Simplified Universal Composability Framework

Douglas Wikström(✉)

KTH Royal Institute of Technology, Stockholm, Sweden
dog@kth.se

Abstract. We introduce a simplified universally composable (UC) security framework in our thesis (2005). In this paper we present an updated more comprehensive and illustrated version. The introduction of our simplified model is motivated by the difficulty to describe and analyze concrete protocols in the full UC framework due to its generality and complexity.

The main differences between our formalization and the general UC security framework are that we consider: a fixed number of parties, static corruption, and simple ways to bound the running times of the adversary and environment. However, the model is easy to extend to adaptive adversaries. Authenticated channels become a trivial ideal functionality.

We generalize the framework to allow protocols to securely realize other protocols. This allows a natural and modular description and analysis of protocols.

We introduce invertible transforms of models that allow us to reduce the proof of the composition theorem to a simple special case and transform any hybrid protocol into a hybrid protocol with at most one ideal functionality. This factors out almost all of the technical details of our framework to be considered when relating our framework to any other security framework, e.g., the UC framework, and makes this easy.

1 Introduction

Canetti [3], and independently Pfitzmann and Waidner [11] propose security frameworks for reactive processes. Both frameworks have composition theorems, and are based on older definitional work. The initial ideal-model based definitional approach for secure function evaluation is informally proposed by Goldreich, Micali, and Wigderson in [6]. The first formalizations appear in Goldwasser and Levin [7], Micali and Rogaway [10], and Beaver [1]. Canetti [2] presents the first definition of security that is preserved under composition. See [2,3] for an excellent background.

The basic approach of all these models is the same. An ideal functionality is defined that implicitly captures the functionality and security properties we expect from a real protocol. The real protocol is then said to be secure if it is indistinguishable from the ideal functionality by any efficient distinguisher. However, in an execution of the real protocol the adversary may influence the execution or extract information that it passes on to the distinguisher. Thus, we introduce an simulation adversary (simulator) that is given the same task,

E. Kushilevitz and T. Malkin (Eds.): TCC 2016-A, Part I, LNCS 9562, pp. 566–595, 2016.
DOI: 10.1007/978-3-662-49096-9_24

but when interacting with the ideal functionality. The ideal functionality is secure by inspection, so the simulation adversary can by definition not attack the ideal functionality in any meaningful way. Instead it must simulate a real attack to the distinguisher. The definition of security then says that if for every real adversary there exists an simulation adversary such that no efficient distinguisher can distinguish: (1) an interaction with the real protocol and the real adversary from (2) an interaction with the ideal functionality and the simulation adversary, then the real protocol is said to securely realize the ideal functionality.

The UC framework is an ambitious attempt to capture the security of a wide range of settings in a uniform way, but the original UC framework was flawed in several ways. The most recent version of the online paper [3] contains a discussion about the issues and pointers to relevant literature. However, the core ideas of the UC framework are correct, and there are no flaws in the basic instantiations needed to prove the security of practical protocols. In this paper we detail one possible instantiation, but before we do so, we point out the main areas where our particular instantiation is more restricted, and hence less complex, than the general framework.

Canetti assumes the existence of an "operating system" that takes care of the instantiation of subprotocols when needed. This is necessary to handle dynamically instantiated subprotocols, but in our application we may assume that all subprotocols are instantiated at the start of the execution. This means that we can view each instance of a subprotocol as a separate Turing machine that exists from scratch that interacts with the invoking protocol with a predefined session identifier.

Canetti models an asynchronous communication network, where the adversary has the power to delete, modify, and insert any messages of his choice. To do this he is forced to give details for exactly what the adversary is allowed to do to messages passed in different ways between interactive Turing machines, which quickly becomes quite complex. We instead factor out all aspects of the communication network into a separate concrete "communication model"-machine. The real, ideal, and hybrid models are then defined solely by how certain machines are linked. The adversary is defined as any interactive Turing machine, and how the adversary can interact with other machines also follows implicitly from the definitions of the real and ideal communication models. With our approach there is also no need for session identifiers.

The above means that the real, ideal, and hybrid models can not only be illustrated by a graph of connected parties, they *are* graphs of Turing machines in a very tangible way, which makes the composition theorem almost trivial.

There are several ways to model corruption in cryptographic protocols. In this paper, we only consider *static* corruption, i.e., the adversary must decide which parties to corrupt before the execution starts. However, it is straightforward to extend the model to adaptive corruption as explained in Remark 2. Even dynamic adversaries could be handled in a similar way, so there is no inherent restriction to static adversaries.

1.1 Contribution

We present a precise and workable security framework using modularized definitions that are easily verified to be sound. Abstractions emerge in a natural way that are firmly grounded in the underlying definitions. Although our treatment may initially seem more complex than the description of the UC framework, the actual content is captured faithfully in simple drawings that are enough to understand the framework, and the composition theorem becomes almost trivial.

Explicit invertible transforms are introduced that can turn any hybrid model into a hybrid model with a single ideal functionality (or a real model). Thus, it suffices to consider how the security of such a protocol in our simplified UC framework relates to its security in any other security framework, in particular the UC framework. This also immediately generalizes the single composition theorem to allow multiple compositions.

We introduce a novel generalization the UC framework and other frameworks we are aware of in that the definition of security captures the case where a hybrid protocol securely realizes another hybrid protocol, and not only ideal functionalities. This allows a novel type of proof that is not only based on securely realizing ideal functionalities and applying the composition theorem. We give natural examples where this technique is applicable.

The essential restriction in our framework compared to general UC is that the set of parties and the protocol, including all subprotocols and ideal functionalities used, are determined at the start of the execution.

1.2 Related Work

Several frameworks have been proposed today, but we only mention two frameworks that perhaps are closest to our framework at a philosphical level.

Constructive cryptography was developed and proposed by Maurer and Renner [8,9] independently of our work. The design of cryptographic primitives and protocols in this framework is viewed as the construction of an ideal resource from assumed or real resources. It shares with our framework the aims of achieving simplicity and eliminating irrelevant artefacts. We have not carried out a detailed analysis of the relations between their model and ours, but we are currently corresponding with the authors.

In subsequent, but independent work, Canetti et al. [4] propose an alternative formalization of a simplified UC framework motivated by the same problems as we do, and to some extent they use also the same approach as we do. Their motivation and the restrictions they introduce compared to the full UC framework are the same. Several features of the formalization that distinguishes it from the UC framework are also similar, e.g., their explicit "router" corresponds to our "communication model".

We consider the main difference between our framework and theirs to be that they use top-down approach, whereas we gradually build the model from the bottom up. They explicitly relate their model to the general UC model. We instead provide transforms that allow us to relate our framework to any

other framework with ease, since today there are many proposals of security framework and it is nearly impossible to understand each framework sufficiently well to perform a valid comparison.

That said, we hope that the reader takes the time to read both papers, since they both attempt to capture the core ideas of the UC framework in a way that is easier to understand and use.

2 Interactive Turing Machines

Parties and algorithms are modeled as probabilistic Turing machines, but to be able to talk about multiple parties that interact with each other we need to augment this model with a notion of communication. We follow the approach of Goldreich [5] and Canetti [3] and define *interactive Turing machines*, but we replace the activation bit used by Goldreich by a slightly more complicated gadget to allow seamless treatment of multiparty protocols.

Definition 1 (Interactive Turing Machine). *An* interactive Turing machine *(ITM) is a Turing machine with the following tapes and tape heads in addition to its work tapes: a read-only* identity tape*, a read-only* security parameter tape*, a read-once* input tape*, a write-once* output tape*, a read-once* random tape*, a write-once* send head s*, a read-once* receive head r*, and two single-bit read/write activity heads a_s and a_r. The following restrictions apply to an ITM, where we use brackets to indicate the value stored in the cell pointed at by a tape head.*

1. *If* $([a_s], [a_r]) \in \{(0,0),(1,0)\}$*, then it is* inactive *and can not change its state in a state transition, or read, write, or move on any tape.*
2. *If* $([a_s], [a_r]) = (0,1)$*, then it is* active *and can change its state in a state transition.*
3. *A special instruction allows it to atomically: set* $([a_s], [a_r]) = (1,0)$ *and become inactive.*

Note that a single ITM is not a complete computational model, since some tape heads do not have matching tapes. Two ITM's are connected by adding

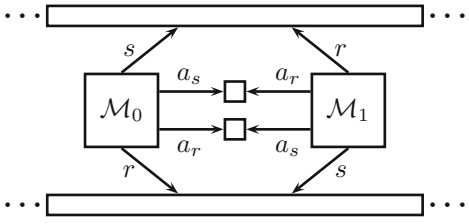

Fig. 1. The ITM's $\mathcal{M}_0$ and $\mathcal{M}_1$ share activation and send/receive tapes. The send head of $\mathcal{M}_0$ points to same tape as the receive head of $\mathcal{M}_1$ and vice versa. A corresponding configuration is used for the activation tapes. The figure does not contain the other tapes of the ITM's.

the missing tapes and pairing the write-once send head of one party with the read-once receive head of the other and the activity head a_s of one party with the activity head a_r of the other. Intuitively, the activation tapes implement an "activation token" that is passed back and forth between the parties. This is illustrated in Fig. 1. We denote the set of all ITM's by ITM.

3 Graph of Interactive Turing Machines

To connect multiple ITM's with each other without introducing extra tapes for each machine and thereby change the computational model, we introduce a gadget that plays the role of a router. A router is a Turing machine with several sets of tape heads that can share tapes with interactive Turing machines (ITM) or other routers.

Definition 2 (Router). *An* l-router *is a Turing machine with write-once* send heads *denoted* $s_0, \ldots, s_l$*, read-once* receive heads*, denoted* $r_0, \ldots, r_l$*, and single-bit read/write* activity heads $a_{s,i}$ *and* $a_{r,i}$ *for* $i = [0, l]$ *such that* $\sum_{i=0}^{k}([a_{s,i}] + [a_{r,i}]) \in \{0, 1\}$.

Active. *If* $[a_{r,i}] = 1$ *for some* $i \in [0, k]$*, then it is* active *and proceeds as follows.*

1. *To form a string* w *it reads and stores symbols from its* i*th receive tape using* r_i *until it encounters* $\perp$.
2. *If* $i = 0$ *then*
   - *if* $|w| \geq n$ *and the last* n *bits of* w *is an integer* $j \in [k]$*, then it writes* w *except the last* n *bits to its* j*th send tape using* s_j*, and*
   - *otherwise it writes* $\Diamond \| w$ *to its* 0*th send tape using* s_0.

   If $i \neq 0$*, then it sets* $j = 0$ *and writes* w *and a* n*-bit representation of* i *to its* 0*th send tape using* s_0.
3. *It sets* $([a_{s,j}], [a_{r,i}]) = (1, 0)$ *(as an atomic operation) to pass the activity token to the* j*th party.*

Inactive. *If* $[a_{r,i}] = 0$ *for all* $i \in [0, k]$*, then it is* inactive *and keeps its state and does not read, write, or move on any tape.*

The use of routers inbetween ITM's makes sure that an ITM activates another ITM (indirectly through the router) if and only if it first sends it a message. The message may of course be empty to simply pass activation. Note that the address of a message is appended to the *end* of the message. This may seem odd, but it turns out to be useful for technical reasons (see Appendix A.4 for details).

Due to the test in step 2, a message can only be copied from the 0th receive tape to the ith send tape for $i > 0$, or from the ith receive tape for $i > 0$ to the 0th send tape. Furthermore, data written to or read from the 0th tape contains the index of another pair of tapes as an n-bit appendix, whereas it does not for other tapes. Thus, data written to the 0th write-once tape may be badly formed in which case the data is simply written back to the 0th write-once tape with the prefix $\Diamond$. This prefix is a special symbol used only for this purpose that indicates badly formed inputs.

Remark 1 (Concatenation). Concatenations such as that in Step 2 are common in this chapter and the chapters that follows. Care has to be taken to avoid that such concatenation, directly or indirectly, give rise to strings that can not be decoded uniquely into the original components. We can not solve this by simply stating that concatenation is a short hand for an invertible encoding algorithm, since we need the associative property of concatenation to prove that routers and communication models "commute". Fortunately, it is easy to see that there is no risk of ambiguous representations for most uses of concatenation.

To connect routers and ITM's with each other we let them share tapes pairwise. We formalize this as follows.

Definition 3 (Slot of Interactive Turing Machine or Router). *A tuple of heads of an ITM* (s, r, a_s, a_r) *or a tuple of heads of a router* $(s_i, r_i, a_{s,i}, a_{r,i})$ *is a* slot. *(Using notation from Definitions 1 and 2.)*

Definition 4 (Linked). *Two slots* (s, r, a_s, a_r) *and* (s', r', a'_s, a'_r) *are* linked *if there are four tapes such that the heads of each pair* (s, r'), (s', r), (a_s, a'_r), *and* (a'_s, a_r) *point to the same tape and no other heads point to any of these tapes.*

An ITM graph is simply a number of ITM's that are linked to each other indirectly using routers. Note that a router of which the 0th slot is linked to an ITM effectively increases the number of slots of the ITM. From now on we take this view. A basic requirement of an ITM graph to be executable, is that no ITM has any "dangling" tape heads.

Definition 5 (ITM Graph). *An* ITM graph *is a set* V *of ITM's, a set* R *of routers, and a set of additional tapes such that the slot of each ITM is linked to the* 0*th slot of a router, the* 0*th slot of each router is linked to the slot of an ITM, and every other slot of every router in* R *is linked to a slot of a different router in* R. *The set of all ITM graphs is denoted* $\mathsf{G}_{\mathsf{ITM}}$.

In other words, we use the routers to increase the number of slots of ITM's and then link the slots of routers to each other to allow the ITM's to communicate. Figure 2 illustrates this. The idea behind this approach is to restrict the notion of an ITM to Turing machines that have a fixed number of tapes. This avoids the need to change the computational model by adding tapes for parties in a protocol depending on how many parties there are.

Definition 6 (Initializing an ITM Graph). *To* initialize *an ITM graph with ITM's* $\mathcal{M}_1, \ldots, \mathcal{M}_k$, *the identity tape of* $\mathcal{M}_j$ *is assigned the integer* j *in binary, every cell of every activity tape is set to zero, every cell of every random tape is set to a randomly chosen bit, every cell of every other tape is set to* $\perp$, *and tape heads pointing to the same tape are set to point to the same cell.*

We say that a tape of an initialized ITM graph is assigned a string x when we fill the consecutive cells starting at the cell pointed to by the tape heads with x. This is done in the reachable direction for directed tape heads and in some canonical direction for other tape heads.

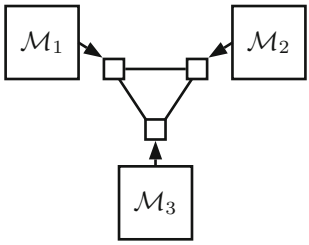

Fig. 2. An ITM graph consisting of parties $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ linked by three unnamed routers providing three slots each. The 0th slot of each router is marked by an arrow. We use this convention throughout this paper.

To simplify the analysis of running times, we ignore the state transitions occuring in routers when stating running times. This does not change any results about concrete protocols in any essential way, since only a small constant number of routers are used and they all run in linear time in the messages forwarded.

Definition 7 (Executing an ITM Graph). *An ITM graph with ITM's $\mathcal{M}_1, \ldots, \mathcal{M}_k$, that has been initialized, is* executed *starting at $\mathcal{M}_1$ on security parameter n and input z to $\mathcal{M}_1$ as follows.*

1. *Assign 1^n to the security parameter tape of $\mathcal{M}_j$ for $j \in [k]$.*
2. *Set the input tape of $\mathcal{M}_1$ to z.*
3. *Set $[a_r] = 1$, where a_r is the receiving activity head of $\mathcal{M}_1$.*
4. *Repeatedly execute the transition functions of all ITM's in unison.*

Note that due to the demand that an ITM or a router is active to change its state, or read, write, or move on a tape, this effectively means that a single machine is executing at any time.

Definition 8 (Bounding the Running Time). *Let G be an ITM graph and let X be a subset of the ITM's in G. We say that the* running time of G is bounded at X by T_X *if the number of active state transitions taking place in ITM's in X is bounded by T_X.*

The above gives a solid foundation for defining a simple and explicit version of the UC framework, but the notation is cumbersome. From now on we say that two ITM's are *linked* if two or more slots of their routers are linked. This allows us to take an abstract view of an ITM graph as a set of ITM's V and a set of links E describing how the ITM's are connected. If two machines are linked, then they can exchange messages and activate each other.

However, an ITM with a set of slots not only expects to be linked to some other ITM's, it expects that particular slots are used to form links to particular slots of other ITM's. Thus, we must label the slots of each ITM and introduce notation for forming a link using two such slots. Suppose that the ITM's $\mathcal{M}_1$ and $\mathcal{M}_2$ have slots $[a]$ and $[b]$ respectively. Then $\langle \mathcal{M}_1[a], \mathcal{M}_2[b] \rangle$ denotes a link formed between slot $[a]$ of $\mathcal{M}_1$ and slot $[b]$ of $\mathcal{M}_2$. Due to the restrictions on ITM's,

the definition of a router, and the starting state of an initialized ITM graph, this guarantees that exactly one ITM is active at any given time. In figures, we now draw the machines as circles instead of squares to indicate that we have abstracted from the details of communication.

Throughout we use the convention that a small letter in a slot, e.g., a in $[a]$, is a variable over the set of all labels of slots, and a capital letter is the label given verbatim, e.g., $\mathcal{M}$ in $[\mathcal{M}]$.

4 Entities of Models

Before we introduce the real, ideal, and hybrid models, we introduce the ITM's used to form these models. To be able to talk about different types of ITM's below without ambiguity we *mark* them. This can be formalized by adding an additional read-only tape on which the marking is written when the ITM is initialized, but we avoid formalizing this to avoid cluttering. Furthermore, each ITM of a given type has dedicated named slots.

An implementation of a function in software typically checks that the input is of a given form and returns an error code or throws an exception otherwise. It is then the responsibility of the caller of the function to deal with the error or exception. We mirror this in that if an ITM receives a message w on a slot $[a]$ that does not match the explicitly stated format of valid messages, then $\Diamond \| w$ is written to $[a]$. We have already used this convention in Definition 2.

A communication model captures how the parties of a protocol can communicate in the presence of an adversary.

Definition 9 (Communication Model). *A* k-communication model $\mathcal{C}$ *is an ITM marked as a "communication model" with one ideal functionality slot* $[\mathcal{F}]$*, party slots* $[\mathcal{P}_1], \ldots, [\mathcal{P}_k]$*, and an adversary slot* $[\mathcal{A}]$*. If* $\Diamond \| w$ *is read from* $[\mathcal{P}_i]$ *or* $[\mathcal{F}]$*, then* $\Diamond \| w$ *is written to* $[\mathcal{A}]$*.*

The adversary slot is used by an adversary to influence the behaviour of the communication model, e.g., if the communication model represents the Internet, then the adversary can insert, delay, or remove messages. The party slots are used by parties to communicate through the communication model. The ideal functionality slot is used to communicate with an ideal functionality. Note that the above definition implies that whenever a party or an ideal functionality refuses to accept an input, then the adversary is informed about this incident and activated. When no ideal functionality is needed we tacitly assume that an ideal functionality that refuses any input is used.

Definition 10 (Ideal Functionality). *An* ideal functionality $\mathcal{F}$ *is an ITM marked as an "ideal functionality" with a single communication slot* $[\mathcal{C}]$*.*

The communication slot is used by the ideal functionality both to accept inputs and to return outputs.

Definition 11 (Party). *An* f-party $\mathcal{P}$ *is an ITM marked "party" with an environment slot* $[\mathcal{Z}]$*, a communication slot* $[\mathcal{C}]$*,* f *subparty slots* $[\mathcal{U}_1], \ldots, [\mathcal{U}_f]$*, and an adversary slot* $[\mathcal{A}]$*. When* $f = 0$ *we simply say that* $\mathcal{P}$ *is a* party*.*

The subprotocol slots are used in the hybrid model to formalize access to subprotocols and ideal functionalities. The adversary slot is only used by corrupted parties. If it is not used in the formation of a model, then we assume that it is simply linked to an ITM that does not accept any input.

Definition 12 (Protocol). *A* (k, f)-protocol π *is a list* $(\mathcal{P}_1, \ldots, \mathcal{P}_k)$ *of* f*-parties. When* $f = 0$ *we simply say that* π *is a* k-protocol *(or* protocol *when* k *is clear from the context).*

Definition 13 (Adversary). *A* (k, f)-adversary $\mathcal{A}$ *is an ITM marked as an "adversary" with a communication slot* $[\mathcal{C}]$*, an environment slot* $[\mathcal{Z}]$*,* f *subadversary slots* $[\mathcal{A}_1], \ldots, [\mathcal{A}_f]$*, and* k *corrupted party slots* $[\mathcal{P}_1^*], \ldots, [\mathcal{P}_k^*]$*. When* $f = 0$ *we simply say that* $\mathcal{A}$ *is a* k-adversary*.*

The corrupted party slots are used to communicate with corrupted parties in protocols. Depending on which parties, and how many parties, are corrupted some of these slots may remain unused. To meet the requirement that a model is an ITM graph we assume that each such slot is linked to an ITM that does not accept any input. Figures 3 and 4 illustrate a communication model, an ideal functionality, a party, an adversary, and a corrupt party.

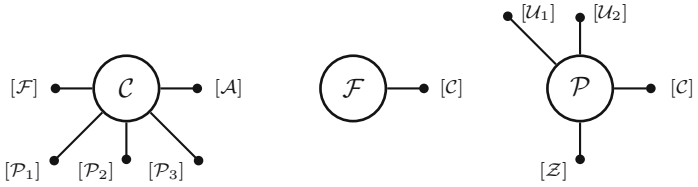

Fig. 3. To the left a 3-communication model $\mathcal{C}$ with ideal functionality slot $[\mathcal{F}]$, adversary slot $[\mathcal{A}]$, and party slots $[\mathcal{P}_1]$, $[\mathcal{P}_2]$, and $[\mathcal{P}_3]$. In the middle an ideal functionality $\mathcal{F}$ with a single communication slot $[\mathcal{C}]$. To the right a 2-party with subparty slots $[\mathcal{U}_1]$ and $[\mathcal{U}_2]$, communication slot $[\mathcal{C}]$, and environment slot $[\mathcal{Z}]$.

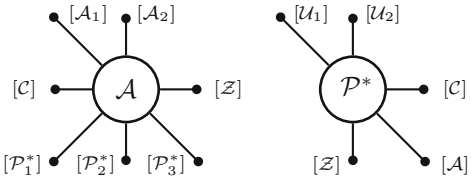

Fig. 4. To the left a $(3, 2)$-adversary with a communication slot $[\mathcal{C}]$, subadversary slots $[\mathcal{A}_1]$ and $[\mathcal{A}_2]$, an environment slot $[\mathcal{Z}]$, and corrupted party slots $[\mathcal{P}_1^*]$, $[\mathcal{P}_2^*]$, and $[\mathcal{P}_3^*]$. To the right a corrupted 2-party $\mathcal{P}^*$ with a communication slot $[\mathcal{C}]$, subparty slots $[\mathcal{U}_1]$ and $[\mathcal{U}_2]$, an adversary slot $[\mathcal{A}]$, and an environment slot $[\mathcal{Z}]$.

5 Real Free Models

The real communication model formalizes a network in which the adversary can read, delete, modify, and insert any message of its choice. The Internet is an example of such a network.

Definition 14 (Real Communication Model). *The* real k-communication model $\mathcal{N}_k$ *is defined as follows.*

- *If* w *is read from* $[\mathcal{P}_i]$*, where* $i \in [k]$*, then* $\mathcal{P}_i \| w$ *is written to* $[\mathcal{A}]$.
- *If* $\mathcal{P}_i \| w$ *is read from* $[\mathcal{A}]$*, where* $i \in [k]$*, then* w *is written to* $[\mathcal{P}_i]$.

A real free model describes a protocol that executes over a real communication model. We define a map that combines a communication model, parties, and an adversary into a graph of linked ITM's. Recall that $\langle \mathcal{M}_0[a], \mathcal{M}_1[b] \rangle$ denotes a link between slot $[a]$ of $\mathcal{M}_0$ and slot $[b]$ of $\mathcal{M}_1$.

Definition 15 (Real Model Map). *The* real (k, I, f)-model map *is the map* $\mathscr{R}_{k,I,f} : (\pi, \mathcal{A}, \pi^*) \mapsto (V, E)$*, where* $\pi = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$ *is a* (k, f)*-protocol,* $\mathcal{A}$ *is an* f*-adversary, and* $\pi^* = \{\mathcal{P}_i^*\}_{i \in I}$ *is a set of corrupted* f*-parties, defined by*

$$
\begin{aligned}
V =& \{\mathcal{N}_k, \mathcal{A}\} \cup \bigcup\nolimits_{i \notin I} \{\mathcal{P}_i\} \cup \bigcup\nolimits_{i \in I} \{\mathcal{P}_i^*\} \quad \text{and} \\
E =& \left\{ \langle \mathcal{A}[\mathcal{C}], \mathcal{N}_k[\mathcal{A}] \rangle \right\} \cup \bigcup\nolimits_{i \notin I} \left\{ \langle \mathcal{P}_i[\mathcal{C}], \mathcal{N}_k[\mathcal{P}_i] \rangle \right\} \\
& \cup \bigcup\nolimits_{i \in I} \left\{ \langle \mathcal{P}_i^*[\mathcal{C}], \mathcal{N}_k[\mathcal{P}_i] \rangle, \langle \mathcal{P}_i^*[\mathcal{A}], \mathcal{A}[\mathcal{P}_i] \rangle \right\}.
\end{aligned}
$$

Definition 16 (Real Free Model). *A* real free (k, I, f)-model M *is an output of the real free* (k, I, f)*-model map. If* $f = 0$*, then we simply say that* M *is a real free* (k, I)*-model.*

We say that the real model is *free*, since the parties and the adversary in it have free environment slots (and possibly free subparty or subadversary slots), i.e., a real free model is not an ITM graph and can not be executed. Figures 5 and 6 illustrate real free models without and with corruption.

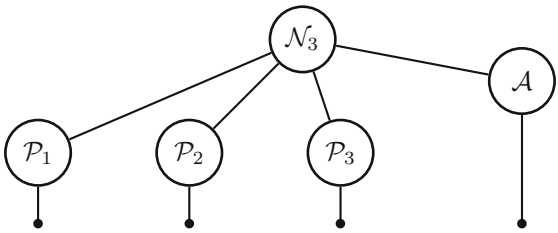

Fig. 5. A real free $(3, \emptyset)$-model $\mathscr{R}_{3,\emptyset,0}(\pi, \mathcal{A}, \emptyset)$ with a real 3-communication model $\mathcal{N}_3$, 3-protocol $\pi = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$, and real 3-adversary $\mathcal{A}$.

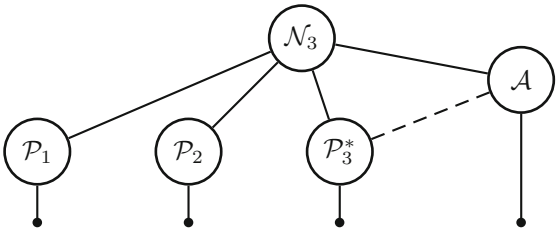

Fig. 6. The real $(3, I)$-model $\mathscr{R}_{3,I,0}(\pi, \mathcal{A}, \pi^*)$ with indices of corrupted parties $I = \{3\}$, real 3-communication model $\mathcal{N}_3$, 3-protocol $\pi = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$, real 3-adversary $\mathcal{A}$, and set of corrupted parties $\pi^* = \{\mathcal{P}_3^*\}$. Note the link between $\mathcal{A}$ and the corrupted party $\mathcal{P}_3^*$.

6 Ideal Free Models

The ideal model formalizes a protocol execution in an ideal world where there is an ideal functionality, i.e., a trusted party that performs some service. The trusted party is simply an ITM executing a program, and it communicates with the parties through the ideal communication model.

The ideal communication model below captures the fact that the adversary may decide if and when it would like to deliver a message from the ideal functionality to a party, but it cannot read the contents of the communication between parties and the ideal functionality.

Definition 17 (Ideal Communication Model). *The* ideal k-communication model $\mathcal{I}_k$ *is defined as follows.*

- *If* $\mathcal{F}\|m$ *is read from* $[\mathcal{A}]$*, then* $\mathcal{S}\|m$ *is written to* $[\mathcal{F}]$*.*
- *If* $\mathcal{S}\|m$ *is read from* $[\mathcal{F}]$*, then* $\mathcal{F}\|m$ *is written to* $[\mathcal{A}]$*.*
- *If* w *is read from* $[\mathcal{P}_i]$*, then* $\mathcal{P}_i\|w$ *is written to* $[\mathcal{F}]$*.*
- *If* $w\|(\mathcal{P}_j, w_j)_{j\in J}\|e$ *is read from* $[\mathcal{F}]$*, where* $J \subset [k]$*, then for* $j \in J$*:*
  1. τ_j *is chosen randomly, and*
  2. $(\mathcal{P}_j, w_j\|e)$ *is stored in a database under* τ_j*.*
  Then $w\|(\mathcal{P}_j, \tau_j)_{j\in J}\|e$ *is written to* $[\mathcal{A}]$*.*
- *If* τ *is read from* $[\mathcal{A}]$ *and* $(\mathcal{P}_j, w\|e)$ *is stored under* τ *in the database, then* $w\|e$ *is written to* $[\mathcal{P}_j]$*.*

In our thesis we use an authenticated bulletin board for communication. Authenticated channels are trivial to define using an ideal functionality. Although we could absorb this into a separate communication model, this makes little sense.

Definition 18 (Authenticated Channels Functionality). *The* authenticated channels functionality $\mathcal{F}_{auth}$ *repeatedly reads an input of the form* $\mathcal{P}_i\|(\mathcal{P}_j, m)$ *from* $[\mathcal{C}]$ *and writes* $(\mathcal{P}_j, \mathcal{P}_i\|m)\|(\mathcal{P}_j, \mathcal{P}_i\|m)$ *to* $[\mathcal{C}]$*.*

In most formalizations the lengths of messages are provided to the simulation adversary by the communication model. This is needed to prove the security of most protocols, since without it the ideal functionality could hide the lengths of messages from the simulation adversary (something that would be impossible to achieve in a real protocol). Our formalization requires the definition of each ideal functionality to provide the lengths explicitly. However, for concrete protocols this is rarely needed, since the lengths of messages can be derived by the simulation adversary from the security parameter.

Definition 19 (Dummy Party). *A* dummy party *is a party that writes any input on* $[\mathcal{Z}]$ *to* $[\mathcal{C}]$*, and writes any input on* $[\mathcal{C}]$ *to* $[\mathcal{Z}]$*.*

Dummy parties are introduced to provide identical interfaces to the parties in real models and to ideal functionalities. There may be many copies of the dummy party. Dummy parties are denoted by $\mathcal{Q}_i$ to distinguish them from real parties and may be thought of as labels for links. We denote a dummy k-protocol by $(\mathcal{Q}_1, \ldots, \mathcal{Q}_k)$.

The ideal free model below captures the setup one wishes to realize, i.e., the environment may interact with the ideal functionality $\mathcal{F}$, except that the adversary $\mathcal{S}$ has some control over how the communication model behaves.

Definition 20 (Ideal Free Model Map). *The* ideal free (k, I)-model map *is the map* $\mathscr{I}_{k,I} : (\mathcal{F}, \mathcal{S}, \sigma^*) \mapsto (V, E)$*, where* $I \subset [k]$ *is a set of indices of corrupted parties,* $\mathcal{F}$ *is an ideal functionality,* $\mathcal{S}$ *is a simulation* k*-adversary, and* $\sigma^* = \{\mathcal{Q}_i^*\}_{i \in I}$ *is a set of corrupted parties, defined by*

$$
\begin{aligned}
V =& \{\mathcal{I}_k, \mathcal{F}, \mathcal{S}\} \cup \bigcup\nolimits_{i \notin I} \{\mathcal{Q}_i\} \cup \bigcup\nolimits_{i \in I} \{\mathcal{Q}_i^*\}, \quad \textit{and} \\
E =& \big\{\langle \mathcal{I}_k[\mathcal{F}], \mathcal{F}[\mathcal{C}]\rangle, \langle \mathcal{S}[\mathcal{C}], \mathcal{I}_k[\mathcal{A}]\rangle\big\} \cup \bigcup\nolimits_{i \notin I} \big\{\langle \mathcal{Q}_i[\mathcal{C}], \mathcal{I}_k[\mathcal{P}_i]\rangle\big\} \\
& \cup \bigcup\nolimits_{i \in I} \big\{\langle \mathcal{Q}_i^*[\mathcal{C}], \mathcal{I}_k[\mathcal{P}_i]\rangle, \langle \mathcal{Q}_i^*[\mathcal{A}], \mathcal{S}[\mathcal{P}_i]\rangle\big\}.
\end{aligned}
$$

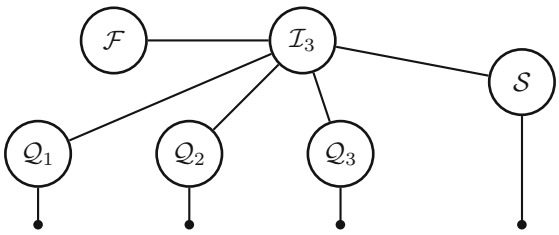

Fig. 7. An ideal free $(3, \emptyset)$-model $\mathscr{I}_{3,\emptyset}(\mathcal{F}, \mathcal{S}, \emptyset)$ with ideal 3-communication model $\mathcal{I}_3$, dummy 3-protocol $(\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3)$, ideal functionality $\mathcal{F}$, and simulation 3-adversary $\mathcal{S}$.

Definition 21 (Ideal Free Model). *An* ideal free (k, I)-model *is an output of the ideal free* (k, I)*-model map.*

Figure 7 illustrate an ideal free model without corruption.

7 Hybrid Free Models

A hybrid free model formalizes the execution of a real protocol that has access to other real subprotocols, ideal functionalities, or hybrid protocols. It can both be used to describe protocols that need setup assumptions (or trusted parties) for specific tasks and as a tool to construct protocols in a modular way.

Note that the following definitions give a joint inductive definition of the hybrid free model map and hybrid free models.

Definition 22 (Hybrid Free Model). *A* hybrid free (k, I, f)-model *is an output of the hybrid free k-model map $\mathscr{H}_{k,I,f}$ of Definition 26 below. We drop f from our notation if it is zero.*

Definition 23 (Free Model). *A* free (k, I, f)-model *is a real free (k, I, f)-model, a hybrid free (k, I, f)-model, or provided $f = 0$, an ideal free (k, I)-model.*

A free model is complete if it does not have any dangling subparty slots. Thus, every free ideal model and every real/hybrid (k, I)-model is complete.

Definition 24 (Complete Free Model). *A free $(k, I, 0)$-model is* complete.

Definition 25 (Root of Free Model). *The* root *of a free (k, I, f)-model (V, E) is the unique pair of a protocol and adversary $((\mathcal{X}_1, \ldots, \mathcal{X}_k), \mathcal{A})$ such that $\mathcal{X}_i \in V$ is a party with a free slot $[\mathcal{Z}]$ for $i \in [k]$ and $\mathcal{A} \in V$ is an adversary with a free slot $[\mathcal{Z}]$.*

We stress that if $i \in I$, then $\mathcal{X}_i$ is a corrupted party usually denoted $\mathcal{P}_i^*$ (or $\mathcal{Q}_i^*$), and otherwise it is an uncorrupted party $\mathcal{P}_i$ (or $\mathcal{Q}_i$) defined by the original protocol or dummy protocol of the ideal functionality.

Definition 26 (Hybrid Free Model Map). *The* hybrid free (k, I, f)-model map *is the map $\mathscr{H}_{k,I,f}$ with $f > 0$ that takes as input:*

- *A real free (k, I, f)-model (V, E) with root $((\mathcal{X}_1, \ldots, \mathcal{X}_k), \mathcal{A})$.*
- *A complete free (k, I)-model (V_j, E_j) with root $((\mathcal{X}_{j,1}, \ldots, \mathcal{X}_{j,k}), \mathcal{A}_j)$ for $j \in [f]$.*

and outputs a complete free model (V', E') where

$$V' = V \cup \bigcup\nolimits_{j \in [f]} V_j \quad \textit{and}$$

$$E' = E \cup \bigcup\nolimits_{j \in [f]} \left(E_j \cup \left\{ \langle \mathcal{A}[\mathcal{A}_j], \mathcal{A}_j[\mathcal{Z}] \rangle \right\} \cup \bigcup\nolimits_{i \in [k]} \left\{ \langle \mathcal{X}_i[\mathcal{U}_j], \mathcal{X}_{j,i}[\mathcal{Z}] \rangle \right\} \right).$$

8 Environments and Models

To be able to execute a free model we need an *environment* that connects to the free slots of the root protocol and root adversary. We formalize the environment in which a protocol is executed as an ITM (Fig. 8).

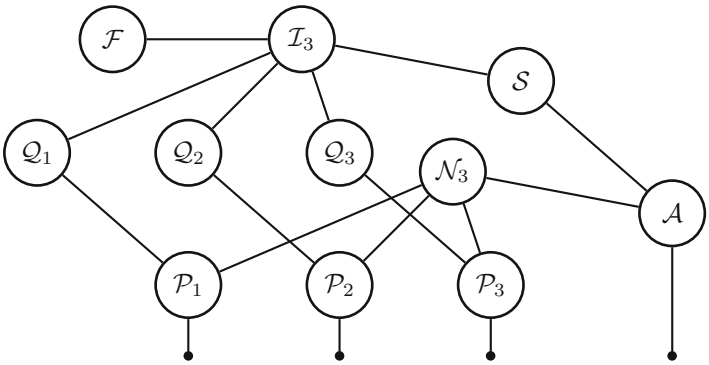

Fig. 8. A hybrid free model $\mathscr{H}_{3,I,1}\big(\mathscr{R}_{3,I,1}(\pi, \mathcal{A}, \emptyset), \mathscr{I}_{3,I}(\mathcal{F}, \mathcal{S}, \emptyset)\big)$ with indices of corrupted parties $I = \emptyset$, real 3-communication model $\mathcal{N}_3$, root $(3,1)$-protocol $\pi = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$, root $(3,1)$-adversary $\mathcal{A}$, ideal 3-communication model $\mathcal{I}_3$, dummy 3-protocol $(\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3)$, ideal functionality $\mathcal{F}$, and simulation 3-subadversary $\mathcal{S}$.

Definition 27 (Environment). *A* k-environment *is an ITM marked as an "environment" with party slots* $[\mathcal{P}_1], \ldots, [\mathcal{P}_k]$ *and an adversary slot* $[\mathcal{A}]$.

Figure 9 illustrates an environment. The environment provides the data used by the parties in the protocol and is always the first ITM to be activated during the execution of the model.

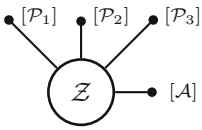

Fig. 9. A k-environment with party slots $[\mathcal{P}_1]$, $[\mathcal{P}_2]$, and $[\mathcal{P}_3]$, and an adversary slot $[\mathcal{A}]$.

Definition 28 (Environment Map). *The* (k, I)-environment map $\mathscr{Z}_k : (M, \mathcal{Z}) \mapsto (V', E')$ *takes a complete free* (k, I)*-model* $M = (V, E)$ *with root* $\big((\mathcal{X}_1, \ldots, \mathcal{X}_k), \mathcal{A}\big)$ *and a* k*-environment* $\mathcal{Z}$ *as input and outputs* (V', E') *where*

$$V' = V \cup \{\mathcal{Z}\} \quad \textit{and}$$
$$E' = E \cup \big\{\langle \mathcal{Z}[\mathcal{A}], \mathcal{A}[\mathcal{Z}]\rangle\big\} \cup \bigcup\nolimits_{i \in [k]} \big\{\langle \mathcal{Z}[\mathcal{P}_i], \mathcal{X}_i[\mathcal{Z}]\rangle\big\}.$$

Definition 29 (Model). *A* (k, I)-model *is an output of the* (k, I)*-environment map.*

Note that a model is an ITM graph, which means that it can be executed. In an execution of a model the environment is always activated first with some auxiliary input. Figures 10, 11, and 12 illustrate a real model, an ideal model, and a hybrid model respectively. We abuse notation and write $\mathscr{R}_{k,I,f}(\pi, \mathcal{A}, \pi^*, \mathcal{Z})$ instead of $\mathscr{Z}_k(\mathscr{R}_{k,I,f}(\pi, \mathcal{A}, \pi^*), \mathcal{Z})$ and correspondingly for ideal and hybrid free model maps.

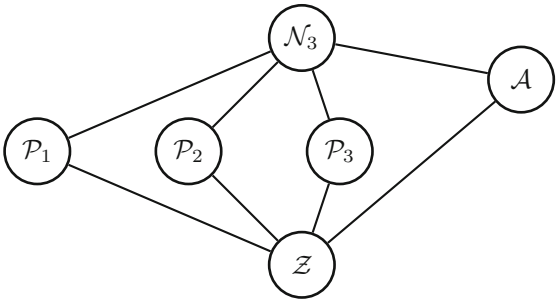

Fig. 10. A real $(3, \emptyset)$-model $\mathscr{R}_{3,\emptyset,0}(\pi, \mathcal{A}, \emptyset, \mathcal{Z})$ with real 3-communication model $\mathcal{N}_3$, 3-protocol $\pi = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$, real 3-adversary $\mathcal{A}$, and 3-environment $\mathcal{Z}$.

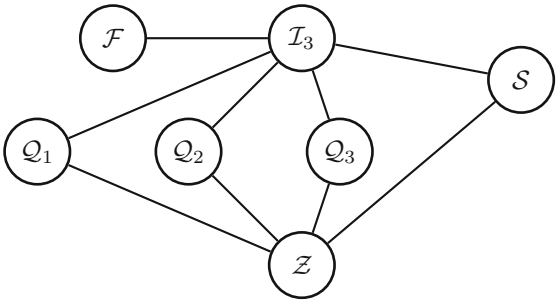

Fig. 11. An ideal $(3, \emptyset)$-model $\mathscr{I}_{3,\emptyset}(\mathcal{F}, \mathcal{S}, \emptyset, \mathcal{Z})$ with ideal 3-communication model $\mathcal{I}_3$, dummy 3-protocol $(\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3)$, ideal functionality $\mathcal{F}$, simulation 3-adversary $\mathcal{S}$, and 3-environment $\mathcal{Z}$.

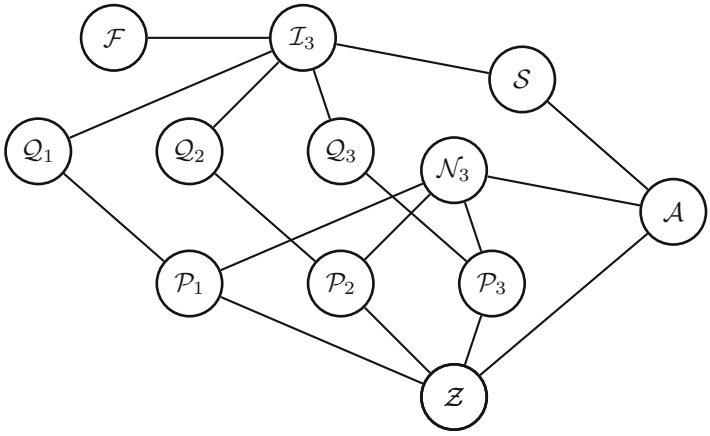

Fig. 12. A hybrid model $\mathscr{H}_{3,I,1}\big(\mathscr{R}_{3,I,1}(\pi, \mathcal{A}, \emptyset), \mathscr{I}_{3,I}(\mathcal{F}, \mathcal{S}, \emptyset), \mathcal{Z}\big)$ with real 3-communication model $\mathcal{N}_3$, root $(3,1)$-protocol $\pi = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$, root $(3,1)$-adversary $\mathcal{A}$, ideal 3-communication model $\mathcal{I}_3$, dummy 3-protocol $(\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3)$, ideal functionality $\mathcal{F}$, simulation 3-subadversary $\mathcal{S}$, and 3-environment $\mathcal{Z}$.

9 Classes of Adversaries

We need to bound the running times of the adversary, the simulation adversary, and the environment to give a definition of security. Several ways to do this have been proposed in the literature. We choose a simple solution that gives concrete bounds on the security reductions. Given a model $M = (V, E)$ with an adversary $\mathcal{H}$ (real, ideal, or hybrid) and environment $\mathcal{Z}$ we say that:

1. $\mathcal{H}$ has running time $T_{\mathcal{H}}$ if the running time of M is bounded by $T_{\mathcal{H}}$ at $V \setminus \{\mathcal{Z}\}$.
2. $\mathcal{Z}$ has running time $T_{\mathcal{Z}}$ if the running time of M is bounded by $T_{\mathcal{Z}}$ at $\{\mathcal{Z}\}$.

We remark that this approach differs from the simpler approach used in our thesis [12] and in [4], where the running time of each ITM was simply bounded by a polynomial in the security parameter. The advantage with the current approach is that ideal functionalities and protocols never halt until they are explicitly asked to by the adversary or the environment. However, both approaches are possible in our formalization.

10 Simplified Notation

At this point we have defined the models of the simplified UC framework rigorously, but it is convenient to introduce some alternative notation more in line with the literature to emphasize protocols, ideal functionalities, and adversaries instead of the technical details of how these are linked. We stress that we do not abandon the original notation; the freedom to change notation when convenient greatly simplifies describing and analyzing protocols.

It is easy to see that we may assume that all corrupted parties and all adversaries except the one linked to the environment are simulations of the router of Definition 2 with a suitable number of heads. This is illustrated in Fig. 13.

The subprotocols and ideal functionalities of a hybrid model are arranged in a tree of subprotocols where every ideal functionality is a leaf. Thus, given the set of indices of corrupted parties and the tree of subprotocols and ideal functionalities, an adversary, and an environment we can introduce an indexing scheme and recover the hybrid model. We denote a tree of subprotocols and ideal functionalities by inductively applying the rules that:

1. An ideal free model based on an ideal functionality $\mathcal{F}$ is denoted by $\mathcal{F}$.
2. A real free model based on a protocol π is denoted by π.
3. A hybrid free model based on a real protocol π, and complete free models based on hybrid protocols $\rho_1, \ldots, \rho_t$ is denoted $\pi(\rho_1, \ldots, \rho_t)$.

We may consider the set of indices of corrupted parties to be embedded in the description of the adversary and simply say that we consider an adversary that corrupts a certain set of parties. This convention gives less concrete notation than the original, but it is more in line with the literature.

Suppose that ρ is such a description of a protocol, $\mathcal{Z}$ is an environment, and $\mathcal{A}$ is an adversary (where the indices of corrupted parties have been encoded).

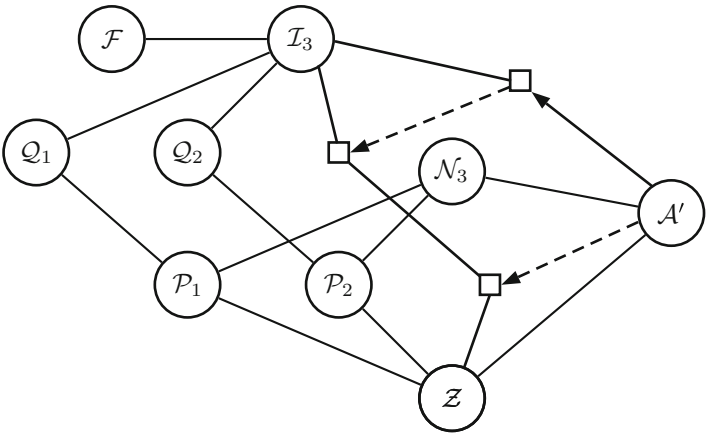

Fig. 13. A modification of a hybrid free model with corruption, where $\mathcal{Q}_3^*$, $\mathcal{P}_3^*$, and $\mathcal{S}$ are replaced by routers and $\mathcal{A}'$ is a corresponding modification of $\mathcal{A}$, but with an environment $\mathcal{Z}$ turning it into a model. The 0th slot of each router is marked by an arrow. We stress that strictly speaking each router is simulated by an ITM to adhere to our definitions. The routers needed for this ITM to have multiple links are hidden by our abstractions.

Then we denote by $\mathcal{Z}_z(\rho, \mathcal{A})$ the output of the environment $\mathcal{Z}$ running on auxiliary input z when executing the model recovered from ρ, $\mathcal{Z}$, and $\mathcal{A}$. Sometimes we structure the adversary to match the topology of the protocols and ideal functionalities, i.e., we denote each simulation subadversary by $\mathcal{S}$ and each hybrid or real subadversary by $\mathcal{A}$ with suitable subscripts.

We remark that in hybrid models the number of dummy parties linked to any ideal functionalities that are used is easily derived. Thus, there is no need to state this explicitly. This is not the case for ideal models, but the number of parties is always clear from the context.

Example 1. Suppose that π is a protocol that uses real subprotocols π_0 and π_1, and an ideal functionality $\mathcal{F}$, where π_1 in turn uses an ideal functionality $\mathcal{F}_1$. Suppose further that $\mathcal{A}$ is the overall adversary that attacks π, and orchestrates: (1) subadversaries $\mathcal{S}$ and $\mathcal{S}_1$ of $\mathcal{F}$ and $\mathcal{F}_1$ respectively, and (2) real subadversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ of π_1 and π_2 respectively. Then the output of the corresponding model executed with auxiliary input z is denoted by $\mathcal{Z}_z\big(\pi(\pi_0, \pi_1(\mathcal{F}_1), \mathcal{F}), \mathcal{A}(\mathcal{A}_0, \mathcal{A}_1(\mathcal{S}_1), \mathcal{S})\big)$. If we are not interested in the internal structure of $\mathcal{A}$, then we simply write $\mathcal{A}$ instead of $\mathcal{A}(\mathcal{A}_0, \mathcal{A}_1(\mathcal{S}_1), \mathcal{S})$.

11 Definition of Security

Following the approach outlined at the beginning of the paper we now formalize the security of protocols. In this paper we only consider static corruption, i.e., an adversary may only choose a set of parties to corrupt before execution starts.

Remark 2. Adaptive corruption is easy to add to our framework as follows. (1) Add a link between each party and the adversary. There are already slots prepared for this. (2) Wrap each party in an ITM that simulates the party until it receives "corrupt" from the adversary, at which point it writes the state of the party to the adversary, and waits for a new ITM with a given state in return that it executes instead. The adversary may now use the link to the wrapped replacement freely. (3) Stipulate to which sets of parties the adversary may send "corrupt". Another wrapper of the adversary can be used to enforce this to avoid restrictions when quantifying over adversaries.

One would typically assume a uniform adversarial structure for subprotocols as for static corruption, but the approach works even when this is not the case.

Most proofs of security only hold as long as the adversary does not corrupt certain parties or some subsets of parties. An adversarial structure is a collection of sets, where each set is a set of indices of parties that the adversary can corrupt. We use J to denote an adversarial structure.

Example 2. If we have five parties $\mathcal{P}_1, \ldots, \mathcal{P}_5$ in a protocol and we are able to prove that the protocol is secure provided that at most one out of $\mathcal{P}_1$ and $\mathcal{P}_2$ is corrupted and two out of $\mathcal{P}_3$, $\mathcal{P}_4$, and $\mathcal{P}_5$ are corrupted. Then the adversarial structure we consider is $\mathsf{J} = \{\{1,3,4\},\{1,4,5\},\{1,3,5\},\{2,3,4\},\{2,4,5\},\{2,3,5\}\}$.

Here we only consider the case where corruption takes place in a uniform way in all free models within a model, i.e., if a party is corrupted, then so are all its subparties recursively. However, it is quite natural to generalize this in certain situations.

We use A to denote a class of adversaries with running time bounded by T_{A}, where the number of parties k and the topology of hybrid adversaries are implicit. Furthermore, the subset of such adversaries that corrupt the parties with indices in a set J are denoted by A_J. We use the same conventions for a class of simulation adversaries S and the corresponding class S_J of adversaries that corrupt dummy parties with indices in J. Finally, we use Z to denote a class of environments with running time bounded by T_{Z}. Given two classes A and A' of adversaries with the same topology, we simply write $\mathsf{A}+\mathsf{A}'$ to denote the class of adversaries with the same topology and running time $T_{\mathsf{A}} + T_{\mathsf{A}'}$.

For standard asymptotic security we can simply require that T_{A}, T_{S}, and T_{Z} are polynomially bounded, but for concrete security claims we can give explicit upper bounds.

Definition 30 (Secure Realization). *A protocol ρ is a* $(\mathsf{J},\mathsf{A},\mathsf{S},\mathsf{Z},\mu)$-secure realization *of a target protocol τ if for every $J \in \mathsf{J}$ and every adversary $\mathcal{A} \in \mathsf{A}_J$, there exists a simulation adversary $\mathcal{S} \in \mathsf{S}_J$ such that for every environment $\mathcal{Z} \in \mathsf{Z}$ and every auxiliary input $z \in \{0,1\}^*$:*

$$|\Pr[\mathcal{Z}_z(\rho,\mathcal{A}) = 1] - \Pr[\mathcal{Z}_z(\tau,\mathcal{S}) = 1]| \leq \mu.$$

The above definition is considerably more general than other flavours of the UC framework in that *a protocol can securely realize another protocol* and not

only an ideal functionality. This may seem contrived at first glance, but is in fact an important generalization that simplifies the description and analysis of concrete protocols.

Consider for example an ideal functionality for distributed key generation and decryption. It outputs a public key and can then be used to decrypt ciphertexts if asked to do so by the parties using its service. This works well with a CCA2-secure cryptosystem, but for IND-CPA secure cryptosystems the functionality can not be securely realized, since a simulator has no way of limiting access to the plaintexts needed to simulate decryption. Thus, any application of such a functionality must ensure that this information is otherwise available, but there are several ways to do this, e.g., a trusted party, secret sharing, and proofs of knowledge, and these are actually used in various electronic voting systems (see [13] for a discussion).

We can formalize an intuitive ideal functionality $\mathcal{F}$ for distributed key generation and decryption, and several different ideal functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_l$ for submitting a ciphertext as an input to the ideal functionality. The individual functionalities may be impossible to securely realize in isolation, but we can consider a hybrid protocol $\pi(\mathcal{F}, \mathcal{F}_i)$, where π forces any inputs to $\mathcal{F}$ to first be processed by $\mathcal{F}_i$ (possibly along with other information or through interaction) in such a way that $\mathcal{F}_i$, and hence the simulator, knows the plaintext of any ciphertexts decrypted by $\mathcal{F}$. This hybrid protocol can then be securely realized by a protocol of the form $\pi(\sigma, \sigma_i)$, where σ and σ_i are the natural and often classic implementations in practice. The hybrid protocol $\pi(\mathcal{F}, \mathcal{F}_i)$ may either be viewed as a type of ideal functionality that is secure by inspection, in which π should be a "thin" middle layer that is trivial to understand, or there could be another ideal functionality $\mathcal{F}'$ that it securely realizes. Thus, this approach avoids some of the artificial complexity of the UC framework and allows a more modular approach.

12 Universal Composition Theorem

Canetti [3] proves a powerful composition theorem. Loosely speaking it says that if a protocol π securely realizes some functionality $\mathcal{F}$, then the protocol π can be used instead of the ideal functionality regardless of how the functionality $\mathcal{F}$ is employed. The general composition theorem can handle polynomially many instances of a constant number of ideal functionalities for many different adversarial models, but we only need the following weaker special case due to the results in Appendix A.

Theorem 1 (Special Universal Composition Theorem). *If ρ_0 is a $(\mathsf{J}, \mathsf{A}, \mathsf{S}, \mathsf{Z}, \mu)$-secure realization of τ_0 and $\pi(\tau_0, \mathcal{F}_1)$ is a $(\mathsf{J}, \mathsf{A} + \mathsf{S}, \mathsf{S}', \mathsf{Z}, \mu)$-secure realization of τ, then $\pi(\rho_0, \mathcal{F}_1)$ is a $(\mathsf{J}, \mathsf{A}, \mathsf{S}', \mathsf{Z}, \mu + \mu')$-secure realization of τ.*

Proof. The triangle inequality implies that for every simulation adversary $\mathcal{S}_0$, every hybrid adversary $\mathcal{A}(\mathcal{A}_0, \mathcal{S}_1)$, every simulation adversary $\mathcal{S}$, every environment $\mathcal{Z}$ and every auxiliary input $z \in \{0,1\}^*$

$$\begin{aligned}
&\left|\Pr\left[\mathcal{Z}_z\big(\pi(\rho_0,\mathcal{F}_1),\mathcal{A}(\mathcal{A}_0,\mathcal{S}_1)\big)=1\right]-\Pr\left[\mathcal{Z}_z(\tau,\mathcal{S})=1\right]\right| \\
&\le\left|\Pr\left[\mathcal{Z}_z\big(\pi(\rho_0,\mathcal{F}_1),\mathcal{A}(\mathcal{A}_0,\mathcal{S}_1)\big)=1\right]-\Pr\left[\mathcal{Z}_z\big(\pi(\tau_0,\mathcal{F}_1),\mathcal{A}(\mathcal{S}_0,\mathcal{S}_1)\big)=1\right]\right| \\
&\quad+\left|\Pr\left[\mathcal{Z}_z\big(\pi(\tau_0,\mathcal{F}_1),\mathcal{A}(\mathcal{S}_0,\mathcal{S}_1)\big)=1\right]-\Pr\left[\mathcal{Z}_z(\tau,\mathcal{S})=1\right]\right| \qquad (1)
\end{aligned}$$

We now denote by $\mathcal{Z}_z(\mathcal{A},\mathcal{S}_1)$ the environment that simulates the environment $\mathcal{Z}$ on auxiliary input z, the real free model $\mathscr{R}_{k,J,2}(\pi,\mathcal{A},\pi^*)$, and the ideal free model $\mathscr{I}_{k,J}(\mathcal{F}_1,\mathcal{S}_1,\sigma_1^*)$. Here π^* and σ_1^* are the sets of corrupted subparties, but without loss of generality we may assume that they are routers. This allows us to rewrite the right side of Inequality (1) as

$$\begin{aligned}
&\left|\Pr\left[\mathcal{Z}_z(\mathcal{A},\mathcal{S}_1)(\rho_0,\mathcal{A}_0)=1\right]-\Pr\left[\mathcal{Z}_z(\mathcal{A},\mathcal{S}_1)(\tau_0,\mathcal{S}_0)=1\right]\right| \\
&+\left|\Pr\left[\mathcal{Z}_z\big(\pi(\tau_0,\mathcal{F}_1),\mathcal{A}(\mathcal{S}_0,\mathcal{S}_1)\big)=1\right]-\Pr\left[\mathcal{Z}_z(\tau,\mathcal{S})=1\right]\right|,
\end{aligned}$$

without restricting the quantification.

Note that if $\mathcal{A}(\mathcal{A}_0,\mathcal{S}_1)\in\mathsf{A}_J$ and $\mathcal{S}_0\in\mathsf{S}_J$, then $\mathcal{A}_0\in\mathsf{A}_J$ and $\mathcal{A}(\mathcal{S}_0,\mathcal{S}_1)\in\mathsf{A}_J+\mathsf{S}_J$. Morover, if $\mathcal{Z}(\mathcal{A},\mathcal{S}_1)\in\mathsf{Z}$, then $\mathcal{Z}\in\mathsf{Z}$. From the hypothesis of the theorem we know that for every hybrid adversary $\mathcal{A}(\mathcal{A}_0,\mathcal{S}_1)\in\mathsf{A}_J$ there exists a simulation adversary $\mathcal{S}_0\in\mathsf{S}_J$ such that for the hybrid adversary $\mathcal{A}(\mathcal{S}_0,\mathcal{S}_1)\in(\mathsf{A}_J+\mathsf{S}_J)$ there exists a simulation adversary $\mathcal{S}\in\mathsf{S}'_J$ such that for every environment $\mathcal{Z}_z(\mathcal{A},\mathcal{S}_1)\in\mathsf{Z}$ and every auxiliary input $z\in\{0,1\}^*$

$$\begin{aligned}
&\left|\Pr\left[\mathcal{Z}_z(\mathcal{A},\mathcal{S}_1)(\rho_0,\mathcal{A}_0)=1\right]-\Pr\left[\mathcal{Z}_z(\mathcal{A},\mathcal{S}_1)(\tau_0,\mathcal{S}_0)=1\right]\right|\le\mu \quad\text{and} \\
&\left|\Pr\left[\mathcal{Z}_z\big(\pi(\tau_0,\mathcal{F}_1),\mathcal{A}(\mathcal{S}_0,\mathcal{S}_1)\big)=1\right]-\Pr\left[\mathcal{Z}_z(\tau,\mathcal{S})=1\right]\right|\le\mu'.
\end{aligned}$$

We conclude that for every $\mathcal{A}(\mathcal{A}_0,\mathcal{S}_1)\in\mathsf{A}_J$ there exists a simulation adversary $\mathcal{S}\in\mathsf{S}'$ such that for every $\mathcal{Z}\in\mathsf{Z}$ and every auxiliary input $z\in\{0,1\}^*$

$$\left|\Pr\left[\mathcal{Z}_z\big(\pi(\rho_0,\mathcal{F}_1),\mathcal{A}(\mathcal{A}_0,\mathcal{S}_1)\big)=1\right]-\Pr\left[\mathcal{Z}_z(\tau,\mathcal{S})=1\right]\right|\le\mu+\mu'.$$

13 Transforms of Models

It is intuitively clear that we can absorb any real subprotocols into the main protocol by simply combining each real party and its subparties into single new real party, but this does not give a valid model according to our definitions, since each such party is linked to *multiple* real communication models. A similar problem appears when bundling multiple ideal communication models.

In Appendix A we describe and analyze three explicit faithful transforms that allow us to: (1) simulate multiple ITM's in a single ITM, (2) simulate multiple links between two ITM's using a single link, and (3) simulate multiple identical communication models using a single communication model. The first two are straightforward, but the third depends on the details of the definitions of the communication models. A transform is faithful if it is invertible and preserves functionality.

These transforms give us the freedom to view protocols with subprotocols and ideal functionalities in the most convenient way for each situation without

sacrificing rigor. In particular, it means that we can apply Theorem 1 to protocols with more than two ideal functionalities. More precisely, we can transform any protocol and adversary into a protocol of the form $\pi(\mathcal{F}_0, \mathcal{F}_1)$, as required by the composition theorem and a corresponding adversary $\mathcal{A}$. Suppose that π_0 securely realizes $\mathcal{F}_0$. Then, due to the composition theorem we know that there is a simulation adversary $\mathcal{S}$ which shows that $\pi(\pi_0, \mathcal{F}_1)$ securely realizes $\pi(\mathcal{F}_0, \mathcal{F}_1)$. Due to faithfulnesss, we may then recover the original protocol along with a modified simulation adversary $\mathcal{S}'$, which implies that the composition is secure for the original protocol. We provide details in Appendix A.

14 Relation to Other Security Frameworks

It is natural to ask if the simplified UC framework captures the same notion of security as other security frameworks. Instead of providing relations and proofs for particular other frameworks we exploit our transforms to make this easy for any security framework.

The faithful transforms allow us to turn any protocol into a protocol with at most one ideal functionality. If a protocol securely realizes an ideal functionality, then its transform does as well. Thus, proving that it securely realizes the functionality in another security framework is reduced to the special case where the protocol has at most one ideal functionality. More precisely, to relate the simplified UC framework to an alternative framework it suffices that: (1) protocols with at most one ideal functionality can be expressed in the alternative framework (with suitable restrictions), and (2) if there is an adversary that contradicts the security of such a protocol in the alternative framework, then there is an adversary that violates the security in the simplified UC framework.

In particular, relating the simplified UC framework to *any* reasonable presentation of the UC framework is straightforward. This should be contrasted with the analysis of Canetti et al. [4] which relates their presentation of the simplified UC framework with a *particular* presentation of the UC framework. Determining if their proof still holds after further modifications of the UC framework or for other alternative presentations is cumbersome.

A Transforms of Models

This section is dedicated to define and analyze the transforms informally described in the body of the paper. Although the definitions are somewhat technical in nature, the ideas and concepts are simple and illustrated in Figs. 14, 15, 16, 17, and 18. For all practical purposes, i.e., when analyzing concrete protocols, browsing these illustrations should be enough.

Throughout, we assume without loss of generality that if a Turing machine M_i simulates some other Turing machines for $i = 1, \ldots, m$ and M is said to simulate the M_i's, then during execution M instead simulates the machines simulated by each M_i directly. Thus, we may freely argue in terms of nested simulations without any computational penalty. To avoid cluttering we also assume that simulation of multiple Turing machines can be done without any overhead.

A.1 Faithful Transforms of ITM Graphs

We are interested in transforms of ITM graphs that preserve the functionality of the original, but we must also be able to invert each transform and recover the original ITM graph. Below we give rigorous definitions that captures these properties, but for all our transforms it is straightforward to see that this is the case.

Intuitively, the first component of an input to a transform is the ITM graph to be transformed and the second component parametrizes the transform, e.g., it may pinpoint particular ITMs to remove, move, or link in a specific way.

Definition 31 (Transform). *An* ITM graph transform *is a map* $\Phi : S \to \mathsf{G}_{\mathsf{ITM}}$*, where* $S \subset \mathsf{G}_{\mathsf{ITM}} \times \mathsf{ITM}^*$.

Given an ITM graph $G = (V, E)$ we may assume that we can list the ITMs in V in a canonical order. Thus, it is meaningful to view any inputs and random tapes of these ITMs as lists $m = (m_1, \ldots, m_{|V|})$ and $r = (r_1, \ldots, r_{|V|})$, respectively, and denote by $\mathcal{Z}_G(n, m, r)$ the output of $\mathcal{Z} \in V$ in an execution of G starting at $\mathcal{Z}$, using security parameter n, on inputs m and random tapes r.

We need to argue about the behaviour of both an ITM graph and its transform on the "same" random tapes, but the latter may have more or less ITMs. Thus, for every integers $a, b > 0$, we need a bijection $\epsilon : (\{0,1\}^*)^a \to (\{0,1\}^*)^b$ such that both ϵ and its inverse are efficiently computable. Such bijections are readily constructed, e.g., we can use interleaving of bits.

Definition 32 (Faithful Transforms). *An ITM graph transform* $\Phi : S \to \mathsf{G}_{\mathsf{ITM}}$ *is* faithful *if*

1. ***Preservation of functionality.*** *For every* $(G, C) \in S$*, where* $G = (V, E)$ *with communication models* U*, every* $\mathcal{Z} \in V$*, every security parameter* n*, every random tapes* $r \in (\{0,1\}^*)^{|V|-|U|}$ *to non-communication models, and every inputs* $m \in (\{0,1\}^*)^{|V|}$ *the transformed ITM graph* $\Phi(G, C)$ *computes the same function at* $\mathcal{Z}$ *with overwhelming probability, i.e.,*

$$\Pr\left[\mathcal{Z}_G(n, m, r) = \mathcal{Z}_{\Phi(G,C)}(n, m', r')\right] < 2^{-\mathsf{poly}(n)},$$

   where $m' = \epsilon(m)$*,* $r' = \epsilon(r)$*, and the probability is taken over the random tapes of the communication models.*
2. ***Invertibility.*** *There exists a transform* $\Theta : S \to \mathsf{G}_{\mathsf{ITM}}$ *that computes the original ITM graph, i.e., for every* $(G, C) \in S$ *we have* $\Theta(\Phi(G, C), C) = G$.

In the following we compose transforms and it is not possible in general to invert each step without access to the parameter C used in the transform, e.g., we may modify different parts of an ITM graph and it is impossible to know afterwords which part was modified first.

However, for composed transforms we may view the sequence of parameters used as a transcript and recover the previous ITM graph of each step due to the invertibility property. Thus, if $G' = \Phi(G, C)$, then without loss of generality we abuse notation and simply write $G = \Phi^{-1}(G')$ instead of $G = \Theta(G', C)$ and assume that the parameter C is available.

A.2 Simulating Multiple Interactive Turing Machines

The most obvious simplification of the description of an ITM graph is to let a single ITM simulate several other ITMs as well as their links. This is illustrated in Fig. 14 and defined below. Given subsets A and B of a set V of ITMs, we denote by $\mathsf{E}(A, B)$ the set of links between slots of ITMs in A and B respectively and set $\mathsf{E}(A) = \mathsf{E}(A, A)$.

Definition 33 (Simulation of ITMs). *Let $G = (V, E)$ be an ITM graph and let $A \subset V$. Denote by $S_{\mathcal{X}}$ the set of slots of $\mathcal{X} \in A$ that are not part of a link in $\mathsf{E}(A)$. Then $\Omega_{ITM}(A)$ denotes the ITM that simulates all ITMs in A with slots $\bigcup_{\mathcal{X}\in A}\bigcup_{[a]\in S_{\mathcal{X}}}\{[\mathcal{X}|a]\}$, where $[\mathcal{X}|a]$ is identified with the slot $[a]$ of $\mathcal{X}$ in the simulation for every $\mathcal{X} \in A$.*

Definition 34 (Simulation Transform). *Define the* simulation transform *$\Phi_{ITM}(G, A) = (V', E')$, where $G = (V, E)$ is an ITM graph and $A \subset V$, by*

$$
\begin{aligned}
B &= V \setminus A \\
\mathcal{X}_A &= \Omega_{ITM}(A) \\
V' &= B \cup \{\mathcal{X}_A\} \\
E' &= \mathsf{E}(B) \cup \bigcup\nolimits_{\langle \mathcal{X}[a], \mathcal{Y}[b]\rangle \in \mathsf{E}(A,B)} \{\langle \mathcal{X}_A[\mathcal{X}|a], \mathcal{Y}[b]\rangle\}.
\end{aligned}
$$

We abuse notation and write $G' = \Phi_{ITM}(G, A_1, A_2)$ instead of the more cumbersome $G' = \Phi_{ITM}(\Phi_{ITM}(G, A_1), A_2)$ and correspondingly for multiple sets $A_1, \ldots, A_l$.

Theorem 2. *The simulation transform is faithful.*

Proof. It is clear that the transform is faithful, since the parties in A are simply simulated and we merely replace the links to parties outside A with corresponding links with differently labeled slots.

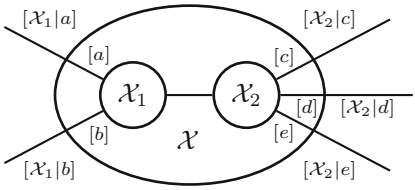

Fig. 14. Two ITMs $\mathcal{X}_1$ and $\mathcal{X}_2$ with links to other ITMs (not shown in the figure) are simulated by a single ITM $\mathcal{X}$ that inherits the links of all the original parties. More precisely, the slots $[a]$ and $[b]$ of $\mathcal{X}_1$ are exposed as the slots $[\mathcal{X}_1|a]$ and $[\mathcal{X}_1|b]$, and the slots $[c]$ and $[d]$, and $[e]$ of $\mathcal{X}_2$ are exposed as $[\mathcal{X}_2|c]$, $[\mathcal{X}_2|d]$, and $[\mathcal{X}_2|e]$.

A.3 Simulate Multiple Links

Suppose that two ITMs have multiple direct links between them. Then we simply plug in two routers and absorb these routers into the respective ITMs using wrappers. This is illustrated in Fig. 15.

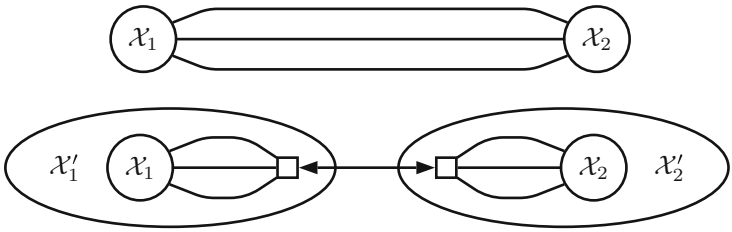

Fig. 15. The upper part shows two ITMs $\mathcal{X}_1$ and $\mathcal{X}_2$ that are linked by three links. The lower part shows how two routing wrappers can be used to form slightly modified ITMs $\mathcal{X}'_1$ and $\mathcal{X}'_2$ that are connected by a single link.

Definition 35 (Wrapper). *Let $\mathcal{X}$ be an ITM with slots $[a_1], \ldots, [a_l]$, let $\mathcal{R}$ be an l-router, define links $E = \{\langle \mathcal{X}[a_j], \mathcal{R}[j]\rangle\}_{j\in[l]}$, and define $\mathcal{X}' = \Omega_{wrap}(\mathcal{X}, ([a_1], \ldots, [a_l]), [a])$ to be the* wrapper *ITM that simulates $\mathcal{X}$ and $\mathcal{R}$ including the links in E, and identifies $\mathcal{R}[0]$ with a new slot $[a]$ of $\mathcal{X}'$. All other slots of $\mathcal{X}$ are exposed by $\mathcal{X}'$.*

Definition 36 (Swap). *Let $G = (V, E)$ be an ITM graph, let $\mathcal{X} \in V$, and define $\Omega_{swap}(E, \mathcal{X}, \mathcal{Y})$, where L is the set of common labels of slots of $\mathcal{X}$ and $\mathcal{Y}$, by*

$$\Omega_{swap}(E, \mathcal{X}, \mathcal{Y}) = \bigcup\nolimits_{a\in L} \bigcup\nolimits_{\langle \mathcal{X}[a], \mathcal{Z}[a]\rangle \in E} \{\langle \mathcal{Y}[a], \mathcal{Z}[a]\rangle\}.$$

We generalize Ω_{swap} to lists of ITMs in the natural way, i.e., we simply write $\Omega_{swap}(E, (\mathcal{X}_1, \mathcal{X}_2), (\mathcal{Y}_1, \mathcal{Y}_2))$ instead of $\Omega_{swap}(\Omega_{swap}(E, \mathcal{X}_1, \mathcal{Y}_1), \mathcal{X}_2, \mathcal{Y}_2)$ and similarly for longer lists.

Definition 37 (Link Simulation Transform). *Define the* link simulation transform *$\Phi_{links}(G, A) = (V', E')$, where $G = (V, E)$ is an ITM graph and $A = \{\mathcal{X}_1, \mathcal{X}_2\}$ with $A \subset V$ and $E_A = \mathsf{E}(\mathcal{X}_1, \mathcal{X}_2) = \{\langle \mathcal{X}_1[a_i], \mathcal{X}_2[b_i]\rangle\}_{i\in[l]}$, by*

$$\begin{aligned}
\mathcal{X}'_1 &= \Omega_{wrap}(\mathcal{X}_1, ([a_i])_{i\in[l]}, [a]) \quad \textit{where } [a] \textit{ is not a slot of } \mathcal{X}_1\\
\mathcal{X}'_2 &= \Omega_{wrap}(\mathcal{X}_2, ([b_i])_{i\in[l]}, [b]) \quad \textit{where } [b] \textit{ is not a slot of } \mathcal{X}_2\\
V' &= (V \setminus A) \cup \{\mathcal{X}'_1, \mathcal{X}'_2\}\\
E' &= \mathsf{E}(V \setminus A) \cup \Omega_{swap}(E \setminus E_A, (\mathcal{X}_1, \mathcal{X}_2), (\mathcal{X}'_1, \mathcal{X}'_2)) \cup \{\langle \mathcal{X}'_1[a], \mathcal{X}'_2[b]\rangle\}.
\end{aligned}$$

Theorem 3. *The link simulation transform is faithful.*

Proof. The flow of information between slots $[a_i]$ and $[b_i]$ is identical in the original ITM graph and its transform, since routers are deterministic and take no input, and we can recover the original ITM graph from its transform given A.

We abuse notation and simply write $\Phi_{links}(G)$ for the repeated application of the link simulation transform to, starting from G, a sequence of ITM graphs and any pair of ITMs with multiple links in it until no such pair exists.

A.4 Redundant Communication Models

Even if we absorb subparties into real parties and simulate multiple links with a single link as explained above we still need to combine multiple communication models into one to turn an ITM graph into a model. This is illustrated in Figs. 16 and 17 and formalized in the next definition.

Definition 38 (Redundant Communication Models). *Let $G = (V, E)$ be an ITM graph and let $B = \{\mathcal{C}_{k,1}, \ldots, \mathcal{C}_{k,l}\}$ be a set of ideal/real communication models in V. Then B is a* set of l-redundant ideal/real k-communication models *of G if $l > 1$ and there is a subset $A = \{\mathcal{X}_1, \ldots, \mathcal{X}_k, \mathcal{H}, \mathcal{Y}\}$ of V, such that $\mathsf{E}(A \cup B)$ is of the form*

$$\bigcup\nolimits_{j\in[l]} \left\{\langle \mathcal{H}[c_j], \mathcal{C}_{k,j}[\mathcal{A}]\rangle, \langle \mathcal{Y}[c_j], \mathcal{C}_{k,j}[\mathcal{F}]\rangle\right\} \cup \bigcup\nolimits_{i\in[k]} \left\{\langle \mathcal{C}_{k,j}[\mathcal{P}_i], \mathcal{X}_i[c_j]\rangle\right\}.$$

Note that $\mathcal{X}_j$ plays the role of a party, except that it is linked to multiple communication models. Similarly, $\mathcal{H}$ and $\mathcal{Y}$ represent an adversary and an ideal functionality, respectively, except that they are linked to multiple communication models.

We stress that the definition should be interpreted to say that all communication models of a set of redundant communication models must either be ideal or real and never a mix of both.

Definition 39 (Redundant Communication Model Transform). *Define the* redundant communication model transform $\Phi_{red}(G, B) = (V', E')$, *where $G = (V, E)$ is an ITM graph with a set of l-redundant k-communication models B (with notation from Definition 38), $c = ([c_1], \ldots, [c_l])$, $[c_i] \neq [\mathcal{C}]$, and $\mathcal{C}_k$ is a k-communication model, by*

$$\begin{aligned}
&\mathcal{X}'_i = \Omega_{wrap}(\mathcal{X}_i, c, [\mathcal{C}]), \mathcal{H}' = \Omega_{wrap}(\mathcal{H}, c, [\mathcal{C}]), \text{ and } \quad \mathcal{Y}' = \Omega_{wrap}(\mathcal{Y}, c, [\mathcal{C}]),\\
&V' = (V \setminus A) \cup \{\mathcal{C}_k, \mathcal{X}'_1, \ldots, \mathcal{X}'_k, \mathcal{H}', \mathcal{Y}'\}\\
&E' = \{\langle \mathcal{H}'[\mathcal{C}], \mathcal{C}_k[\mathcal{A}]\rangle, \langle \mathcal{Y}'[\mathcal{C}], \mathcal{C}_k[\mathcal{F}]\rangle\} \cup \bigcup\nolimits_{i\in[k]} \{\langle \mathcal{X}'_i[\mathcal{C}], \mathcal{C}_k[\mathcal{P}_i]\rangle\}\\
&\qquad \cup \Omega_{swap}\left(E, (\mathcal{H}, \mathcal{Y}, \mathcal{X}_1, \ldots, \mathcal{X}_k), (\mathcal{H}', \mathcal{Y}', \mathcal{X}'_1, \ldots, \mathcal{X}'_k)\right).
\end{aligned}$$

Note that if the redundant communication models are ideal, then each $\mathcal{X}_i$ is the result of a combining multiple dummy parties with the simulation transform, which means that $\mathcal{X}'_i$ is equivalent to a single dummy party. We abuse notation and simply write $\Phi_{red}(G)$ for the repeated application of the redundant communication model transform until there is no longer any set of redundant communication models.

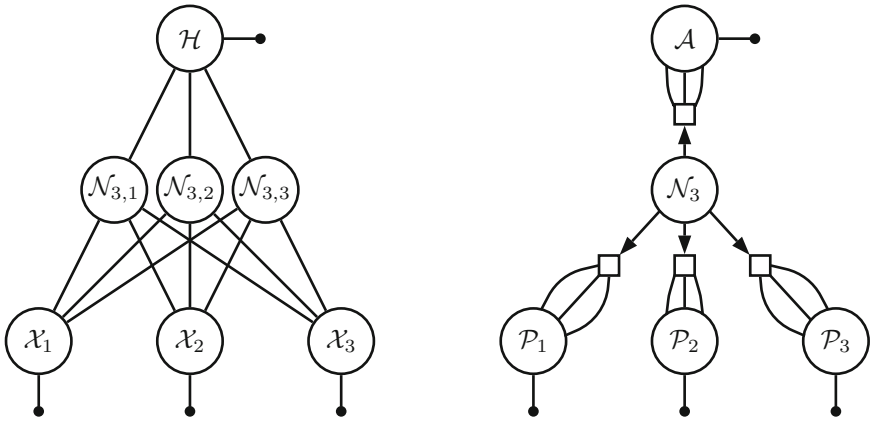

Fig. 16. The left side shows a real free model except that each party and the adversary is linked by a set of 3-redundant real 3-communication models. The right side shows how a single equivalent real communication model can be formed. Here it is understood in the figure that the routers with multiple links to a party or adversary would be absorbed into the party to reduce the number of links.

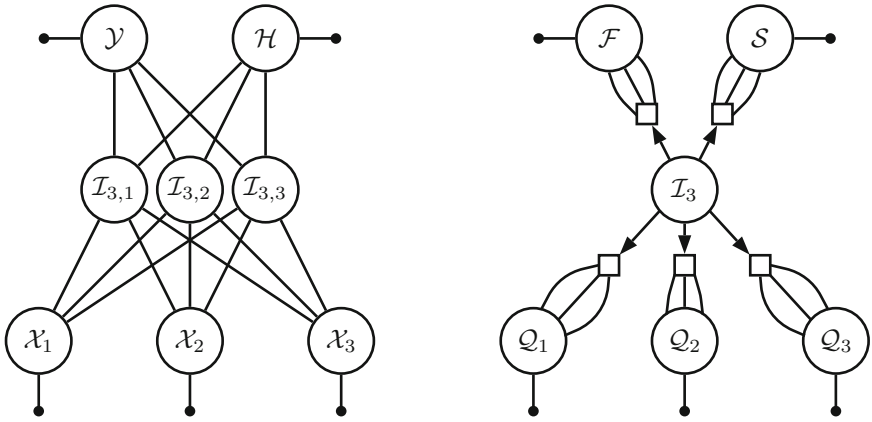

Fig. 17. The left side shows an ideal free model except that each party and the simulation adversary is linked by a set of 3-redundant ideal 3-communication models. The right side shows how a single equivalent ideal communication model can be formed. Here it is understood that the routers with multiple links to a party or adversary would be absorbed into the party.

Theorem 4. *The redundant communication model transform is faithful.*

Proof. Note that a communication model is routing messages based on the *prefixes* of messages. The routers used in the wrappers on the other hand route messages based on the *postfixes* of messages. This means that adding/removing a prefix commutes with adding/removing a postfix. Ideal communication models

behave in the same way. The probability that the same randomly chosen tag appears in two ideal communication models is exponentially small. (This is where we need the extra leg room in the definition of preservation of functionality.)

Remark 3. Consider a dummy party $\mathcal{Q}$ and two routers R and R' with the same number of slots l. If a slot $[a]$ of $\mathcal{Q}$ is linked to the 0th slot of R and the ith slot of R is linked to the ith slot of R' for $i \in [l]$, then an ITM $\mathcal{Q}'$ that simulates $\mathcal{Q}$, R, and R' and exposes the 0th slot of R' as $[a]$ is equivalent to $\mathcal{Q}$. Thus, if multiple dummy parties are simulated by a single ITM and then the corresponding redundant ideal communication models are combined, then we may view the resulting ITMs as dummy parties. We tacitly ignore this technicality below.

Given an ITM graph with multiple links between some parties or redundant communication models it is natural to simplify it by eliminating them. Thus, we define the *simplifying transform* as a short hand for cleaning up an ITM graph

$$\Phi_{sim}(G) = \Phi_{red}(\Phi_{links}(G)).$$

A.5 Transforms of Models

We are now ready to introduce transforms that turn one model into another. We begin with a transform that takes several free models, ideal or real, and applies the simulation transform to the adversaries, the subparties for each index, and the ideal functionalities if there are any.

Definition 40 (Combining Transform). *Define the* combining transform *by*

$$\Phi_{com}(G, H) = \Phi_{sim}(\Phi_{ITM}(G, H, X_1, \ldots, X_k, F))$$

where G is a (k, I)-model, $H = \{\mathcal{H}_1, \ldots, \mathcal{H}_l\}$ is a set of real/simulation subadversaries with the same real parent adversary, $\mathcal{X}_{j,i}$ is the ith party linked to the same communication model $\mathcal{C}_{k,j}$ as $\mathcal{H}_j$, and $X_i = \{\mathcal{X}_{1,i}, \ldots, \mathcal{X}_{l,i}\}$. Furthermore, if the communication models are ideal, then $\mathcal{F}_j$ is the ideal functionality linked to $\mathcal{C}_{k,j}$ and $F = \{\mathcal{F}_1, \ldots, \mathcal{F}_l\}$, and otherwise $F = \emptyset$.

We stress that the definition must be interpreted to say that the input adversaries are either all real or all ideal and never a mix. We sometimes abuse notation and write $\Phi_{com}(G, F)$, where F is a set of ideal functionalities, to denote $\Phi_{com}(G, H)$ where H is the set of simulation adversaries linked to the ideal communication models linked to the ideal functionalities in F. We also write $\Phi_{com}(G)$ to denote the transform that repeatedly applies the combining transform to a sequence of models starting with G until there are no real/simulation subadversaries with the same parent in a model.

Definition 41 (Absorbing Transform). *Define the* absorbing transform *by*

$$\Phi_{abs}(G, A) = \Phi_{sim}(\Phi_{ITM}(G, A, P_1, \ldots, P_k))$$

where G is a (k, I)-model, $A = \{\mathcal{A}_1, \ldots, \mathcal{A}_l\}$ is a set of real subadversaries such that $\mathcal{A}_{j+1}$ is a real subadversary of $\mathcal{A}_j$ for $j = 1, \ldots, l-1$, $\mathcal{P}_{j,i}$ is the ith party linked to the same real communication model as $\mathcal{A}_j$, and $P_i = \{\mathcal{P}_{1,i}, \ldots, \mathcal{P}_{l,i}\}$.

Note that the absorbing transform not only absorbs the subprotocol. It also absorbs real subadversaries into the root adversary. We abuse notation and write $\Phi_{abs}(G)$ for the transform that repeatedly applies the absorbing transform to a sequence of models starting with G until no real subadversary exists in a resulting model. We say that a model without subprotocols is normalized, i.e., a model for which the absorbing transformation can not be applied.

Definition 42 (Normalized Model). *A (k, I)-model is* normalized *if it has no subprotocols, or equivalently no real subadversaries.*

Another natural transform is to collapse parts of models. This is useful to focus on particular parts of a model and allows generalizing the composition theorem.

Definition 43 (Collapsing Transform). *Define the* collapsing transform *$\Phi_{col}(G, \rho)$, where G is a (k, I)-model and ρ is a hybrid protocol embedded in G, as the transform that repeatedly applies the combining and/or the absorbing transforms to G except the free model uniquely identified by ρ until no longer possible.*

Note that if G is a model, then $\Phi_{col}(G, \rho)$ has at most one ideal functionality outside of the free model in G based on ρ, i.e., the corresponding protocol is of the form $\pi(\rho, \mathcal{F})$ for some root protocol π and ideal functionality $\mathcal{F}$. In particular, we can use the collapsing transform without restriction to put a model into a minimal form where all ideal functionalities have been combined into a single ideal functionality and all subprotocols have been absorbed.

Definition 44 (Minimal Model). *A (k, I)-model is* minimal *if it is normalized and has at most one ideal functionality.*

A.6 Adversary Converter

We need an explicit way to map an arbitrary adversary (not constructed through our transforms) for a transformed protocol back into an equivalent adversary of the protocol.

Fortunately, the transforms leave a blue print for what to do. Note that a template adversary resulting from our transforms of models consists of an original adversary and routers forming trees where the leaves of the trees are linked to the original adversary and the roots of the trees are exposed as slots of the adversary. We may think of the trees as mapped onto an annulus where the inner circle represents the original adversary and the outer circle represents the template adversary. This is illustrated in Fig. 18 and made precise below.

To convert an adversary with identical slots to the template adversary, we simply fold the annulus inside out, link the roots of the trees of routers to the adversary and relabel the slots of the leaves of the trees of routers to the labels of the original adversary. If we plug this converted adversary into the original model and transform it, then we get a transformed converted adversary that is

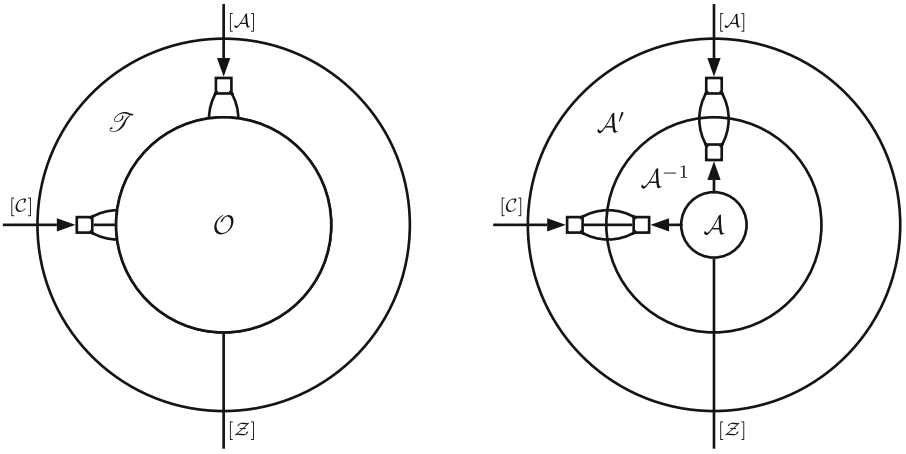

Fig. 18. The left side illustrates a template adversary $\mathcal{T}$ resulting from applying some of the above transforms on an original adversary $\mathcal{O}$ that does not corrupt any parties. The original model could for example have had two subprotocols that were absorbed and two ideal functionalities that were combined.

equivalent to the adversary, since each tree of routers is effectively canceled by its mirror embedded in the converted adversary.

We say that a set of routers form a tree if exactly one router has a free 0th slot and the 0th slot of every other router is linked to the ith slot of another router for some $i > 0$. We say that the free 0th slot is the root of the tree and all other free slots are leaves of the tree.

Definition 45 (Adversary Converter). *The* adversary converter Φ_{adv} *is defined as follows. Let* $\mathcal{T}$ *be an adversary that simulates an original adversary* $\mathcal{O}$ *and trees* $t_1, \ldots, t_r$ *of routers with roots exposed as slots* $[a_1], \ldots, [a_r]$ *of* $\mathcal{T}$ *and leaves* $[b_{j,1}], \ldots, [b_{j,s_j}]$ *of* t_j *linked to slots* $[c_{j,1}], \ldots, [c_{j,s_j}]$ *of* $\mathcal{O}$*. Let* $\mathcal{A}$ *be an adversary with slots* $[a_1], \ldots, [a_r]$*. Then* $\Phi_{adv}(\mathcal{T}, \mathcal{A})$ *is the adversary that simulates* $\mathcal{A}$ *and* $t_1, \ldots, t_r$ *with the set of links* $\left\{\langle \mathcal{A}[a_j], t_j[0]\rangle\right\}_{j\in[r]}$ *and exposes* $[b_{j,i}]$ *as* $[c_{j,i}]$ *for* $j = 1, \ldots, r$ *and* $i \in 1, \ldots, s_j$*.*

The importance of the adversary converter can be illustrated as follows. Suppose we are given a model M and wish to prove that its embedded hybrid protocol securely realizes some ideal functionality $\mathcal{F}$. To do this we need to show that for every adversary $\mathcal{A}$, there is a suitable simulator $\mathcal{S}$. Given an adversary we can of course apply our transforms and get a new model M' along with a transformed adversary $\mathcal{A}'$ for which the simulator is still suitable.

More interesting is to consider the transformed protocol of M' directly. If this securely realizes $\mathcal{F}$, then for every adversary $\mathcal{A}'$ in M', there exists a suitable simulator $\mathcal{S}'$. We may plug in a place-holder adversary $\mathcal{O}$ and apply the transforms to get a template adversary $\mathcal{T}$ as in the definition. Then we can use this

to construct an adversary $\mathcal{A}$ such that if we transform M with $\mathcal{A}$ we will recover M' and an adversary that is functionally identical to $\mathcal{A}'$.

Thus, we can safely prove the security for any transformed protocol and conclude that any other transformation of it is secure as well.

References

1. Beaver, D.: Foundations of secure interactive computing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 377–391. Springer, Heidelberg (1992)
2. Canetti, R.: Security and composition of multi-party cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
3. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 136–145. IEEE Computer Society Press (2001). (Full version at Cryptology ePrint Archive, Report 2000/067. http://eprint.iacr.org, October 2001)
4. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. Cryptology ePrint Archive, Report 2014/553 (2014). http://eprint.iacr.org/
5. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press, New York (2000)
6. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: 19th ACM Symposium on the Theory of Computing (STOC), pp. 218–229. ACM Press (1987)
7. Goldwasser, S., Levin, L.A.: Fair computation of general functions in presence of immoral majority. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 77–93. Springer, Heidelberg (1991)
8. Maurer, U.: Constructive cryptography – a new paradigm for security definitions and proofs. In: Mödersheim, S., Palamidessi, C. (eds.) TOSCA 2011. LNCS, vol. 6993, pp. 33–56. Springer, Heidelberg (2012)
9. Maurer, U., Renner, R.: Abstract cryptography. In: The Second Symposium on Innovations in Computer Science, ICS 2011, pp. 1–21, January 2011
10. Micali, S., Rogaway, P.: Secure computation. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 392–404. Springer, Heidelberg (1992)
11. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: 7th ACM Conference on Computer and Communications Security (CCS), pp. 245–254. ACM Press (2000)
12. Wikström, D.: On the security of mix-nets and hierarchical group signatures. Doctoral thesis, Department of Numerical Analysis and Computer Science, Royal Institute of Technology, TRITA NA 05–38, ISSN 0348–2952, ISRN KTH/NA/R–05/38–SE, ISBN 91-7283-717-9, December 2005. http://www.kth.se
13. Wikström, D.: Simplified submission of inputs to protocols. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 293–308. Springer, Heidelberg (2008)