{Z}'$ with multiple $\mathcal{F}$ is otherwise identical to the execution of $\mathcal{S}, \mathcal{Z}, \mathsf{s}\mathcal{F}$, we conclude that

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'} \stackrel{\mathrm{s}}{\equiv} \mathrm{IDEAL}_{\mathsf{s}\mathcal{F},\mathcal{S},\mathcal{Z}}. \tag{3}$$

By combining (1), (2), and (3), we have shown that

$$\mathrm{EXEC}^{\mathcal{F}_{\mathrm{SA}}}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}} = \mathrm{EXEC}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}'} \stackrel{\mathrm{c}}{\equiv} \mathrm{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'} \stackrel{\mathrm{s}}{\equiv} \mathrm{IDEAL}_{\mathsf{s}\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

This completes the proof of Lemma 4.1. □

4.5. *Putting It All Together*

Recall that by Theorem 11 there exists a protocol ρ that realizes $\mathcal{F}_{\mathrm{SA}}$ with UC security. The protocol ρ requires the existence of signature schemes that are secure against a chosen message attack, which can be based on the existence of one-way functions. Using the UC theorem, we obtain that protocol Π_F^{ρ} (i.e., the protocol obtained from Π_F by replacing each instance of $\mathcal{F}_{\mathrm{SA}}$ with an instance of ρ) emulates protocol Π_F with gUC security. Since emulation with gUC security implies emulation with UC and with bounded-UC security, we obtain:

Lemma 4.2. *Let $\mathcal{F}$ be ideal functionality that can be realized with bounded-UC (rep., UC, gUC) security, given authenticated communication. Then, assuming the existence of one-way functions, there exists a protocol that securely realizes the split functionality $\mathsf{s}\mathcal{F}$ with bounded-UC (rep., UC, gUC) security.*

Finally, Theorem 1 is obtained by combining Theorem 12 and Lemma 4.2. Theorem 2 is obtained by combining Theorems 13, 14, and Lemma 4.2.

5. Partially Authenticated Networks

A natural and realistic question is what security guarantees can be obtained in *partially authenticated* networks. In such a network some parties share an authenticated link between them and some do not. In fact, some parties may have a link that is authenticated only in one direction. For example, if a server's public key is published, then the link from this server to any client is authenticated, while the link from the client to the server is not authenticated. Another situation where this may arise is when all links are authenticated using passwords taken from a domain which is not too large. If the number of links is large relative to the size of the password domain, then, even if one uses state-of-the-art password-based key exchange protocols, the adversary will still be able to compromise some of the links. Moreover, the uncorrupted parties will not be able to know which links were compromised and which were not. Note also that in the realistic situation where passwords used to authenticate different links are correlated, so will be the distribution of compromised links.

G-Authenticated Networks. All these considerations lead us to define partially authenticated networks in a more general way. Specifically, we consider the notion of an *authentication graph G* which is an n-vertex directed graph (where n is the number of parties). An edge (i, j) is in the graph if the communication from P_i to P_j is authenticated. That is, we place $(i, j) \in G$ if both P_i and P_j are uncorrupted and if we are guaranteed that whenever P_j accepts a message m from P_i, then P_i indeed sent this message at some earlier point in time. (As usual, we let the adversary delete and eavesdrop to messages from P_j to P_i; we only assume that she cannot forge or duplicate such messages.) We call such a network *G-authenticated*. In particular the model of unauthenticated networks we considered in the previous section corresponds to an $\emptyset$-authenticated network (where $\emptyset$ denotes the empty graph), and the standard authenticated-channels model of previous works corresponds to a K_n-authenticated network (where K_n is the complete n-vertex graph). A bit more formally, a G-authenticated network is captured via a relaxed version of the multimessage authentication functionality, $\mathcal{F}_{\text{MAUTH}}$. Specifically, given a graph G, the G-authentication functionality, $\mathcal{F}^G_{\text{MAUTH}}$ (see Fig. 5) is identical to $\mathcal{F}_{\text{MAUTH}}$ except that it guarantees security only for messages from party P_i to party P_j where $(i, j) \in G$.

Our formalism fixes the authentication graph ahead of time, rather than letting it be determined dynamically as the computation proceeds. However, this is not a limitation since we will see that the parties running the protocol need not be aware of the particular current authentication graph. That is, our protocols are oblivious of the authentication graph and give a security guarantee that becomes stronger as the graph becomes more

Functionality $\mathcal{F}_{\text{MAUTH}}^{G}$

1. Upon activation with input $(\texttt{Init}, sid)$, where $sid = (\mathcal{P}, sid')$ and $\mathcal{P}$ is a set of PIDs, record $\mathcal{P}$ and forward it to the adversary.
2. Upon receiving $(\texttt{send}, sid, P_i, P_j, m)$ from P_i, where $P_i, P_j \in \mathcal{P}$, send (P_i, P_j, m) to the adversary and add (P_i, P_j, m) to an (initially empty) list $\mathcal{W}$ of waiting messages. Note that the same triple (P_i, P_j, m) can appear multiple times in the list.
3. Upon receiving $(\texttt{deliver}, sid, P_i, P_j, m)$ from the adversary, if P_i is in $\mathcal{P}$ and is currently corrupted, or $(i, j) \in G$, then output $(\texttt{received}, sid, P_i, P_j, m)$ to P_j. Else, if there is a triple $(P_i, P_j, m) \in \mathcal{W}$, then remove one appearance of it from $\mathcal{W}$ and output $(\texttt{received}, sid, P_i, P_j, m)$ to P_j. Otherwise do nothing.

Fig. 5. The multimessage authentication functionality with respect to authentication graph G.

dense. In particular, if the authentication graph is the complete graph, then our protocol realizes the original functionality rather than its split version. (Technically speaking, $\mathcal{F}_{\text{MAUTH}}^{G}$ may be created by the adversary or environment, who set it up with the appropriate graph G.)

G-Valid Sets and the Functionality $\mathsf{s}_G\mathcal{F}$. We say that a subset $H \subseteq [n]$ is *G-valid* if H does not have an incoming edge from its complement $[n] \setminus H$. That is, for every $(i, j) \in G$, if $j \in H$, then $i \in H$. Note that in a G-authenticated network, members of a G-valid set H cannot verify authenticity of messages coming from outside the set H, and hence an adversary can always split the uncorrupted parties G-valid sets $H_1, \ldots, H_k$ and run the computation in each set independently, as long as all the sets are G-valid. More formally, our notion of security will be this: we define the functionality $\mathsf{s}_G\mathcal{F}$ to act exactly as $\mathsf{s}\mathcal{F}$ except that it only accepts *G-valid* sets H from the adversary. That is, we only change Step 2 in the Initialization part of $\mathcal{F}_{\text{SA}}$ (see Fig. 1) to accept a set H only if H is consistent with G. (Recall that for the purpose of determining validity, we remove the corrupted parties from the sets $H_1, \ldots, H_k$.)

We note that $\mathsf{s}_G\mathcal{F}$ presents the same interface as $\mathsf{s}\mathcal{F}$ to the uncorrupted parties but presents a reduced interface to the adversary (prohibiting her from sending sets that are not G valid). Thus, any protocol that securely realizes $\mathsf{s}_G\mathcal{F}$ also securely realizes $\mathsf{s}\mathcal{F}$ (but possibly not vice versa).

Our Results. We show that if the real network is G-authenticated, then our protocols not only securely realize the functionality $\mathsf{s}\mathcal{F}$ but in fact also the functionality $\mathsf{s}_G\mathcal{F}$. Furthermore, the protocols are oblivious of G, and only their analysis depends on G. That is, we prove the following strengthened versions of Theorems 9 and 10, respectively:

Theorem 16 (Partially Authenticated Computation). *For any n-party well-formed PPT functionality $\mathcal{F}$ and for every n-vertex graph G, the protocol from Section 4 with access to $\mathcal{F}_{\text{MAUTH}}^{G}$ realizes the functionality $\mathsf{s}_G\mathcal{F}$ with bounded-UC (resp., UC, gUC) security, in the presence of static (resp., adaptive) adversaries, under the same computational assumptions as in Theorems 1 and 2.*

The key observation to proving Theorem 16 is that in a G-authenticated network (namely, with access to $\mathcal{F}_{\text{MAUTH}}^{G}$), Protocol 1 actually UC-securely implements not

only $\mathcal{F}_{\text{SA}}$ but in fact also $\mathcal{F}_{\text{SA},G}$, where $\mathcal{F}_{\text{SA},G}$ is again defined as a variant of $\mathcal{F}_{\text{SA}}$ which only accepts G-valid sets (i.e., $\mathcal{F}_{\text{SA},G}$ is equal to $\mathsf{s}_G\mathcal{F}_{\text{MAUTH}}$). That is, we prove the following theorem:

Theorem 17. *Assume that the signature scheme used in Protocol 1 (see p. 743) is existentially secure against chosen-message attacks. Then, Protocol 1 with access to $\mathcal{F}_{\text{MAUTH}}^{G}$ realizes $\mathcal{F}_{\text{SA},G}$ with UC security in the presence of malicious, adaptive adversaries, without any additional setup.*

Proof. Loosely speaking, the reason is the following: suppose that the edge (i, j) is in the authentication graph G. Then in Protocol 1 party i will receive the *correct* public key VK_j of party j. Thus, if party i finishes the protocol, then this means that it received the signature of Party j on sid_i, which means that they are in the same authentication set.

Formally, it suffices to prove is that, except with negligible probability, the simulator $\mathcal{S}$ constructed in the proof of Theorem 11 never queries the functionality with a set H that is not G-valid. (This is so since in this case $\mathcal{F}_{\text{SA}}$ and $\mathcal{F}_{\text{SA},G}$ behave in exactly the same way.)

Indeed, let $H = H_i$ be a set sent by the simulator $\mathcal{S}$ to the ideal functionality in Step 3 of its operation. Note that H_i is chosen by some uncorrupted party P_i which completed the link initialization phase (Phase I) of the protocol, and it is defined to be the set of all j such that the authentic verification key sent by party P_j is in sid_i.

Now suppose that there is an edge (i', j) in G such that $j \in H$. We need to prove that $i' \in H$. The fact that $(i', j) \in G$ implies that both $P_{i'}$ and P_j are uncorrupted and that link is authenticated. Hence sid_j contains the valid verification key of $P_{i'}$. Now because sid_i contains the authentic verification key of P_j, and because P_i completed the link initialization phases successfully, the security of the signature scheme implies that with very high probability the value sid_{i_j} that P_i receives is equal to sid_j. Hence, it contains the authentic verification key of $P_{i'}$. Since the link initialization phase is completed by P_i only if for all j, $sid_{i_j} = sid_i$, in particular it holds that sid_i contains the authentic verification key of $P_{i'}$. However, this implies that $i' \in H$. (A more formal reduction to the security of the underlying signature scheme follows the lines of the reduction in the case of $\mathcal{F}_{\text{SA}}$.) □

Proving Theorem 16 from Theorem 17. Theorem 17 implies Theorem 16 in an analogous way to the way that, in the completely unauthenticated case, Theorem 11 implies Theorems 9 and 10. As there, the main part is the following extension of Lemma 4.1:

Lemma 5.1. *Let $\mathcal{F}$ be an n-party ideal functionality, and let π_F be a protocol that securely n-realizes $\mathcal{F}$ with bounded-UC (resp. UC, gUC) security, using $\mathcal{F}_{\text{MAUTH}}^{G}$ for graph G. Then, protocol Π_F in Fig. 4 using only an instance of $\mathcal{F}_{\text{SA},G}$, securely realizes the split functionality $\mathsf{s}_G\mathcal{F}$ with bounded-UC (resp. UC, gUC) security.*

Proof. The proof closely follows the proof of Lemma 4.1. That is, we convert an adversary $\mathcal{A}$ and environment $\mathcal{Z}$ that invoke a single execution of Π_F in the $\mathcal{F}_{\text{SA},G}$-hybrid model to an adversary $\mathcal{A}'$ and environment $\mathcal{Z}'$ that invoke up to n concurrent executions

of π in the fully authenticated communication model. We then use the simulator $\mathcal{S}'$ for $\mathcal{A}'$, $\mathcal{Z}'$ to construct a simulator $\mathcal{S}$ for $\mathcal{A}$ in the $\mathcal{F}_{\mathrm{SA},G}$-hybrid model. The conversion is almost the same as in the proof of Lemma 4.1, except that now $\mathcal{A}'$ emulates the uncorrupted action of $\mathcal{F}_{\mathrm{SA},G}$ rather than $\mathcal{F}_{\mathrm{SA}}$. The only thing we need to worry about is to ensure that $\mathcal{S}$ asks $\mathcal{F}_{\mathrm{SA},G}$ a query with a set H that is not G-valid only with negligible probability. However, as there, this follows since the adversary $\mathcal{A}'$ by construction never asks such queries, and the simulator cannot deviate from this behavior except with negligible probability. □

Plugging Theorem 17 into Lemma 5.1, we obtain Theorem 16. Note that all our protocols are oblivious of the underlying authentication network G.

Implications. It suffices to have a G-authenticated network for *some* connected graph G in order to carry out multiparty computation. Specifically, if there is one uncorrupted party with a digital certificate for a public-key infrastructure that can be used for generating a secure channel, then it is possible to achieve secure computation for all parties. Importantly, it is not necessary to know which party is uncorrupted. Thus, if there are a few such parties, all we need to assume is that at least one is uncorrupted. This can be used in real-world settings to obtain secure multiparty computation (e.g., auctions) where pairwise authenticated channels do not (and are unlikely to) exist.

Acknowledgements

We thank Shai Halevi, Johan Håstad, Jonathan Katz, and Alon Rosen for conversations on the topic and Victor Shoup and the anonymous referees for very helpful remarks on earlier drafts.

References

[1] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for noncryptographic fault-tolerant distributed computations, in *20th STOC*, pp. 1–10, 1988

[2] J. Camenisch, N. Casati, T. Gross, V. Shoup, Credential authenticated identification and key exchange, in *Crypto'10*, 2010

[3] R. Canetti, Security and composition of multiparty cryptographic protocols. *J. Cryptol.* **13**(1), 143–202 (2000)

[4] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, in *42nd FOCS*, pp. 136–145, 2001. The full version is available for download from http://eprint.iacr.org/2000/067

[5] R. Canetti, Y. Dodis, R. Pass, S. Walfish, Universally composable security with pre-existing setup, in *The Fourth Theory of Cryptology Conference (TCC)*. LNCS, vol. 4392 (Springer, Berlin, 2007), pp. 61–85

[6] R. Canetti, M. Fischlin, Universally composable commitments, in *CRYPTO 2001*. LNCS, vol. 2139 (Springer, Berlin, 2001), pp. 19–40

[7] R. Canetti, S. Halevi, A. Herzberg, Maintaining authenticated communication. *J. Cryptol.* **13**(1), 61–105 (2000)

[8] R. Canetti, S. Halevi, J. Katz, Y. Lindell, P. MacKenzie, Universally composable password-based key exchange, in *EUROCRYPT 2005*. LNCS, vol. 3494 (Springer, Berlin, 2005), pp. 404–421

[9] R. Canetti, E. Kushilevitz, Y. Lindell, On the limitations of universally composable two-party computation without set-up assumptions, in *EUROCRYPT '03*. LNCS, vol. 2656 (Springer, Berlin, 2003), pp. 68–86
[10] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai, Universally composable two-party and multi-party secure computation, in *34th STOC*, pp. 494–503, 2002
[11] R. Canetti, T. Rabin, Universal composition with joint state, in *CRYPTO 2003*. LNCS, vol. 2729 (Springer, Berlin, 2003), pp. 265–281
[12] D. Chaum, C. Crepeau, I. Damgard, Multiparty unconditionally secure protocols, in *20th STOC*, pp. 11–19, 1988
[13] D. Dolev, C. Dwork, M. Naor, Non-malleable cryptography. *SIAM J. Comput.* **30**(2), 391–437 (2000)
[14] C. Dwork, M. Naor, Pricing via processing or combating junk mail, in *CRYPTO'92*. LNCS, vol. 740 (Springer, Berlin, 1992), pp. 139–147
[15] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, A. Smith, Detectable byzantine agreement secure against faulty majorities, in *21st PODC*, pp. 118–126, 2002
[16] O. Goldreich, *Foundations of Cryptography*, vol. 2 (Cambridge University Press, Cambridge, 2004)
[17] O. Goldreich, Y. Lindell, Session-key generation using human passwords only, in *CRYPTO 2001*. LNCS, vol. 2139 (Springer, Berlin, 2001), pp. 408–432
[18] S. Goldwasser, S. Micali, R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2), 281–308 (1988)
[19] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
[20] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game, in *19th STOC*, pp. 218–229, 1987
[21] S. Halevi, H. Krawczyk, Public-key cryptography and password protocols. *ACM Trans. Inf. Syst. Secur.* **2**(3), 230–268 (1999)
[22] Y. Lindell, Bounded-concurrent secure two-party computation without setup assumptions, in *35th STOC*, pp. 683–692, 2003
[23] Y. Lindell, Lower bounds for concurrent self composition, in *1st TCC*. LNCS, vol. 2951 (Springer, Berlin, 2004), pp. 203–222
[24] M. Nguyen, S. Vadhan, Simpler session-key generation from short random passwords, in *1st TCC*. LNCS, vol. 2951 (Springer, Berlin, 2004), pp. 428–445
[25] R. Pass, Bounded-concurrent secure multi-party computation with a dishonest majority, in *36th STOC* (Springer, Berlin, 2004), pp. 232–241
[26] R. Pass, A. Rosen, Bounded-concurrent secure two-party computation in a constant number of rounds, in *44th FOCS*, pp. 404–413, 2003
[27] V. Shoup, On formal models for secure key exchange. IBM Research Report RZ 3120 (April 1999). Revised version appears at www.shoup.net
[28] A.C. Yao, How to generate and exchange secrets, in *27th FOCS*, pp. 162–167, 1986