Secure Computation Without Authentication

Boaz Barak[1], Ran Canetti[2], Yehuda Lindell[3], Rafael Pass[4*], and Tal Rabin[2]

[1] IAS. E:mail: boaz@ias.edu
[2] IBM Research. E-mail: {canetti,talr}@watson.ibm.com
[3] Bar-Ilan University, Israel E-mail: lindell@cs.biu.ac.il
[4] MIT. E-mail: pass@csail.mit.edu

Abstract. In the setting of secure multiparty computation, a set of parties wish to jointly compute some function of their inputs. Such a computation must preserve certain security properties, like privacy and correctness, even if some of the participating parties or an external adversary collude to attack the honest parties. Until this paper, *all* protocols for general secure computation assumed that the parties can communicate reliably via authenticated channels. In this paper, we consider the feasibility of secure computation without *any* setup assumption.
We consider a completely unauthenticated setting, where all messages sent by the parties may be tampered with and modified by the adversary (without the honest 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. In particular, we define a relaxed notion of what it means to "securely compute" a function in the unauthenticated setting. Then, we construct protocols for securely realizing any functionality in the stand-alone model, *with no setup assumptions whatsoever*. In addition, we construct universally composable protocols for securely realizing any functionality in the common reference string model (while still in an unauthenticated network). We also show that 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 *non-malleable commitments*.

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 honest parties' security.

Since the introduction of this problem in the 1980's [25, 17, 2, 10], the research in this area has not subsided. The area has produced hundreds of papers that deal

* Research supported by an Akamai Presidential Fellowship.

with many aspects of the problem. Works have included the definitional issues of secure computation, protocols with low round and communication complexity, protocols that rely on a wide variety of computational assumptions, lower bounds, security under composition and much much more.

Interestingly, one assumption that has appeared in *all* of the works in the field of secure computation until now is that *authenticated channels exist between the parties.* That is, it has always been assumed that the participating parties can communicate reliably with each other, without adversarial interference. In particular, the adversary is unable to send messages in the name of honest parties, or modify messages that the honest parties send to each other. There seem to be two main reasons that this assumption was always considered. First, the common belief was that these channels can easily be achieved, either through a physical designated channel connecting every pair of parties, or more realistically, via the deployment of a public-key infrastructure that can be used for implementing secure digital signatures. Second, it was assumed that no meaningful security guarantees can be provided in a distributed setting, unless honest parties can reliably communicate with each other.

Despite the above common belief, in real life the assumption that authenticated channels can be easily achieved is actually very problematic. It is clear that physical channels are generally unrealistic. In addition, a *fully deployed* public-key infrastructure is also far from reach. That is, although we can typically expect that most servers have an appropriate certificate for digital signatures, it is unreasonable today to require every participant (client) to also have one. This observation leads us to the following natural question:

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

As we have seen, this question has important ramifications regarding the usefulness of secure multiparty protocols in real-world settings. However, it is also of great *theoretical* interest. In general, the theory of cryptography aims at understanding what tasks can be securely solved and under what (complexity and other) assumptions. Considered in this light, it is most natural to examine what security can be achieved in a setting with no setup assumptions whatsoever. In addition to highlighting the borders of feasibility and infeasibility for secure computation, answering this question enhances our understanding of the role of authentication in secure computation (detailed discussion follows).

Security without authenticated channels. For simplicity, we begin by considering the important 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 honest parties completely, and engage in separate executions with each of the two parties. Such an attack is unavoidable since there is no authentication between the parties. Therefore, the parties have no way of distinguishing the case that they interact with each other from the case that they each interact separately with a third party (in this case, the adversary). Given that this is an inherent limitation, 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.
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 with A while playing B's role, and an execution with B while playing A's role. The security guarantee here is that the adversary is unable to make one execution depend on the other. Rather, the adversary must essentially choose an input for each execution and then run each execution as if it was running by itself. We remark that such "full" independence is actually impossible to achieve because the adversary can always run a complete execution with one of the parties, and then subsequently use the output it already received in order to choose its input for the execution with the other party. Therefore, our security definition guarantees that the *only* dependence the adversary can achieve is due to running the executions sequentially and choosing its input in the second execution after receiving its output from the first.

When considering the two-party case, the above security notion is a direct extension of the notion of *non-malleability*, introduced by Dolev, Dwork and Naor [11]. The work of [11] considered the specific tasks of encryption, commitments and zero-knowledge proofs. Here, we generalize these ideas to the more general concept of two-party (and multiparty) computation.

The same line of reasoning can be applied to analyze what is possible also in the case of multi-party protocols. Specifically, an adversary can always partition the honest 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 of honest parties H, 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. In particular, we consider an adversary who interacts with a set of parties who are each willing to run a single execution (with each other). Our definition then states that although the adversary can actually run many executions with subsets of parties, it is guaranteed that:

1. The subsets of honest parties are disjoint,
2. Once a subset of honest parties is chosen, it is fixed for the duration of the protocol, and
3. The only dependence between the executions is due to the capability of the adversary to run the executions sequentially and choose its inputs as a function of the outputs from executions that have already terminated.

We remark that within each subset of parties, the execution that takes place is actually the same as when there are authenticated channels.

1.1 The Main Result

Our main result is a general proof of feasibility for stand-alone computation in the unauthenticated network setting. That is, we show that it is possible to securely compute any functionality according to the above security guarantees, even in a network with *no setup assumptions whatsoever*. This is in contrast to the widely held belief that authenticated channels, or some other setup, is a necessity for obtaining meaningful security. As an unusual step, before discussing the definition in more detail, we will first present the high-level idea behind our protocol. We feel that presenting the results in this order actually makes them easier to understand, especially because our protocol is in fact very simple.

It is clear that in order to run any of the known protocols for secure computation, authenticated channels are required. Our protocol for the unauthenticated setting is therefore comprised of two stages. In the first stage, some authenticated channels are set up. Then, in the second stage, a secure protocol is run on top of these authenticated channels. The basic idea of the protocol is:

Stage 1 – link initialization: In this stage, each party P_i generates a pair of signing and verification keys (s_i, v_i) and sends the signature verification key v_i to all other parties. In addition, after receiving verification keys v_j from all other parties, P_i signs on the series of all keys received (with its secret signing key s_i) and sends the signature σ_i to all other parties. Finally, each P_i checks that the signature it generated and all the signatures that it received refer to the same set of verification keys.

The idea behind this step is as follows. Let P_i and P_j be honest 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_i will actually receive v_i and not some $v'_i \neq v_i$ generated by the adversary (and likewise for P_j). However, *if P_i and P_j do receive each other's real keys*, then they can set up a secure channel between them. In particular P_i has a verification key v_j associated with a secret signing-key known only to P_j, and vice versa. Thus, digital signatures can be used in a standard way in order to achieve authenticated communication between P_i and P_j. We note that if P_i received v_j (i.e., the key sent by P_j), then it will only continue if P_j received the exact same set of keys as P_i. This is guaranteed by the fact that the parties also sign on all the keys that they received. Thus, if P_j received different keys than P_i, then its signature σ_j will not include the same keys P_i received. When P_i receives the signature from P_j, it will therefore detect that adversarial interference has taken place, and so will abort. (Note that by our assumption here P_i already received v_j as generated by P_j, and so the adversary cannot hand P_i any other signature without breaking the signature scheme.)

From the above, we have that at the end of this stage, if P_i and P_j received each other's keys, then they have a secure bidirectional channel between them *and* they received the same set of verification keys. In contrast, if this is not the case, then we are guaranteed that their views of the verification keys are different. As we will show, this actually defines a *partition* of the honest parties so that **(1)** within each partition all of the honest parties hold each others' verification keys (and so all have mutually authenticated channels), and **(2)** the honest parties in different partitions have different views of the verification keys.

Stage 2 – secure computation using the generated links: In this stage, the parties run a protocol on top of the authenticated channels generated in the link initialization phase. The basic idea is to force the executions of the different subsets of honest parties, as defined by the above partition, to be *independent*. In order to do this, we view the series of verification keys as a *session identifier*. Then, we run a protocol for the authenticated channels model that is secure under concurrent composition, and guarantees independence between executions with different session identifiers. (We need security under concurrent composition, because different executions with different subsets may be run concurrently.)

As can be seen from the above protocol, and as we have discussed above, the only power provided to the adversary here is the ability to partition the honest parties into disjoint subsets and run separate executions with each set. (This adversarial "attack" can be carried out on *any* protocol in the unauthenticated model, and is not due to a weakness of our protocol.) We therefore model security by allowing the adversary to carry out such an "attack" in the ideal model as well. However, rather than modifying the standard ideal model, our basic definition of security is actually the same as in the standard model with authenticated channels and no honest majority. Then, the additional power awarded to the adversary here is modeled by modifying the definition of the functionality that is to be computed. That is, for any functionality $\mathcal{F}$ to be realized by two or more parties, we define a relaxed version of $\mathcal{F}$ called $\mathsf{split}\text{-}\mathcal{F}$, or $\mathsf{s}\mathcal{F}$, which is an *interactive functionality* and works as follows. Functionality $\mathsf{s}\mathcal{F}$ lets the adversary define disjoint sets of parties, called authentication sets. Then, a separate and independent instance of the original functionality $\mathcal{F}$ is invoked for each authentication set. In an ideal execution of $\mathcal{F}$ for a given authentication set, functionality $\mathsf{s}\mathcal{F}$ also allows the adversary to play the roles of all the honest parties not in the set (i.e., providing their inputs and receiving their outputs). In the two-party case, the adversary can either choose a *single* authentication set containing both parties (and then it cannot do anything more than in the authenticated channels model), or it can choose two authentication sets, each containing a single party (and so it must run an independent and separate execution with each party).

Theorem 1 (unauthenticated stand-alone computation): *Assume the existence of collision-resistant hash functions and enhanced trapdoor permutations,*[5] *and consider the stand-alone model with no setup whatsoever. Then, for any probabilistic polynomial-time multiparty functionality $\mathcal{F}$ there exists a protocol that securely computes the split functionality $\mathsf{s}\mathcal{F}$, in the presence of static, malicious adversaries.*

[5] See [19, Appendix C.1] for the definition of *enhanced* trapdoor permutations.

Theorem 1 holds irrespective of the number of corrupted parties. In particular, this means that no honest majority is assumed (and therefore fairness and output delivery are not guaranteed, as is standard for this setting). We stress that unlike the setting of authenticated channels, here it would not help even if we did assume that a large fraction of the parties are honest. This is due to the fact that the adversary can always choose all the subsets to be small, thereby ensuring an honest minority in each execution (and making it impossible to prevent an adversarial early abort).

We stress that although Theorem 1 constitutes a "general feasibility result", 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, and indeed the split-functionality formalization explicitly removes all flavor of agreement (note that honest parties in different subsets run independent executions and so clearly cannot agree on anything).

Concurrency in a stand-alone world. From the above informal description of our protocol we see that the *stand-alone setting* with unauthenticated channels implicitly enables the adversary to run *concurrent executions* with different sets of honest parties. Thus, the protocol that is used in the second stage must be secure under concurrent composition. However, an important observation here is that when there are n honest parties, the adversary can force at most n concurrent executions (because the sets are disjoint and each honest party runs only once). It therefore follows that we only need security under *bounded* concurrency, which is fortunately much easier to achieve. (See [21] for impossibility results for the setting of unbounded concurrency, in contrast to the feasibility results of [20, 23] and specifically for our use [24] when the concurrency is bounded.)

Entity authentication versus session authentication. One interesting corollary of our results is a more explicit distinction between entity authentication and session authentication. Entity authentication relates to a situation where a party A can verify that messages that it received in the name of party B were indeed sent by B. In contrast, session authentication relates to the fact that a party A establishes an authenticated channel with some other fixed party within a protocol execution or 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 honest, then the adversary cannot interfere with any messages that are sent on the channel. This distinction is not new, and appears already in [11]. However, our results make it more explicit. Indeed, in the first stage of our protocol, we carry out a "session authentication protocol". Then, in the second stage, secure computation is carried out on top of this. By including entity authentication into the secure protocol of the second stage (or equivalently into the functionality being computed), we obtain an explicit separation of session authentication from entity authentication. This separation enables the entity authentication to be carried out *on top* of the 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 and so on.

1.2 Additional Results and Applications

The above result is of interest due to the fact that it requires no setup assumption whatsoever. However, it only holds for the rather limited stand-alone model. In this section, we briefly discuss some extensions and applications of this result. Formal statements and proofs of these results will appear in the full version of this paper.

UC protocols without authenticated channels. Universal composability (UC) is a definition of security with the property that any protocol that is UC-secure is guaranteed to remain secure under concurrent general composition [4] (i.e., when it is run many times concurrently with arbitrary other protocols). As in the stand-alone model, all UC-secure protocols until today assumed the existence of authenticated channels. We therefore extend our results to this setting.

We first note that in this setting, there is no hope of succeeding without setup assumptions. This is due to the fact that broad impossibility results for obtaining UC security have been demonstrated, even when there are authenticated channels [5, 4, 7]. We therefore consider the feasibility of obtaining UC-security without authenticated channels, but in the common reference string (CRS) model, where it is assumed that all parties have access to a single string that was chosen according to some predetermined distribution. In the CRS model and assuming authenticated channels, it has been shown that UC-secure protocols exist for essentially every functionality [8]. We combine our "link-initialization" protocol, as described above, with the protocol of [8] in order to achieve UC-secure protocols that compute essentially any split functionality $\mathsf{s}\mathcal{F}$ in the CRS model with unauthenticated channels.

This combination of setup assumptions may seem strange. However, first note that at the very least, our result reduces the setup assumptions required for obtaining UC-security. More importantly, we argue that the assumption regarding a CRS is incomparable to that of authenticated channels. On the one hand, the generation of a CRS requires global trust of a stronger nature than that required for authenticated channels. On the other hand, it requires that only one string is generated and posted on some "secure bulletin board". In contrast, setting up authenticated channels essentially requires that all parties obtain a certificate for digital signatures. We also note that a common reference string by itself does not provide the means for parties to authenticate themselves to each other. Indeed, it is impossible to construct authenticated channels from unauthenticated ones even in the CRS model.

Partially authenticated networks. So far, we have discussed the completely unauthenticated setting, and have contrasted it to the standard completely authenticated setting. However, the most realistic setting is actually that of a *partially authenticated network*, where some of the parties have authenticated links and others do not. In addition, the authentication on these links may be unidirectional or bidirectional. For example, consider the case that only some of the parties have certificates for public (signature) keys as part of an implemented public-key infrastructure. Current protocols guarantee nothing in this setting.

However, this is the real setting of the Internet today. We should be able to use a secure auction protocol, even if the only party who has a certificate is the auctioneer (in this case, all parties can obtain authenticated communication from the auctioneer, but that is all). In the full version of this paper, we show how to use our results in order to obtain secure computation in a partially authenticated network, while utilizing the authenticated links that do exist.

Password-based authenticated key-exchange. One problem that has received much attention, and is cast in the setting without authenticated channels, is that of password-based key exchange. Our results can also be applied to this problem. First, note that our definitional framework provides a way of modeling the problem easily within the setting of secure computation. Specifically, we define a functionality $\mathcal{F}$ as follows. Each party provides an input; if the inputs are equal, then $\mathcal{F}$ provides each with a long random value; if the inputs are not equal, then $\mathcal{F}$ hands $\perp$ to each party. Of course, the inputs we are referring to here are the parties' secret passwords.

The functionality $\mathcal{F}$, as defined, does not enable the adversary to make on-line password guesses, which is possible in password based key-exchange schemes. However, the transformation of $\mathcal{F}$ to its split functionality $\mathsf{s}\mathcal{F}$ provides this exact capability. Thus, the problem of securely computing $\mathsf{s}\mathcal{F}$ is exactly the problem of obtaining secure password-based authenticated key exchange. In particular, if the adversary plays a message relay strategy, then the parties will succeed in obtaining a shared secret key. In contrast, if the adversary runs independent executions, then the adversary will obtain exactly two password guesses. Furthermore, if the adversary guesses incorrectly, then the parties will obtain $\perp$. We note that this definition is essentially the same as that proposed in [6].

Our result therefore yields a conceptually simple framework and definition for solving this problem. Furthermore, we improve on previous solutions as follows. First, applying our first theorem 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 [14, 22]. Comparing our result to [14, 22] 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 [6] 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). We note that like [14, 22], our password-based protocol for the model with no setup assumptions (beyond the passwords themselves) is only secure if the same password is not used in concurrent executions of the protocol.

In addition to the above, we can apply our UC-secure protocol and obtain UC-secure password-based authenticated key-exchange in the common reference string model. This problem was previously considered by [6], who present highly efficient protocols based on specific assumptions. In contrast, we obtain protocols less efficient protocols, based on general assumptions. In addition, we can also extend our result to the setting of adaptive adversaries.

Alternative authentication mechanisms. Passwords are just one mechanism for authenticating parties. Due to the generality of our result which demonstrates that any function can be securely computed, we can obtain secure protocols for other, non-standard 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 non-trivial computational task, like the "proof of work" in the anti-spam work of [12]. Finally, our protocols can be used to obtain "anonymous authentication" 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.

Non-malleable commitments. We remark that non-malleable commitments [11] can be obtained using our results in a similarly simple manner. Namely, define $\mathcal{F}$ to be a non-interactive (and potentially malleable) commitment function. Then, a protocol that securely computes $\mathsf{s}\mathcal{F}$ constitutes a non-malleable commitment. This protocol does not improve on other known results. Nevertheless, it demonstrates the power of our general framework.

2 Split Functionalities

Due to the lack of space in this abstract, we will not present the definitions of secure computation. We refer the reader to [19, Chapter 7] for motivation and definitions. We note that we consider *reactive functionalities* here; see [4, Full version] for a formal discussion of this notion. Informally, the setting of secure computation with reactive functionalities is very similar to that of the more familiar "secure function evaluation". In the setting of secure function evaluation, an ideal model is defined where all parties send there inputs to a trusted party who computes the output and sends it back. When considering reactive functionalities, the only difference is that inputs and outputs can be provided interactively and at different stages. Thus, the trusted party interacts with the honest parties *and the adversary* multiple times, as specified by the code of the functionality.

In this section, we define what it means to realize an ideal functionality in an unauthenticated network without any setup. Before doing so, we remark that an unauthenticated network is formally modeled by having all communication go via the adversary. Thus, when a party P_i wishes to send a message m to P_j, it essentially just hands the tuple (P_i, P_j, m) to the adversary. It is then up to the adversary to deliver whatever message it wishes to P_j. We also remark that in the ideal model that we consider here, the communication between the honest parties and the trusted party remains ideally private and authenticated. Thus, the only change is to the real model.

We now proceed to the definition. As we have mentioned above, defining security in the unauthenticated model essentially involves defining a class of "split functionalities" that specifies the code of the trusted party in an ideal execution.[6] As we have mentioned, this class of functionalities enables the ideal-model adversary to split the honest 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. However, each such execution has the property that the adversary plays the roles of all the parties not in H (i.e., the parties that complete H to the full set of parties). Our specific formulation below provides three important 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 honest parties in that set);
2. The computation within each set is secure in the standard sense (as in the case that authenticated channels are assumed);
3. The computation in a set 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.

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 s$\mathcal{F}$, in Figure 1. We note that the functionality is slightly more involved than what is needed for the stand-alone case. The additional complications are included so that the same functionality will also be useful for the UC setting.

The split functionality s$\mathcal{F}$ – explanation. In the initialization stage of the functionality, the adversary adaptively chooses subsets of honest parties H (the adaptivity relates to the fact that an authentication set can be chosen and a full execution completed, before the next authentication set is chosen). The adversary can choose any subsets that it wishes under the following constraints: First, the subsets must be *disjoint*. Second, the adversary must choose a *unique* session identifier sid_H for each authentication set H.

In the computation stage of the functionality s$\mathcal{F}$, 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 $P_i \in H$ provide their own inputs, and the adversary provides the inputs for all $P_j \notin H$. 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 $P_i \in H$ all receive their specified outputs as computed by their copy of $\mathcal{F}$. However, the adversary receives all of its own outputs, as well as the outputs of the parties $P_j \notin H$ (as is to be expected, since it plays the role of all of these parties in the execution). We stress that there is no interaction whatsoever between the different copies of $\mathcal{F}$ run by s$\mathcal{F}$.

[6] As we have mentioned, this set of instructions for the trusted party could be incorporated into the definition of the ideal model. Equivalently, we have chosen to leave the ideal-model unchanged, and instead modify the functionality to be realized.

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

For parties $P_1, \ldots, P_n$ and a given $\mathcal{F}$, functionality $\mathsf{s}\mathcal{F}$ proceeds as follows:

Initialization:

1. Upon receiving a message $(\texttt{Init}, sid)$ from a party P_i, send $(\texttt{Init}, P_i)$ to the adversary.
2. Upon receiving a message $(\texttt{Init}, sid, P_i, H, sid_H)$ from the adversary, verify that party P_i previously sent $(\texttt{Init}, sid)$, that the list H of party identities includes P_i, and that for all previously recorded sets H', it holds that either **(1)** H and H' are disjoint 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), send $(\texttt{Init}, sid, sid_H)$ to P_i, and initialize a new instance of the original functionality $\mathcal{F}$ with session identifier sid_H. Let $\mathcal{F}_H$ denote this instance of $\mathcal{F}$.

Computation:

1. Upon receiving a message $(\texttt{Input}, sid, v)$ from party P_i, find the set H such that $P_i \in H$, and forward the copy of the functionality $\mathcal{F}_H$ the message v from P_i. If no such H is found then ignore the message.
2. Upon receiving a message $(\texttt{Input}, sid, H, P_j, v)$ from the adversary, if $\mathcal{F}_H$ is initialized and $P_j \notin H$, then forward v to $\mathcal{F}_H$ as if coming from party P_j. Otherwise, ignore the message.
3. When a copy $\mathcal{F}_H$ generates an output v for party $P_i \in H$, functionality $\mathsf{s}\mathcal{F}$ sends v to P_i. When the output is for a party $P_j \notin H$ or for the adversary, $\mathsf{s}\mathcal{F}$ sends the output to the adversary.

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

The functionality $\mathsf{s}\mathcal{F}$ – remarks:

1. The requirement that the authentication sets are disjoint guarantees that all the parties in an authentication set have consistent views of the interaction. In particular, each party participates in only one execution, and this is consistent with the other parties in its set.
2. $\mathsf{s}\mathcal{F}$ requires the adversary to provide a unique identifier, sid_H for each authentication set. 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. In a sense, this forces the adversary in the ideal process to mimic the same partitioning to authentication sets as in the protocol execution.
3. The above formalization of $\mathsf{s}\mathcal{F}$ assumes for simplicity that the number and identities of the parties is known in advance. However, this requirement is not essential and neither the number of parties nor their identities need to be known in advance. Furthermore, they can be determined adaptively by the adversary as the computation proceeds. In this case, the only difference is that each party needs to receive the set of parties with which it should interact as part of its first input.

3 Obtaining Split Authentication

In this section, we show how to securely implement a link initialization phase. We proceed in two steps. First, we present an ideal functionality $\mathcal{F}_{\mathrm{SA}}$ that captures the property of authentication *within* an authentication set. Next, we present a simple protocol that UC-securely computes the $\mathcal{F}_{\mathrm{SA}}$ functionality *in the bare model,* without any setup. In Section 4 we will use the $\mathcal{F}_{\mathrm{SA}}$ functionality in order to obtain secure protocols for any split functionality $\mathsf{s}\mathcal{F}$.

3.1 The Split Authentication Functionality $\mathcal{F}_{\mathrm{SA}}$

The split authentication functionality $\mathcal{F}_{\mathrm{SA}}$ is essentially a functionality that enables parties in the same authentication set to communicate in a reliable way. In particular, if the adversary wishes to deliver a message m to a party P_j with an alleged sender P_i, then $\mathcal{F}_{\mathrm{SA}}$ proceeds as follows:

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

Formally, $\mathcal{F}_{\mathrm{SA}}$ is the split functionality of the functionality $\mathcal{F}_{\mathrm{AUTH}}$ defined in Figure 2 (we note that $\mathcal{F}_{\mathrm{AUTH}}$ here is a "multiple-session extension" of the $\mathcal{F}_{\mathrm{AUTH}}$ functionality defined in [4]). In other words, we define $\mathcal{F}_{\mathrm{SA}} = \mathsf{s}\mathcal{F}_{\mathrm{AUTH}}$.

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

$\mathcal{F}_{\mathrm{AUTH}}$ interacts with an adversary and parties $P_1, \ldots, P_n$ as follows:

1. Upon receiving $(\mathtt{send}, sid, P_i, P_j, m)$ from P_i, 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 entry can appear multiple times in the list.
2. Upon receiving $(\mathtt{deliver}, sid, P_i, P_j, m)$ from the adversary, if there is a message $(P_i, P_j, m) \in \mathcal{W}$ then remove it from $\mathcal{W}$ and send $(\mathtt{received}, sid, P_i, P_j, m)$ to P_j. Otherwise do nothing.

Fig. 2. The authentication functionality $\mathcal{F}_{\mathrm{AUTH}}$

3.2 Realizing $\mathcal{F}_{\mathrm{SA}}$

In this section, we present a simple protocol for securely computing $\mathcal{F}_{\mathrm{SA}}$ in the bare model without any setup. The protocol that we present is actually UC-secure. This is important for two reasons. First, it is useful for achieving the extension of our results to the UC setting. Second, it enables us to claim that it remains secure even when run concurrently with any other protocol. This will be important in our final protocol (presented in Section 4) where the protocol for computing $\mathcal{F}_{\mathrm{SA}}$ is run together with the protocol of [23].

Our protocol uses a signature scheme that is existentially unforgeable against chosen message attacks as in [15] and is reminiscent of the technique used in

[11] to construct non-malleable encryption. On a high-level our protocol also resembles the Byzantine Agreement protocol of [13] (although the goal and the actual protocol is 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:** *Upon input* $(\texttt{Init}, sid)$*, each party* P_i *proceeds as follows:*

1. P_i *chooses a key pair* (VK_i, SK_i) *for the signature scheme.*
2. P_i *sends* VK_i *to all parties* P_j*. (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.)*
3. P_i *waits until it receives keys from every* P_j*, for* $j \in [n], j \neq i$*. (Recall that these keys are actually received from the adversary and do not necessarily correspond to keys sent by other parties.) Denote by* VK_{i_j} *the key that* P_i *received from* P_j *and denote* $VK_{i_i} = VK_i$*. Now, let* $VK_{i'_1}, \ldots, VK_{i'_n}$ *be the same set of keys* $VK_{i_1}, \ldots, VK_{i_n}$ *arranged in ascending lexicographic order. If there are two keys that are the same, then* P_i *halts. Otherwise,* P_i *defines* $sid_i = \langle VK_{i'_1}, \ldots, VK_{i'_n} \rangle$*.*
4. P_i *computes* $\sigma_i = \mathsf{Sign}_{SK_i}(sid_i)$ *and sends* $\alpha_i = (sid_i, \sigma_i)$ *to all parties* P_j*.*
5. P_i *waits until it receives an* α_j *message from every* P_j*, for* $j \in [n], j \neq i$*. Denote by* $\alpha_{i_j} = (sid_{i_j}, \sigma_{i_j})$ *the pair that* P_i *received from* P_j *and denote* $\alpha_{i_i} = \alpha_i$*. Then,* P_i *checks that for every* j*,* $\mathsf{Verify}_{VK_{i_j}}(sid_{i_j}, \sigma_{i_j}) = 1$ *and that* $sid_{i_1} = sid_{i_2} = \cdots = sid_{i_n}$*. 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 the following theorem:

Theorem 2 *Assume that the signature scheme used in Protocol 1 is existentially secure against chosen message attacks. Then, Protocol 1 securely computes the* $\mathcal{F}_{\mathrm{SA}}$ *functionality under the UC-definition in the presence of malicious, adaptive adversaries, and in the bare model with no setup whatsoever.*

Proof Sketch: We show that for any adversary $\mathcal{A}$ there exists an ideal-process adversary (i.e., a simulator) $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with parties running Protocol 1 and adversary $\mathcal{A}$, or with $\mathcal{F}_{\mathrm{SA}}$ and simulator $\mathcal{S}$. The simulator $\mathcal{S}$ internally invokes $\mathcal{A}$ and perfectly simulates the honest parties interacting with $\mathcal{A}$. Then, when an honest party P_i in the internal simulation by $\mathcal{S}$ completes its link initialization phase and computes 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.

Next, when $\mathcal{A}$ delivers a signed message to some P_i, simulator $\mathcal{S}$ asks $\mathcal{F}_{\rm SA}$ to deliver the message to P_i in the ideal process only if the internally simulated honest party would accept the signature, according to the protocol specification. More specifically, $\mathcal{S}$ locally runs an interaction between $\mathcal{A}$ and simulated copies of all the parties. In addition:

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 $\mathcal{S}$ receives a message that an honest party P_i sent an $(\texttt{Init}, sid)$ message to $\mathcal{F}_{\rm SA}$, simulator $\mathcal{S}$ simulates the actions of an honest P_i in the link initialization phase of Protocol 1.
3. Whenever an internally simulated party P_i completes the link initialization phase with sid_i, simulator $\mathcal{S}$ determines the set H_i to be the set of honest 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 honest parties, so it can do this.) $\mathcal{S}$ then checks that for all previously computed sets H, it holds that either:
   - H_i and H are disjoint 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 $\mathcal{F}_{\rm SA}$. Otherwise, $\mathcal{S}$ halts and outputs $\mathsf{fail1}$.
4. Whenever $\mathcal{S}$ receives a message $(\mathsf{send}, sid, P_i, P_j, m)$ from $\mathcal{F}_{\rm SA}$ where P_i is honest, simulator $\mathcal{S}$ simulates the actions of an honest 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 $\mathcal{F}_{\rm SA}$. Likewise, if P_j is not in the same authentication set as P_i, then $\mathcal{S}$ sends the appropriate send message to $\mathcal{F}_{\rm SA}$ itself. Then, $\mathcal{S}$ sends $\mathcal{F}_{\rm SA}$ the message $(\mathsf{deliver}, P_j, P_i, m)$, instructing it to deliver m to P_i from P_j.[7] If the request is not fulfilled then $\mathcal{S}$ halts and outputs $\mathsf{fail2}$.
6. Whenever $\mathcal{A}$ corrupts a party P_i, simulator $\mathcal{S}$ hands $\mathcal{A}$ the state of the internally simulated P_i.

It is straightforward to verify that as long as $\mathcal{S}$ does not output $\mathsf{fail1}$ or $\mathsf{fail2}$, 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 honest parties. In addition, the local outputs of the honest parties in the internal simulation correspond exactly to the outputs of the actual honest parties in the ideal model.) It therefore suffices to show that $\mathcal{S}$ outputs a fail message with at most negligible probability.

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

[7] Note that if P_j is honest 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 $\mathsf{deliver}$ message.

1. *There exist two honest 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 non-negligible probability, notice that $\mathcal{S}$ only defines sets H_i and H_j for parties that conclude the Link Initialization portion of the protocol and places P_i and P_j in the same set if they received each others "authentic" verification keys. By the signatures sent at the end of link initialization phase, it follows that either at least one of the parties aborts, or the adversary forged a signature relative to either P_i or P_j's verification key, or P_i and P_j both conclude with the same sid. (We note that the reduction here to the security of the signature scheme is straightforward.)
2. *There exist two sets $H_i \neq H_j$ that are not disjoint:* Let $P_i \in H_i \cap H_j$ be an honest party. Then, using the same arguments as above, except with negligible probability, P_i must have the same sid as all the honest parties in H_i *and* all the honest parties in H_j. Thus, all of the parties in $H_i \cup H_j$ have the same sid. Since this sid 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 $H_i = H_j$.

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 honest, 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 honest, 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 $\mathcal{F}_{\text{SA}}$. This implies that $\mathcal{S}$ did not generate a signature on (P_i, m, c) in the internal simulation (see step 4 of the simulation by $\mathcal{S}$), and yet P_i received a valid signature on this message. Thus, it follows that $\mathcal{A}$ must have forged a signature relative to the honest P_j's key. As above, such an adversary can be used to break the signature, and the actual reduction is straightforward. We conclude that the views of $\mathcal{Z}$ in the two interactions are statistically close. ■

4 General Functionalities in the Stand-Alone Model

In this section, we prove the following theorem:

Theorem 3 (Theorem 1 – restated): *Assume the existence of collision-resistant hash functions and enhanced trapdoor permutations, and consider the stand-alone model with no setup whatsoever. Then, for any probabilistic polynomial-*

time multiparty functionality $\mathcal{F}$ there exists a protocol that securely computes the split functionality $\mathsf{s}\mathcal{F}$, in the presence of static, malicious adversaries.

Theorem 3 is obtained by combining Protocol 1 for securely computing $\mathcal{F}_{\mathrm{SA}}$ with the protocol of [23] for securely computing any functionality in the setting of bounded-concurrency. Recall that in this model, there is an a priori bound on the number of protocol executions that can take place. As we have remarked above, in the setting considered here, we know that at most n concurrent executions can take place in a stand-alone execution with n parties in the unauthenticated model. Therefore, bounded concurrency suffices.

Our protocol for securely computing any n-party split functionality $\mathsf{s}\mathcal{F}$ works by first running the link initialization stage of Protocol 1, and obtaining a session identifier *sid* from this phase. Then, the protocol of [23] for securely computing $\mathcal{F}$ (under n-bounded concurrent composition) is executed, using the identifier *sid* and authenticating all messages sent and received as described in Protocol 1.[8]

The intuition behind the security of this protocol is that $\mathcal{F}_{\mathrm{SA}}$ guarantees that all the honest parties in a given authentication set H are essentially connected via pairwise authenticated channels. Thus, the execution of the protocol of [23] in our unauthenticated setting is the same as an execution of the protocol of [23] in the *authenticated channels model*, where the participating parties are comprised of the honest parties in H and $n-|H|$ corrupted parties. Now, in the unauthenticated model (by the definition of $\mathsf{s}\mathcal{F}$), the adversary is allowed to play the role of the $n-|H|$ parties not in H. Therefore, the above protocol suffices for securely computing the split functionality $\mathsf{s}\mathcal{F}$.

We note that the protocol of [23] relies on the existence of collision-resistant hash functions and enhanced trapdoor permutations. Furthermore, given any parameter m that is polynomial in the security parameter (and, in particular, setting m to equal the number of parties n), it is possible to obtain a protocol that remains secure for up to m concurrent executions, where in each execution any subset of the parties may participate. We note that these subsets may overlap in an arbitrary way, and security is still guaranteed. This point is *crucial* for our above use of the protocol. Namely, in order to prove security we actually consider a *virtual network* of $2n$ parties $P_1,\ldots,P_{2n}$ where all parties $P_{n+1},\ldots,P_{2n}$ are corrupted. Then, for any authentication set $H\subseteq\{P_1,\ldots,P_n\}$ we consider an execution of the protocol of [23] with the subset of parties comprised of every $P_i\in H$, and every P_{n+j} for $P_j\notin H$. Note that this defines a subset of exactly n parties, where every party *not* in H is controlled by the adversary, as required. The important point to note now, however, is that some P_{n+j} may participate in many different executions of the protocol of [23]. It is therefore crucial that [23] remains secure when arbitrary subsets of parties run the protocol.

References

1. B. Barak. How to Go Beyond the Black-box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[8] More formally, our protocol works in the $\mathcal{F}_{\mathrm{SA}}$-hybrid model, and uses the authentication mechanism provided by $\mathcal{F}_{\mathrm{SA}}$ in order to run the protocol of [23].

2. M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations. In 20*th STOC*, pages 1–10, 1988.
3. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
4. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS*, pages 136–145, 2001. Full version available at `http://eprint.iacr.org/2000/067`.
5. R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pages 19–40, 2001.
6. R. Canetti, S. Halevi, J. Katz, Y. Lindell and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *EUROCRYPT 2005*, Springer-Verlag (LNCS 3494), pages 404–421, 2005.
7. R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions . In *EUROCRYPT '03*, Springer-Verlag (LNCS 2656), pages 68–86, 2003.
8. R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In 34*th STOC*, pp. 494–503, 2002.
9. R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO 2003*, Springer-Verlag (LNCS 2729), pages 265–281, 2003.
10. D. Chaum, C. Crepeau and I. Damgard. Multiparty Unconditionally Secure Protocols. In 20*th STOC*, pages 11–19, 1988.
11. D. Dolev, C. Dwork and M. Naor. Non-malleable Cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
12. C. Dwork and M. Naor. Pricing via Processing or Combating Junk Mail. In *CRYPTO'92*, Springer-Verlag (LNCS 740), pages 139–147, 1992.
13. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Detectable Byzantine Agreement Secure Against Faulty Majorities. 21*st PODC*, pp. 118–126, 2002.
14. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pages 408–432, 2001.
15. S. Goldwasser, S. Micali and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. on Computing*, 17(2):281–308, 1988.
16. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
17. O. Goldreich, S. Micali and A. Wigderson. How to Play Any Mental Game. In 19*th STOC*, pages 218–229, 1987.
18. O. Goldreich. *Foundations of Cryptography – Vol. 1.* Cambridge Univ. Press, 2001.
19. O. Goldreich. *Foundations of Cryptography – Vol. 2.* Cambridge Univ. Press, 2004.
20. Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In 35*th STOC*, pages 683–692, 2003.
21. Y. Lindell. Lower Bounds for Concurrent Self Composition. In *1st TCC*, Springer-Verlag (LNCS 2951), pages 203–222, 2004.
22. M. Nguyen and S. Vadhan. Simpler Session-Key Generation from Short Random Passwords. In *1st TCC*, Springer-Verlag (LNCS 2951), pages 428–445, 2004.
23. R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. In 36*th STOC*, pages 232–241, 2004.
24. R. Pass and A. Rosen Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In 44*th FOCS*, pages 404–413, 2003.
25. A.C. Yao. How to Generate and Exchange Secrets. 27*th FOCS*, pp. 162–167, 1986.