Protocol Initialization for the Framework of Universal Composability

Boaz Barak* Yehuda Lindell† Tal Rabin†

January 8, 2004

Abstract

Universally composable protocols (Canetti, FOCS 2000) are cryptographic protocols that remain secure even when run concurrently with arbitrary other protocols. Thus, universally composable protocols can be run in modern networks, like the Internet, and their security is guaranteed. However, the definition of universal composition actually assumes that each execution of the protocol is assigned a unique *session identifier*, and furthermore, that this identifier is known to all the participating parties. In addition, all universally composable protocols assume that the set of participating parties and the specification of the protocol to be run are a-priori agreed upon and known to all parties. In a decentralized network like the Internet, this setup information must be securely generated by the parties themselves. In this note we formalize the setup problem and show how to securely realize it with a simple and highly efficient protocol.

Key words: Universal composition, secure multiparty computation

1 Introduction

The framework of universal composability (UC) [1] is a powerful tool aiding in the design and analysis of cryptographic protocols. The central feature of this framework is a robust composition theorem that states the following: Any protocol that is proven secure (as stand-alone) under the definition of universal composability, is guaranteed to remain secure when run concurrently with arbitrary other protocols. Thus, for example, universally composable protocols can be safely used in real settings like the Internet, where many sets of parties run many different protocols concurrently.

In order to formalize the notion of security under concurrent composition, the framework of universal composability assumes that for every protocol execution there is a unique session identifier that is known (and agreed upon) by all parties. In addition, all known universally composable protocols assume that all parties know (and agree upon) the set of participating parties and the protocol to be executed. We stress that the above assumptions are not merely technicality; rather they are heavily relied on by known construction in order to achieve security. (For example, the commitment protocol of [2], that forms the basis for their entire construction, binds the session identifier to the committed value, in order to prevent the copying of commitments from one execution to another.) Since the actual security of universally composable protocols relies on the

*Institute for Advanced Study, Princeton NJ, USA. email: boaz@ias.edu. This work was carried out while visiting the IBM T.J.Watson Research Center.

†IBM T.J.Watson Research, 19 Skyline Drive, Hawthorne, NY 10532, USA. email: lindell@us.ibm.com, talr@watson.ibm.com.

"correctness" of the setup information (e.g., that the session identifier is indeed unique and common to all participants), a secure setup protocol must be used. In particular, it does not suffice to allow one party (say, the party initiating the protocol execution) to choose the session identifier and distribute it to all the participants.

Our results. Before describing how the problem of obtaining the required setup can be solved, we must first consider how protocols are invoked in a decentralized network. Specifically, how do parties decide to start running a specific protocol, and with whom? Typically, there exists a protocol *initiator* who "invites" a set of parties to participate in some execution. If the protocol is an auction, then the initiator is most likely to be the auctioneer. Then, parties who receive this "invitation" may participate if they wish. (Prior to the beginning of the protocol execution, parties may approach the initiator and express interest in participating. However, the final decision is in the initiator's hands.) In this note, we provide a formal definition of the "initialization and setup problem"; i.e., the problem of securely obtaining the protocol setup assumptions of the UC framework, in a setting where any party can initiate protocol executions. We then present a very efficient and simple protocol that solves this problem without any central or trusted authorities, and without any assumed threshold regarding the number of corrupted parties.

Our protocol guarantees that *globally* unique session identifiers are used in each execution. That is, the adversary is unable to cause two (or more) different protocol executions to have the same session identifier. At first glance, this may seem to contradict the impossibility results of [4]. In [4], it is proven that in the setting of parallel or concurrent (stateless) composition, it is impossible to achieve authenticated Byzantine agreement when at least a third of the parties are corrupted. In addition, they prove that authenticated Byzantine agreement under concurrent composition *can* be achieved if unique identifiers are somehow initially obtained. Combining these two results, it follows that in the setting of authenticated Byzantine agreement under concurrent composition, it is impossible for the parties to generate unique session identifiers by themselves (because if they could, then they could achieve authenticated Byzantine agreement, in contradiction to the first result). In contrast, we *do* generate unique session identifiers, without any external trusted help and for any number of corrupted parties. The reason that no contradiction actually exists is due to the fact that the requirements on termination are different. That is, the definition of authenticated Byzantine agreement requires that the parties always successfully obtain output. In contrast, in our setting there is no requirement that the protocol will successfully conclude. Rather, it is guaranteed that *if* a party concludes the setup protocol, then it has obtained a unique session identifier. (We also present a variant of the setup protocol where successful termination *is* guaranteed, but only if the initiator is honest.)

Security in the initiator model. As we have mentioned, in this paper, we consider a model where a protocol initiator chooses the set of participating parties as it wishes. It should be noted that in such a case, an adversarial initiator can choose this set so that only *one* party is honest. It is important to be aware of this because in some cases this can have undesired effects. For example, consider a secure protocol for polling the voting patterns of the population. If only one party in the poll is honest, then the adversary can learn the exact vote of this party.

The framework of universal composability. We refer the reader to [1, 2] for a description and definition of the framework of universal composability. Due to differing versions of the framework regarding message delivery, we briefly clarify what we consider here. In the ideal model, all messages between the honest parties and the ideal functionality are delivered immediately without any involvement from the adversary. (This is the model considered in the latest version of [1].) In the basic real model that we consider, the adversary sees all the messages sent, and delivers or blocks these messages at will (but cannot modify them). However, we will also consider a real model where message delivery is guaranteed between honest parties. We note that in this work we consider adaptive, malicious adversaries.

2 The Initialization and Setup Problem

2.1 Problem Definition

In a decentralized network, any party can initiate a protocol execution by inviting some subset of parties to participate. Of course, some of these parties may not want to participate, and may choose not to. This models real settings where parties notify the initiator of their interest to participate, and the initiator then chooses some subset of interested parties as it wishes. The naive way to implement such a scenario is to simply have this protocol initiator send an "initiate" message to all the parties who will participate in the execution. However, this initiate message must also include a unique session identifier, the identities of all participating parties, and the specification of the functionality that is to be called. We note that secure protocols all assume that this information is a-priori known to all parties; the framework of universal composability is no exception. Thus, a dishonest initiator may provide different sets of identities to different parties and may choose a session identifier that has already been used in the past (or is even being used in a concurrently running session; see [4] for an example of where copying session identifiers can be very detrimental to protocols that assume uniqueness). We therefore define an initialize functionality, denoted $\mathcal{F}_{\text{init}}$, that prevents the adversary from such behavior. In this functionality, the initiating party chooses the set of participating parties and the specification of the functionality to be computed by these parties. However, the session identifier is chosen by the adversary, with the only limitation that it *must be unique*. Since uniqueness is the only requirement for session identifiers, allowing the adversary the power to choose the specific unique string does not compromise the security of the system. We stress that the output of the initialize functionality is such that all the parties receive the *same* session identifier, the *same* set of identities, and the *same* functionality specification. This therefore provides the parties with the setup information needed for running a secure protocol. The functionality is defined in Figure 1.
We note that the adversary $\mathcal{S}$ has full control over which parties receive the invoke message from the functionality. In fact, a party only receives output after $\mathcal{S}$ explicitly instructs the functionality to send it to the party. This corresponds to the basic model considered for the UC framework where message delivery is not guaranteed in the real model. In this case the adversary can always prevent a party from receiving output by blocking its last message in the protocol. The ideal functionality therefore also provides the adversary with this capability. The case where message delivery *is* guaranteed in the real model is dealt with in Section 2.3.

Using the $\mathcal{F}_{\text{init}}$ functionality. Recall that the UC framework assumes that when a protocol π contains an ideal call to a functionality $\mathcal{F}$, then all the parties have already agreed upon the set of participating parties, the specification of the functionality they are calling, and the unique session identifier *sid*. Ensuring that this holds is seen to be the "responsibility" of the calling protocol π. In a decentralized network, the functionality $\mathcal{F}_{\text{init}}$ can be called before the first call to a functionality $\mathcal{F}$. Specifically, in order to initiate an execution of $\mathcal{F}$, the initiator first calls $\mathcal{F}_{\text{init}}$. Then, after a party P_j obtains output $(\mathsf{invoke}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$, it can proceed to call $\mathcal{F}$ with the set of participating parties $\mathcal{P}$ and session identifier *sid*.

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

$\mathcal{F}_{\text{init}}$, with fixed session identifier 0, runs in the universe with parties $\mathcal{U}$ and an adversary $\mathcal{S}$. When called for the first time, it sets $\mathcal{H}ist = \emptyset$.

- Upon receiving a value $(\mathsf{initiate}, 0, \langle P_i, \mathcal{P}, \mathcal{F}\rangle)$ from P_i, where $\mathcal{P} \subseteq \mathcal{U}$, execute the following:
  1. Send $(\mathsf{initiate}, 0, \langle P_i, \mathcal{P}, \mathcal{F}\rangle)$ to $\mathcal{S}$.
  2. Upon receiving back $(\mathsf{set\text{-}id}, 0, \langle sid', P_i, \mathcal{P}, \mathcal{F}\rangle)$ from $\mathcal{S}$, do the following:
     (a) If $sid' \in \mathcal{H}ist$, choose an arbitrary $sid \notin \mathcal{H}ist$.
     (b) If $sid' \notin \mathcal{H}ist$, set $sid \leftarrow sid'$.
     (c) Update $\mathcal{H}ist \leftarrow \mathcal{H}ist \cup \{sid\}$.
     (d) Send $(\mathsf{invoke}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$ to $\mathcal{S}$.
  3. Upon receiving a message $(\mathsf{send\text{-}output}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle, P_j)$ from $\mathcal{S}$:
     (a) If $P_j \in \mathcal{P}$ and it has not yet been sent the invoke message with $\langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle$, send it $(\mathsf{invoke}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$.

Figure 1: The Initialize Functionality

We stress that there is only a single copy of the $\mathcal{F}_{\text{init}}$ functionality, and it has the fixed session identifier 0. (If it was necessary to agree upon a unique identifier for every invocation of $\mathcal{F}_{\text{init}}$, then we would have solved nothing.) Technically, we can use the same identifier for every call to $\mathcal{F}_{\text{init}}$, because the functionality does not need to associate different messages from different parties within a single call. (If different parties did send messages to $\mathcal{F}_{\text{init}}$ in a single call, then some mechanism, like a unique identifier, would be needed to ensure that messages would be correctly associated.) We note that for the interaction between the functionality and the adversary it suffices for the functionality to "identify" the execution via the values sent in the initiate message.

2.2 Protocol Construction

We now present a simple protocol that securely computes $\mathcal{F}_{\text{init}}$ in the UC framework. The basic idea behind the protocol is for the parties to jointly generate the session identifier sid by concatenating n-bit random strings (n denotes the security parameter). Then, a party will "accept" the final sid only if its random string is included. This means that honest parties will only accept unique identifiers, because an sid with an n-bit random string is unique except with negligible probability. The set of participating parties and the functionality specification are also appended to the session identifier sid, in order to ensure that all parties that conclude with the same sid agree on the participating parties and functionality specification. We note that a corrupted initiator can cause different parties to conclude with different identifiers. However, this is equivalent to the initiator running *multiple* setups with different parties. Since it can always do this in the ideal model, it is also allowed to do so in a real protocol execution. The protocol is presented in Figure 2.

Theorem 1 *Protocol Π_{init} securely computes the functionality $\mathcal{F}_{\text{init}}$ in the UC framework, in the presence of adaptive, malicious adversaries and in an asynchronous model where message delivery is not guaranteed.*

Protocol Π_{init}

1. Upon input $(\mathsf{initiate}, 0, \langle P_i, \mathcal{P}, \mathcal{F}\rangle)$, party P_i chooses a random value $sid_i \in_R \{0,1\}^n$ and sends $(\text{start}_{\Pi_{\text{init}}}, sid_i, P_i, \mathcal{P}, \mathcal{F})$ to all parties $P_j \in \mathcal{P}$. (Identifier sid_i is used to enable the parties to distinguish messages from this execution from messages from other executions.)
2. Each party P_j that receives the $\text{start}_{\Pi_{\text{init}}}$ message chooses a random string $r_j \in_R \{0,1\}^n$ and sends (sid_i, r_j) to P_i. (If $P_j \notin \mathcal{P}$ then it ignores the message.)
3. Denote the parties in $\mathcal{P}$ by $P_{j_1}, \ldots, P_{j_\ell}$, where the parties are sorted in ascending order of identities (i.e., $j_i < j_{i+1}$ for every i).

   Then, when party P_i receives the (sid_i, r_j) messages from all parties $P_j \in \mathcal{P}$, it chooses $r_i \in_R \{0,1\}^n$, sets $sid = r_i, r_{j_1}, \ldots, r_{j_\ell}, \mathcal{P}, \mathcal{F}$ and sends (sid_i, sid) to all parties $P_j \in \mathcal{P}$.
4. When party P_j receives (sid_i, sid) from P_i it checks the following:
   (a) The set of parties and functionality description appearing at the end of sid equals the set $\mathcal{P}$ and functionality $\mathcal{F}$ that it received from P_i in the first message.
   (b) The random value r_j that P_j chose appears in sid in its "correct" position.

   If both these hold, then P_j outputs $(\mathsf{invoke}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$. Otherwise, it outputs nothing.

Figure 2: Protocol for securely computing the $\mathcal{F}_{\text{init}}$ functionality

Proof: Let $\mathcal{A}$ be a real-model adversary. Then, we construct an ideal-model simulator/adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish a real execution of Π_{init} with $\mathcal{A}$ from an ideal execution of $\mathcal{F}_{\text{init}}$ with $\mathcal{S}$.

The simulator $\mathcal{S}$ invokes $\mathcal{A}$ and emulates an execution of Π_{init}, while playing all of the uncorrupted parties. We distinguish between the case that the initiating party P_i is corrupted at the time that it sends its $\text{start}_{\Pi_{\text{init}}}$ message to the parties in $\mathcal{P}$, from the case that it is uncorrupted:

P_i **is corrupted:** In this case, $\mathcal{A}$ sends a series of $\text{start}_{\Pi_{\text{init}}}$ messages to honest parties, in the name of P_i. For every such $(\text{start}_{\Pi_{\text{init}}}, sid_i, P_i, \mathcal{P}, \mathcal{F})$ message that $\mathcal{A}$ sends an honest party P_j, simulator $\mathcal{S}$ chooses $r_j \in_R \{0,1\}^n$ and internally sends (sid_i, r_j) back to $\mathcal{A}$. ($\mathcal{S}$ simulates P_j sending this reply, unless $P_j \notin \mathcal{P}$, in which case $\mathcal{S}$ does nothing.)

When $\mathcal{A}$ sends another message (sid_i, sid) to the honest P_j in the emulation, simulator $\mathcal{S}$ checks that the random string r_j appears in sid in the correct position and that $\mathcal{P}$ and $\mathcal{F}$ appear at the end of sid. If no, then it does nothing. Otherwise, there are two possibilities:

1. *$\mathcal{S}$ has already sent a* $(\mathsf{set\text{-}id}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$ *message to* $\mathcal{F}_{\text{init}}$*:*
   In this case, $\mathcal{S}$ sends the message $(\mathsf{send\text{-}output}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle, P_j)$ to $\mathcal{F}_{\text{init}}$, instructing it to send output to P_j.
2. *$\mathcal{S}$ has not yet sent a* $(\mathsf{set\text{-}id}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$ *message to* $\mathcal{F}_{\text{init}}$*:* In this case, $\mathcal{S}$ first instructs P_i to send $(\mathsf{initiate}, 0, \langle P_i, \mathcal{P}, \mathcal{F}\rangle)$ to the functionality $\mathcal{F}_{\text{init}}$. Functionality $\mathcal{F}_{\text{init}}$ then sends $(\mathsf{initiate}, 0, \langle P_i, \mathcal{P}, \mathcal{F}\rangle)$ to $\mathcal{S}$, and $\mathcal{S}$ replies with $(\mathsf{set\text{-}id}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$, for the above sid. Finally, after $\mathcal{S}$ receives back the $(\mathsf{invoke}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$ from $\mathcal{F}_{\text{init}}$, it sends the message $(\mathsf{send\text{-}output}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle, P_j)$ to $\mathcal{F}_{\text{init}}$, instructing it to send output to P_j.

P_i **is uncorrupted:** In this case, $\mathcal{S}$ receives a message $(\mathsf{initiate}, 0, \langle P_i, \mathcal{P}, \mathcal{F}\rangle)$ from $\mathcal{F}_{\text{init}}$. Then, $\mathcal{S}$ simulates P_i sending $(\text{start}_{\Pi_{\text{init}}}, sid_i, P_i, \mathcal{P}, \mathcal{F})$ to all parties $P_j \in \mathcal{P}$, for a random $sid_i \in_R$

$\{0,1\}^n$. In addition, $\mathcal{S}$ simulates all the honest parties P_j replying with (sid_i, r_j). Then, $\mathcal{S}$ waits until $\mathcal{A}$ delivers all of the (sid_i, r_j) messages from the honest parties to P_i, and until $\mathcal{A}$ sends $(sid_i, r_{j'})$ messages to P_i from all the corrupted parties $P_{j'}$. Following this, $\mathcal{S}$ computes $sid' = r_i, r_{j_1}, \ldots, r_{j_\ell}, \mathcal{P}, \mathcal{F}$ and sends $(\mathsf{set\text{-}id}, 0, \langle sid', P_i, \mathcal{P}, \mathcal{F}\rangle)$ to $\mathcal{F}_{\text{init}}$.

Now, let $sid = sid'$. Then, $\mathcal{S}$ simulates P_i writing (sid_i, sid) messages on its outgoing communication tape for all $P_j \in \mathcal{P}$. Then, $\mathcal{S}$ sends a $(\mathsf{send\text{-}output}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle, P_j)$ message to $\mathcal{F}_{\text{init}}$ whenever $\mathcal{A}$ delivers the (sid_i, sid) message from P_i to P_j in the emulation.

Dealing with corruptions: Notice that in the above-described simulation, $\mathcal{S}$ simply plays the roles of all the honest parties and sends $\mathcal{F}_{\text{init}}$ the initiate message with the *sid* that is generated by the protocol. Therefore, if $\mathcal{A}$ corrupts P_i at sometime during the execution, $\mathcal{S}$ simply continues by following the instructions for the case that P_i is corrupted. Likewise, corruptions of parties P_j $(j \neq i)$ are dealt with in a straightforward way (they have no secret information, so there is no private state to be revealed).

Analysis of $\mathcal{S}$: We now prove that the simulator $\mathcal{S}$ is such that no environment $\mathcal{Z}$ can distinguish between an ideal execution with $\mathcal{F}_{\text{init}}$ and $\mathcal{S}$, from a real execution of Protocol Π_{init} with $\mathcal{A}$. First note that the honest parties have no secret information. Therefore, $\mathcal{S}$ perfectly simulates a real execution of Protocol Π_{init} for $\mathcal{A}$. Furthermore, assuming that the identifier *sid* sent by $\mathcal{S}$ is *always* unique, the honest parties all output the same values as they would in a real execution. This is the case because the sid' value sent by $\mathcal{S}$ in the set-id message to $\mathcal{F}_{\text{init}}$ is simply the *sid* value that the honest parties would receive from P_i in a real execution. Thus, it remains to show that the value sid' sent by $\mathcal{S}$ is unique, except with negligible probability (we prove this for the case that at least one honest party is participating, otherwise it is of no significance).

In the case that the initiator P_i is corrupted and some P_j is not corrupted, simulator $\mathcal{S}$ only sends $(\mathsf{set\text{-}id}, 0, \langle sid', P_i, \mathcal{P}, \mathcal{F}\rangle)$ to $\mathcal{F}_{\text{init}}$ if the value r_j appears in sid' in the correct position. Recall that $r_j \in_R \{0,1\}^n$ was chosen randomly by $\mathcal{S}$. Therefore, the probability that this sid' was previously used is at most $\mathsf{time}(\mathcal{A})/2^n$, where $\mathsf{time}(\mathcal{A})$ denotes the running time of adversary $\mathcal{A}$. Since $\mathcal{A}$ runs in polynomial time, this probability is negligible. Next, if P_i is not corrupted, then sid' begins with a random value $r_i \in_R \{0,1\}^n$. As above, this means that the probability that sid' has been used before is negligible.

In order to complete the proof, note that $\mathcal{S}$ instructs $\mathcal{F}_{\text{init}}$ to deliver output to a party P_j at the same time that $\mathcal{A}$ delivers the message (sid_i, sid) from P_i to P_j in the emulation. Therefore, honest parties obtain outputs at the same time in a real and ideal execution. ■

Notice that in the case that the initiator is corrupted, Protocol Π_{init} provides almost no "agreement" guarantees. Specifically, every honest party may end up with a different session identifier. However, this does not contradict the security of the protocol because in the ideal model, a corrupted initiator may initiate many different sessions. Furthermore, the adversary can deliver output to only one honest party in each of these sessions. This is equivalent to the situation in the real model where each party concludes with a different session identifier.

2.3 Protocol Initialization With Guaranteed Termination

The real model that we have considered until now is one where the adversary has control over all message delivery between the honest parties. In this case, the adversary can always prevent an honest party from receiving output, by not delivering its last message in the protocol. As we have mentioned, the definition of $\mathcal{F}_{\text{init}}$ for the ideal model therefore explicitly allows the adversary $\mathcal{S}$ to decide when (if at all) an honest party receives its output. In this section, we consider a real model where message delivery *is* guaranteed between honest parties. For simplicity, we assume that the network is *synchronous*, although we could also consider a partially asynchronous network where messages can be delayed for at most ϵ units of time, for a given and publicly known ϵ. In this case, we would like to ensure successful termination of the protocol, with all parties receiving output. This will ensure that if the secure protocol to be run following the initialization guarantees output delivery, then the composition of Protocol $\Pi_{\rm init}$ with the secure protocol will also guarantee output delivery.

We first modify $\mathcal{F}_{\rm init}$ so that output delivery is guaranteed. This cannot be done in a naive way because then Byzantine (or authenticated Byzantine) agreement would be implied, in contradiction to known impossibility results [3, 5, 4]. Rather, we require that if the initiator is honest, then it is guaranteed that all the honest parties conclude with the same session identifier, the same set of participating parties and the same functionality specification. In contrast, if the initiator is not honest, then the effect achieved is like in the previous section. (That is, the honest parties may conclude with different session identifiers, but the identifiers are always unique and fulfill the requirements of a secure setup.)

Modifications to functionality $\boldsymbol{\mathcal{F}_{\rm init}}$. We modify $\mathcal{F}_{\rm init}$ as follows. First, the initiator P_i sends a unique identifier sid_0 in its $\mathsf{initiate}$ message to the functionality. Then, if the adversary $\mathcal{S}$ does not reply with sid' in the next round and $sid_0 \notin \mathcal{H}ist$, the session identifier in the output is set to sid_0. If the adversary $\mathcal{S}$ does reply with sid', then Step 2 of $\mathcal{F}_{\rm init}$ remains the same. The second modification is that instead of $\mathcal{S}$ instructing the functionality to $\mathsf{send\text{-}output}$ in Step 3, these instructions are provided by the initiator. Of course, an honest initiator is expected to choose sid_0 randomly (to ensure uniqueness) and to instruct the functionality to send output to all parties. The result of the above is that in the case that the initiator P_i is honest, successful termination is guaranteed (i.e., all parties receive $(\mathsf{invoke}, 0, \langle sid, P_i, \mathcal{P}, \mathcal{F}\rangle)$ where sid is a unique identifier and $\mathcal{P}, \mathcal{F}$ are as chosen by P_i. On the other hand, if the initiator is corrupted, then the same effect as the original $\mathcal{F}_{\rm init}$ is achieved.

Modifications to protocol $\boldsymbol{\Pi_{\rm init}}$. With a slight modification, Protocol $\Pi_{\rm init}$ can be made to securely realize the modified initialize functionality. The required modification to the protocol is in Step 3 (see Figure 2): Instead of Party P_i waiting until it receives *all* of the (sid_i, r_j) messages, it waits one round (recall, we assume here that the network is synchronous). If a party P_j does not send its message in the next round, then P_i chooses $r_j \in_R \{0,1\}^n$ and continues *as if* P_j sent (sid_i, r_j). Notice that once P_i sends the (sid_i, r_j) messages, all parties receive output. Therefore, this modification to Protocol $\Pi_{\rm init}$ ensures successful termination when the initiator P_i is honest.

References

[1] 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`.

[2] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In 34*th STOC*, pages 494–503, 2002.

[3] L. Lamport, R. Shostack, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[4] Y. Lindell, A. Lysysanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *34th STOC*, pages 514–523, 2002.

[5] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.