Universally Composable Security: A New Paradigm for Cryptographic Protocols*

Ran Canetti†

October 9, 2001

Abstract

We propose a new paradigm for defining security of cryptographic protocols, called universally composable security. The salient property of universally composable definitions of security is that they guarantee security even when a secure protocol is composed with an arbitrary set of protocols, or more generally when the protocol is used as a component of an arbitrary system. This is an essential property for maintaining security of cryptographic protocols in complex and unpredictable environments such as the Internet. In particular, universally composable definitions guarantee security even when an unbounded number of protocol instances are executed concurrently in an adversarially controlled manner, they guarantee non-malleability with respect to arbitrary protocols, and more.

We show how to formulate universally composable definitions of security for practically any cryptographic task. Furthermore, we demonstrate that practically any such definition can be realized using known general techniques, as long as only a minority of the participants are corrupted. We then proceed to formulate universally composable definitions of a wide array of cryptographic tasks, including authenticated and secure communication, key-exchange, public-key encryption, signature, commitment, oblivious transfer, zero-knowledge, and more. We also make initial steps towards studying the realizability of the proposed definitions in other natural settings.

Keywords: cryptographic protocols, security analysis of protocols, concurrent composition, universal composition.

*An earlier version of this work appeared under the title "A unified framework for analyzing security of protocols".
†IBM T.J. Watson Research Center. Email: canetti@watson.ibm.com.

ISSN 1433-8092

Contents

1 Introduction

Rigorously demonstrating that a protocol "does its job securely" is an essential component of cryptographic protocol design. This requires coming up with an appropriate mathematical model for representing protocols, and then formulating, within that model, a *definition of security* that captures the requirements of the task at hand. Once such a definition is in place, we can show that a protocol "does its job securely" by demonstrating that it satisfies the definition of security in the devised mathematical model.

However, coming up with a good mathematical model for representing protocols, and even more so formulating appropriate definitions of security within the devised model, turns out to be a tricky business. The model should be rich enough to represent a large variety of realistic adversarial behaviors, and the definition should guarantee that the intuitive notion of security is captured, for any adversarial behavior under consideration. This in particular means that security should be maintained when the protocol is used as a component within a larger system.

In contrast, cryptographic primitives (or, *tasks*) were traditionally first defined as stand-alone protocol problems. This allowed for relatively concise and intuitive problem statement, as well as simple analysis of protocols. However, in many cases it turned out that the initial definitions were insufficient in more complex contexts, and especially when deploying protocols within larger systems or protocol environments. Examples of such cases include encryption (where semantic security [GM84] was later augmented with several flavors of security against chosen ciphertext attacks, e.g. [NY90, DDN00, RS91, BDPR98], and adaptive security [BH92, CFGN96]); commitment (where the original notions were augmented with some flavors of non-malleability [DDN00, DIO98, FF00] and equivocability, e.g., [BCC88, B96]); Zero-Knowledge protocols (where the original notions [GMRa89, GO94] were shown not to be closed under parallel and concurrent composition [GK88, F91, DNS98]); Key Exchange (where the original notions did not suffice for providing secure sessions [BR93, BCK98, sh99, CK01]); Oblivious Transfer [R81, EGL85, GM00]; and more.

One way to capture the security concerns that arise in some specific protocol environment or in a given application is to directly represent the given environment or application within an extended definition of security. (Such an approach was taken, for instance in the cases of concurrent zero-knowledge [DNS98] and oblivious transfer [GM00] as well as non-malleability of protocols [DDN00], where the definitions explicitly model several adversarially coordinated instances of the protocol in question.) This approach, however, results in definitions with ever-growing complexity, and is inherently limited in scope since it addresses only specific environments and concerns.

An alternative approach, taken in this work, is to use definitions that treat the protocol as stand-alone but guarantee *secure composition.* That is, here definitions of security inspect only a single copy of the protocol *in vitro.* Security in complex settings (where a protocol instance may run concurrently with many other protocol instances, on arbitrary inputs and in an adversarially controlled way) is guaranteed via a general *composition theorem.* On top of simplifying the process of formulating a definition and analyzing protocols, this approach guarantees security in arbitrary protocol environments, even unpredictable ones which have not been explicitly considered.

In order to make such an approach (and in particular, such a composition theorem) meaningful, we first need to have a general framework in which to represent cryptographic protocols and the security requirements of cryptographic tasks. Indeed, several general definitions of secure protocols were developed over the years, e.g. [GL90, MR91, B91, BCG93, PW94, C00, HM00, PSW00, DM00, PW00]. Some of these definitions were shown to maintain security under natural composition operations. These definitions are obvious candidates for such a general framework. However, the composition operations considered in those works fall short of guaranteeing general secure composition of cryptographic protocols, especially in settings where security holds only for computationally bounded adversaries and numerous protocols may be running concurrently in an adversarially coordinated way. Moreover, many of these works choose to concentrate on the task of *secure function evaluation* which, in spite of its generality, does not capture the requirements of many cryptographic primitives, which are reactive in nature. (Secure function evaluation is the task where a set of parties wish to jointly compute a known function of their secret inputs.) We further elaborate on some of these works and their relation to the present one in Section 1.4.

This work proposes a new framework for representing and analyzing cryptographic protocols. Within this framework, we propose a general methodology for expressing the security requirements of practically any cryptographic task in a clear, concise and intuitively satisfying way. The salient property of definitions of security generated using this methodology is that they guarantee security even when the given protocol is running in an arbitrary and unknown multi-party environment. In particular, security is preserved under a very general composition operation that captures, as special cases, the standard notions of concurrent composition (with arbitrarily many instances of either the same protocol or other protocols), non-malleability, and more. We call this composition operation universal composition, and say that definitions of security in this framework (and the protocols that satisfy them) are universally composable (UC).

UC definitions of security tend to be more restrictive than other definitions of security. (This extra restrictiveness may be expected in light of their strong security properties.) Nonetheless, we show that in settings where parties have access to a set of servers, at most a minority of which may be corrupted, standard cryptographic techniques (e.g., [BGW88, RB89, CFGN96]) can be used to carry out practically *any cryptographic task* in a universally composable way. We also formulate UC definitions of a number of well known cryptographic tasks, such as authenticated and secure communication, key-exchange, public-key encryption, signature, commitment, oblivious transfer, Zero-Knowledge, secret sharing, and general function evaluation. In some cases, we present initial results regarding the realizability of the definitions. In other cases realizing the definitions is left open.

1.1 The proposed framework

We briefly sketch the proposed framework and highlight some of its properties. A more comprehensive overview is postponed to Section 2. To better understand the present framework, let us first briefly sketch the definitional approach of [C00], which is the starting point of this work. (The work of [C00] is, in turn, based to a large extent on [B91, MR91, GL90]). This work is geared towards capturing the task of secure function evaluation in a synchronous, ideally authenticated network. The idea is to first formulate a model representing the process of protocol execution in real-life. This is called the real-life model. Next, in order to capture the security requirements of a given task, formulate an ideal process for carrying out the task. Then say that a protocol securely realizes the task at hand if running the protocol in the real-life model amounts to "emulating" the ideal process for that task.

The real-life model of computation in [C00] consists of a set of interactive Turing machines (ITMs) representing the parties running the protocol, plus an ITM representing the adversary. The parties and adversary interact on a given set of inputs and each party generates local output. The concatenation of the local outputs of all parties and adversary is called the global output. The ideal process for evaluating some function f is defined similarly, with the important exception that the parties hand their inputs to an incorruptible *trusted party*, which evaluates f and hands the corresponding outputs back to the parties. A protocol π securely evaluates a function f if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that, for any input vector, the global output of running π with $\mathcal{A}$ in the real-life model is indistinguishable from the global output of the ideal process for f with adversary $\mathcal{S}$. This definitional approach is sufficient for capturing "stand-alone" security of protocols. It is also shown to be closed under non-concurrent composition.

The present framework preserves the overall structure of that approach. The difference lies in new formulations of the models of computation and the notion of "emulation". Specifically, we introduce an additional computational entity, called the environment machine, to both the real-life model and the the ideal process. The environment machine is another ITM that represents "whatever is external to the current protocol execution". This includes other protocol executions and their adversaries, human users, etc. The environment has external input that is known only to itself. It provides all the inputs to all parties and reads all their outputs. More importantly, the environment interacts with the adversary freely throughout the computation. That is, between any two atomic operations carried out by the adversary (e.g., delivery of a message, or corruption of a party) the adversary and the environment may exchange arbitrary information. The security requirement is now that executing the protocol in the real-life model should "look the same" as the ideal process *from the point of view of the environment.* More precisely, a protocol π securely realizes a trusted party for some function f if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that *no feasible environment* can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and π in the real-life model or with $\mathcal{S}$ in the ideal process for f. In a way, the environment serves as an "interactive distinguisher" between the protocol execution and the ideal process. Note that the interaction between $\mathcal{S}$ and the environment is inherently "black-box" from the point of view of $\mathcal{S}$. (This is so since $\mathcal{S}$ does not know the input of $\mathcal{Z}$.) This requirement is essential for our proof of the composition theorem.[1]

Another modification to the definition allows capturing not only secure function evaluation but also *reactive* tasks where new input values become known throughout the computation, and may depend on previously generated output values. This is obtained by replacing the "trusted party" in the ideal process for secure function evaluation with a general algorithmic entity called an ideal functionality. The ideal functionality, which is modeled as another ITM, repeatedly receives inputs from the parties and provides them with appropriate output values. This way, it is guaranteed that the outputs of the parties in the ideal process have the expected properties with respect to the inputs, even when new inputs are chosen adaptively based on previous outputs.

Yet another difference from [C00] is that here we model networks where the communication is *open, unauthenticated,* and *asynchronous* (without guaranteed delivery of messages). We also concentrate on the case where the adversary is probabilistic polynomial time (PPT). This modeling seems more suitable for analyzing protocols in realistic settings.

Universal Composition. We show that the following property holds with respect to a protocol ρ that securely realizes some ideal functionality $\mathcal{F}$. Let π be some arbitrary protocol (we think of π as an "application protocol") that operates in a model where all parties have ideal access to multiple instances of $\mathcal{F}$. That is, in this model (which we call the $\mathcal{F}$-hybrid model) the parties, the adversary and the environment interact as in the real-life model, and in addition the parties can privately communicate with as many instances of $\mathcal{F}$ as they wish. It is stressed that the different instances of $\mathcal{F}$ are running at the same time without any global coordination. They are distinguished via

[1]A very limited variant of the notion of environment appears in [C00]. That variant, aimed at providing non-concurrent composition in the presence of an adaptive adversary, interacts with the parties and the adversary only at few occasions throughout the computation. In particular, the variant there is insufficient for preserving security under concurrent composition.

special identifiers, generated by the calling protocol.

Now, construct the composed protocol π^ρ from π by replacing each call to a new instance of $\mathcal{F}$ with an invocation of a fresh copy of ρ. Similarly, a message sent to an existing instance of $\mathcal{F}$ is replaced with an input value given to the corresponding invocation of ρ, and any output of an invocation of ρ is treated as a message received from the corresponding instance of $\mathcal{F}$. (Note that a run of protocol π^ρ may have an unbounded number of copies of ρ which are running concurrently on related inputs.)

The universal composition theorem states that running protocol π^ρ in the plain real-life model has essentially the same effect as running protocol π in the $\mathcal{F}$-hybrid model. More precisely, it guarantees that for any real-life adversary $\mathcal{A}$ there exists an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no environment machine can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π^ρ in the plain real-life model, or with $\mathcal{H}$ and parties running π in the $\mathcal{F}$-hybrid model. In particular, if π securely realizes some ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model then π^ρ securely realizes $\mathcal{G}$ from scratch.

Notice that, while other composition theorems address only the case where a single protocol instance is composed with another protocol, here the hybrid model allows for an unbounded number of instances of the composed protocol to run concurrently. It may appear that this more complex formulation of the composition operation is not necessary, since it can be obtained by iteratively composing all instances of the protocol, one at a time, with an outside protocol. However, in our computational setting the composition theorem can be safely applied only a *constant* number of times, otherwise the complexity of the adversary $\mathcal{H}$ in the hybrid model may become super-polynomial. Consequently, this weaker formulation of the composition theorem only guarantees secure composition of a *constant* number of protocol instances running concurrently.

Interpreting the composition theorem. Traditionally, composition theorems are treated as tools for modular design and analysis of complex protocols. (For instance, this is the main motivation in [MR91, C00, DM00, PW00].) That is, given a complex task, first partition the task to several, simpler sub-tasks. Then, design protocols for securely realizing the sub-tasks, and in addition design a protocol for realizing the given task in a model where ideal evaluation of the sub-tasks is possible. Finally, use the composition theorem to argue that the protocol composed from the already-designed sub-protocols securely realizes the given task. An example of a context where this interpretation is put to use is the proof of security in [CKOR00]. Note that in this interpretation the protocol designer knows in advance which protocol instances are running together and can control the way protocols are scheduled.

In contrast, here we use the composition theorem as a tool for gaining confidence in the sufficiency of a definition of security in some protocol environment. Indeed, protocols that satisfy a universally composable definition are guaranteed to maintain their security within any protocol environment — even environments that are not known a-priori, and even environments where the participants in a protocol execution are unaware of other instances of the protocol (or other protocols altogether) that may be running concurrently in the system in an adversarially coordinated manner. This is a strong guarantee.

1.2 General satisfiability of the definitions

Definitions of security in the proposed framework tend to be more stringent than other definitions. Moreover, the existing proofs of security of many known protocols do not work in the present framework. Examples include most known Zero-Knowledge protocols, the protocol generator of

[GMW87, G98] and more. This is mainly due to the fact that one of the most common proof-techniques, namely black-box simulation with rewinding of the adversary, does not work in the present framework. (Indeed, here the ideal-process adversary has to interact with the environment machine which cannot be "rewound".)

Nonetheless, it can be seen that some known protocols for the general task of secure function evaluation are, in fact, universally composable. For instance, the [BGW88] protocol (say, with the simplification of [GRR98]), together with encrypting each message using non-committing encryption [CFGN96], is universally composable as long as less than a third of the parties are corrupted. Using [RB89], any corrupted minority is tolerable. The asynchronous setting can be handled using the techniques of [BCG93, BKR94].

We use this fact to demonstrate that practically *any* ideal functionality — even reactive ones and even "two-party functionalities" (i.e., functionalities where only two parties have inputs and outputs) — can be securely realized in the proposed framework. Specifically, our solution uses a set of special parties that have no local inputs or outputs but only assist other parties in realizing the given functionality. All parties share their inputs among the special parties (which can be regarded as "servers"), who run the appropriate multiparty function evaluation protocol and send the output values to the appropriate parties. Iterated evaluations (which may involve implementing ideal functionalities that maintain internal state between invocations) are handled in standard ways. This solution works as long as only a minority of the servers are corrupted. We thus obtain:

Theorem (informal statement): *Any ideal functionality can be securely realized in the present framework, assuming that secure encryption protocols exist and that the network contains a special set of parties such that only a minority of which can be corrupted.*

1.3 UC definitions of some specific tasks

We formulate and study universally composable definitions of a number of standard cryptographic tasks. In fact, much of the definitional work is already done by the general framework described above. All that is left to do on the definitional side is to formulate ideal functionalities that capture the security requirements of these tasks.

We first address the task of *message authentication:* the corresponding functionality, $\mathcal{F}_{\text{AUTH}}$, is invoked with a request by some party, P_i, to transmit a message m to another party, P_j. Then $\mathcal{F}_{\text{AUTH}}$ ideally sends (P_i, m) to P_j and the adversary, and halts. (Forwarding (P_i, m) to the adversary captures the fact that secrecy is not provided.) This way, the standard computational model where the communication is ideally authenticated is rephrased as the $\mathcal{F}_{\text{AUTH}}$-hybrid model. This notion is the natural universally composable extension of the *authenticators* of [CHH00, BCK98]. In particular, the two authenticators presented in [BCK98] securely realize $\mathcal{F}_{\text{AUTH}}$ given an authenticated initialization phase.

The task of providing secure (i.e., authenticated *and secret*) transmission of individual messages is addressed next. It is seen that standard semantically secure encryption (or alternatively non-committing encryption for adaptive adversaries) are sufficient in order to realize the secure communication functionality in the $\mathcal{F}_{\text{AUTH}}$-hybrid model, if any message is encrypted using a different public/private key pair.

Next we formulate ideal functionalities that capture the tasks of *secure sessions* and *key exchange.* Secure sessions is an extension of secure transmission of individual messages to the case where a sequence of messages between a pair of parties are secured together. The main advantage of this functionality over the previous ones is that it allows for more efficient realizations, via key-exchange combined with symmetric cryptography using the generated keys. The key exchange functionality essentially provides parties with "ideally chosen keys." In particular, protocols that securely realize $\mathcal{F}_{\text{KE}}$ are guaranteed to satisfy the security notion of [CK01]. Furthermore, most of the Key-Exchange protocols presented in [CK01] securely realize $\mathcal{F}_{\text{KE}}$.

Next the tasks of *public-key encryption* and *digital signatures* are addressed. Securely realizing the signature ideal functionality turns out to be essentially equivalent to existential security against chosen message attacks as in Goldwasser Micali and Rivest [GMRi88]. In the case of public-key encryption (where many messages may be encrypted by different parties using the same key), securely realizing the proposed functionality turns out to be closely related to (but more relaxed than) security against adaptive chosen ciphertext attacks [DDN00, RS91, BDPR98].

We then proceed to formulate ideal functionalities representing "classic" two-party primitives such as *coin-tossing, commitment, zero-knowledge,* and *oblivious-transfer.* These primitives are treated as two-party protocols in a multi-party setting. As usual, the composition theorem guarantees that security is maintained under concurrent composition, either with other copies of the same protocol or with other protocols, and within any application protocol. In particular, non-malleability with respect to an *arbitrary* set of protocols is guaranteed. Unfortunately, these functionalities cannot be securely realized by two-party protocols in the bare model of computation (or even in the $\mathcal{F}_{\text{AUTH}}$-hybrid model). This result is shown in [CF01] for the cases of commitment and coin-tossing. Using similar techniques, we extend this result to the cases of oblivious transfer and zero-knowledge. Nonetheless, as is demonstrated in [CF01], the commitment and zero-knowledge functionalities can be securely realized by two-party protocols in a hybrid model with ideal access to the coin-tossing functionality. (It is interesting to note that hybrid model with ideal access to the coin-tossing functionality turns out to be a reformulation of the popular *common random string model* of [BFM89].)

Finally, we formulate ideal functionalities that capture traditional multi-party tasks such as Verifiable Secret Sharing and Secure Function Evaluation in *synchronous* networks. In particular, we obtain the first definition of protocols for secure function evaluation that is closed under concurrent composition in a setting where all the communication is public. (Two-party Secure Function Evaluation is obtained as a special case.)

1.4 Related work

Several definitional works on security of protocols have been carried out over the years. The works of [GL90, MR91, B91, C95] are surveyed in [C00]. Here we very briefly review some definitional efforts that are closely related to the present work.

Pfitzmann et. al. [PW94, PSW00, PSW00a, PW00, PW01] were the first to formally model the security requirements of general reactive systems. In a series of works that contain many interesting ideas, they model security of reactive systems in an extended finite-state machine model of computation that is essentially equivalent to the I/O automata model of [L96]. In particular, they introduce the notion of an honest user that 'sees' the functionality (i.e., the inputs and outputs) of a given system, and say that one system "simulates" another if the honest user cannot tell the difference between the two systems. However, they stop short of defining security of protocols for realizing a given task. They also state a composition theorem with respect to their framework; their composition theorem is weaker than the one here in that it deals only with the case where a single protocol execution is carried out concurrently with the calling protocol. (In contrast, much of the complexity in protocol composition appears only when the number of composed copies is not a-priori bounded.) These works contain also descriptions of ideal systems for public-key encryption and certified mail.

Canetti [C00] presents a definition of *secure function evaluation* in a variety of computational settings, including the case where the communication is public and security is guaranteed only against a computationally bounded adversary. Some of the definitions there also use an "environment machine". However, there both the purpose of the environment and its pattern of interaction with the other participants are different than here. The definitions of [C00] are shown to be closed under a composition theorem similar to the one here, but only in the *non-concurrent* case where no more than a single protocol execution is running at any point in time.

Dodis and Micali [DM00] build on the definition of Micali and Rogaway [MR91] for information-theoretically secure function evaluation in *synchronous* networks, where ideally private communication channels are assumed. In that setting, they prove that their definition of security is closed under a general concurrent composition operation similar to the one in this work. They also formulate an additional and interesting composition operation (called *synchronous composition*) that provides stronger security guarantees, and show that their definition is closed under that composition operation in cases where the scheduling of the various invocations of the protocols can be controlled. However, their definition applies only to settings where the communication is ideally private. It is not clear how to extend this definitional approach to realistic settings where the adversary can eavesdrop to the communication between honest parties.

The pioneering work of Dolev, Dwork and Naor [DDN00] points out some important security concerns that arise when running cryptographic protocols within a larger system. In particular, they define and construct encryption schemes secure against chosen ciphertext attacks, non-malleable commitment schemes, and more. That work provides motivation for the present one. In particular, making sure that the concerns pointed out in [DDN00] are answered plays a central role in the present framework.

On the formal-methods approach to security analysis. A large body of work on analyzing security of protocols using techniques for formal verification of computer programs has been carried out over the years (a very partial list of works includes [DY83, BAN90, M94a, KMM94, L96, AG97, AFG98]). In particular, basing themselves on the π-calculus of [MPW92], Abadi and Gordon [AG97] provide a calculus (called *spi-calculus*) for arguing about security of authentication protocols. This line of analysis does not take into account computational limitations of adversaries. In order to make analysis possible, cryptographic primitives and protocol actions are modeled via symbolic operations on formal variables with well-defined, deterministic, syntactic properties. This in effect amounts to representing cryptographic primitives as "ideal black boxes". Consequently, these works do not capture the realistic and inherent weaknesses of cryptographic primitives as functions on bit-strings and cannot be used to guarantee security of protocols (say, based on some underlying hardness assumption), as is done here. Nonetheless, these works have proven to be instrumental in finding security weaknesses in some protocols and for gaining some assurance in the security of others. In addition, these works are typically amenable to automation of the analysis; this is a great advantage.

A promising definitional approach using general formal methods is pursued by Lincoln, Mitchell, Mitchell and Schedrov [LMMS98, LMMS99]. They introduce a different variant of the π-calculus, in an effort to incorporate random choices and computational limitations of the adversary in the model. In that setting, their approach has a number of similarities to ours. They define a notion of *observational equivalence* (which has similarities to our notion of emulation), and say that a real-life process is secure if it is observationally equivalent to an "ideal process" where the desired functionality is guaranteed. However, their ideal process must vary with the protocol to be analyzed, and they do not seem to have an equivalent of the notion of an "ideal functionality" which is associated only with the task and is independent of the analyzed protocol. This makes it harder to formalize the security requirements of a given task. Furthermore, they do not state a composition theorem in their model. Still, as is the case with the other works based on formal-methods, this model has the great advantage that it seems amenable to automation of the analysis process using existing tools.

Potentially, definitions of security in the present framework may serve as a computationally-sound basis for formal-methods based automated analysis of protocols. More specifically, ideal functionalities, as defined and used here, may be regarded as idealizations of cryptographic tasks that "hide the dirty details of cryptography" from applications, but at the same time are realizable by actual protocols. Consequently, it may be possible to develop calculi of concurrent protocols where cryptography is represented via ideal functionalities that are in fact securely realizable (say, based on some hardness assumption). Such calculi may lead to automated proofs of security of protocols that maintain cryptographic soundness. See more on possible connections between these works and the present one in Section 9.

Organization. Section 2 presents an overview of the framework, definition of security, and composition theorem. The basic model for representing multiparty protocols is presented and motivated in Section 3. The general definition of security is presented in Section 4. The composition theorem and its proof are presented in Section 5. Some extensions and variants of the basic model are discussed in Section 6. Section 7 demonstrates how to securely realize any ideal functionality in some settings, and Section 8 presents and discusses a number of ideal functionalities that capture some known cryptographic primitives. Finally, Section 9 suggests some directions for future research.

2 Overview of the framework

This section presents an overview of the framework, the definition of security, and the composition theorem. The presentation builds on the intuition provided in the Introduction (Section 1.1).

2.1 The basic framework

As sketched in Section 1.1, protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality", which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. In the rest of this subsection we overview the model for protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

We concentrate mainly on the following standard model, aimed at representing current realistic communication networks (such as the Internet). The network is *asynchronous* without guaranteed delivery of messages. The communication is public and *unauthenticated.* That is, the adversary may delete, modify, and generate messages at wish. Parties may be broken into (i.e., become corrupted) throughout the computation, and once corrupted their behavior is arbitrary (or, *Byzantine*). Finally, all the involved entities are restricted to probabilistic polynomial time (or, "feasible") computation. Most other standard models of computation (e.g., authenticated communication, synchronous message delivery, the common reference string model, or computationally unbounded adversaries) can be captured via appropriate modifications to the basic model. See more details within.

Protocol syntax. Following [GMRa89, G95], a protocol is represented as a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs. We concentrate on a model where the adversaries have an arbitrary additional input, or an "advice". From a complexity-theoretic point of view, this essentially implies that adversaries are non-uniform ITMs.

Protocol execution in the real-life model. We sketch the process of executing a given protocol π (run by parties $P_1, ..., P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input z. All parties have a security parameter $k \in \mathbf{N}$ and are polynomial in k. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some P_i) is activated. The environment is activated first. In each activation it may read the contents of the output tapes of all parties, and may write information on the input tape of either one of the parties or of the adversary. Once the activation of the environment is complete (i,e, once the environment enters a special waiting state), the entity whose input tape was written on is activated next.

Once the adversary is activated, it may read its own tapes and in addition the contents of the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape[2], or corrupt a party. Upon corrupting a party, the adversary gains access to all the tapes of that party and controls all the party's future actions. Finally, the adversary may write arbitrary information on its output tape. If the adversary delivered a message to some uncorrupted party in an activation then this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

Once a party is activated (either due to an input given by the environment or due to a message delivered by the adversary), it follows its code and possibly writes local outputs on its output tape and outgoing messages on its outgoing communication tape. Once the activation of the party is complete the environment is activated. The protocol execution ends when the environment completes an activation without writing on the input tape of any entity. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Discussion. Several remarks are in order at this point. First notice that, throughout the process of protocol execution, the environment $\mathcal{Z}$ has access only to the input and output tapes of the parties. It does not have direct access to the communication among the parties. In contrast, the adversary $\mathcal{A}$ only sees the communication among the parties and has no access to their input and output tapes. In other words, $\mathcal{Z}$ only sees the "functionality" (or, the input/output behavior) of the given protocol. The communication is treated as being "internal" to the protocol and not part of its functionality.

[2]In the bare model we do not make any restrictions on the delivered messages. In particular, they need not be related to any of the messages generated by the parties.

Nonetheless, the order of events allows $\mathcal{Z}$ and $\mathcal{A}$ to exchange information freely (using their input and output tapes) between each two activations of some party. It may appear at first glance that no generality is lost by assuming that $\mathcal{A}$ and $\mathcal{Z}$ disclose their entire internal states to each other. A close look shows that, while no generality is lost by assuming that $\mathcal{A}$ reveals its entire state to $\mathcal{Z}$, the interesting cases occur when $\mathcal{Z}$ holds some "secret" information back from $\mathcal{A}$ and tests whether the information received from $\mathcal{A}$ is correlated with the "secret" information.

Another point to be highlighted is that the only external input to the process of protocol execution is the input of $\mathcal{Z}$. This input represents an initial state of the system and in particular includes the inputs of all parties. (From a complexity-theoretic point of view, providing the environment with arbitrary input is equivalent to stating that the environment it a non-uniform ITM.)

The ideal process. Security of protocols is defined via comparing the protocol execution in the real-life model to an *ideal process* for carrying out the task at hand. A key ingredient in the ideal process is the ideal functionality that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM that interacts with the environment and the adversary via a process described below. More specifically, the ideal process involves an ideal functionality $\mathcal{F}$, an ideal process adversary $\mathcal{S}$, an environment $\mathcal{Z}$ with input z, and a set of dummy parties $\tilde{P}_1, ..., \tilde{P}_n$.

As in the process of protocol execution in the real-life model, the environment is activated first. As there, in each activation it may read the contents of the output tapes of all (dummy) parties, and may write information on the input tape of either one of the (dummy) parties or of the adversary. Once the activation of the environment is complete the entity whose input tape was written on is activated next.

The dummy parties are fixed and simple ITMs: Whenever a dummy party is activated with input x, it forwards x to the ideal functionality $\mathcal{F}$, say by writing x on the incoming communication tape of $\mathcal{F}$. In this case $\mathcal{F}$ is activated next. Whenever a dummy party is activated due to delivery of some message it copies this message to its output. In this case $\mathcal{Z}$ is activated next.

Once $\mathcal{F}$ is activated, it reads the contents of its incoming communication tape, and potentially sends messages to the parties and to the adversary by writing these messages on its outgoing communication tape. Once the activation of $\mathcal{F}$ is complete, the entity that was last activated before $\mathcal{F}$ is activated again.

Once the adversary $\mathcal{S}$ is activated, it may read its own input tape and in addition it can read the *destinations* of the messages on the outgoing communication tape of $\mathcal{F}$. That is, $\mathcal{S}$ can see the identity of the recipient of each message sent by $\mathcal{F}$, but it cannot see the *contents* of this message (unless the recipient of the message is $\mathcal{S}$). $\mathcal{S}$ may either deliver a message from $\mathcal{F}$ to some party by having this message copied the party's incoming communication tape or corrupt a party.[3] As usual, upon corrupting a party, the adversary gains access to all the tapes of that party and controls all the party's future actions. If the adversary delivered a message to some uncorrupted (dummy) party in an activation then this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

As in the real-life model, the protocol execution ends when the environment completes an activation without writing on the input tape of any entity. The output of the protocol execution is the (one bit) output of $\mathcal{Z}$.

[3]It is stressed that in the ideal process the delivery of messages from $\mathcal{F}$ to the parties is ideally authentic and secret. The adversary only learns that a message was delivered.

Securely realizing an ideal functionality. We say that a protocol ρ securely realizes an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running ρ in the real-life process, or it is interacting with $\mathcal{S}$ and $\mathcal{F}$ in the ideal process. This means that, from the point of view of the environment, running protocol ρ is 'just as good' as interacting with an ideal process for $\mathcal{F}$. (In a way, $\mathcal{Z}$ serves as an "interactive distinguisher" between the two processes. Here it is important that $\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.) That is, we have:

Definition (securely realizing functionalities, informal statement): *A protocol ρ* securely realizes *an ideal functionality $\mathcal{F}$ is for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that, for any environment $\mathcal{Z}$ and on any input, the probability that $\mathcal{Z}$ outputs 1 after interacting with $\mathcal{A}$ and parties running ρ in the real-life model differs by at most a negligible fraction from the probability that $\mathcal{Z}$ outputs 1 after interacting with $\mathcal{S}$ and $\mathcal{F}$ in the ideal process.*

Sometimes we will want to restrict the definition so that it quantifies only over adversaries from a certain class (say, the class of adversaries that corrupt only a certain number of parties.) This is done in a straightforward way (see within).

Remark: On the requirement to generate output. Recall that the ideal process does not require the ideal-process adversary to deliver messages that are sent by the ideal functionality to the dummy parties. Consequently, the definition provides no guarantee that a protocol will ever generate output or "return" to the calling protocol. (In fact, a protocol that "hangs," never sends any messages and never generates output securely realizes any ideal functionality.) Indeed, in our setting where message delivery is not guaranteed it is impossible to require that a protocol "terminates", or generates output. Instead, the definition concentrates on the security requirements *in the case that the protocol generates output.*

In cases where message delivery is guaranteed (such as in synchronous networks) the ideal process can be modified to exclude protocols that do not generate output. See more details in Section 6.

Remark: Protecting individual sessions. A practical concern regarding protocol security is that security of a given protocol execution (or, a session) within some host should be as independent as possible from the security (or, insecurity) of other sessions that are running on the same host. The above definition answers this concern in a strong sense. Recall that our formalization postulates that the inputs to all the parties are generated by an adversarial entity (the environment), and all outputs are read by that entity. This means that a session that runs within a host (or within several hosts in a network) does not trust other protocols running within that host. In essence, the host itself is treated an adversarial entity. Consequently, security is guaranteed on a *session by session* basis: The security of a session is guaranteed, according to the specification, even if all other sessions that are running within the relevant hosts are compromised or adversarially controlled.

2.2 On the composition theorem

The hybrid model. In order to state the composition theorem, and in particular in order to formalize the notion of a real-life protocol with access to multiple copies of an ideal functionality, the hybrid model of computation with access to an ideal functionality $\mathcal{F}$ (or, in short, the $\mathcal{F}$-hybrid model) is formulated. This model is identical to the real-life model, with the following additions. On top of sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID); all messages addressed to this copy and all message sent by this copy carry the corresponding SID.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, once a party sends a message m to a copy of $\mathcal{F}$ with SID s, that copy is immediately activated to receive this message. (If no such copy of $\mathcal{F}$ exists then a new copy of $\mathcal{F}$ is created and immediately activated to receive m.) Furthermore, although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{F}$ to the parties, it does not have access to the contents of these messages. It is stressed that the environment does not have direct access to the copies of $\mathcal{F}$.

Note that the hybrid model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol in the hybrid model. (This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks. See more discussion on the role of the SID below.)

Replacing a call to $\mathcal{F}$ with a protocol invocation. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ (with respect to some class of adversaries). The composed protocol π^ρ is constructed by modifying the code of each ITM in π so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of ρ with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message given to ρ as new input. Each output value generated by a copy of ρ is treated as a message received from the corresponding copy of $\mathcal{F}$.

If protocol ρ is a protocol in the real-life model then so is π^ρ. If ρ is a protocol in some $\mathcal{G}$-hybrid model (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is π^ρ.

Theorem statement. In its general form, the composition theorem basically says that if ρ securely realizes $\mathcal{F}$ then an execution of the composed protocol π^ρ "emulates" an execution of protocol π in the $\mathcal{F}$-hybrid model. That is, for any real-life adversary $\mathcal{A}$ there exists an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and π^ρ in the real-life model or it is interacting with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model.

A corollary of the general theorem states that if π securely realizes some functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, and ρ securely realizes $\mathcal{F}$ in the real-life model, then π^ρ securely realizes $\mathcal{G}$ in the real-life model.(Here one has to define what it means to securely realize functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. This is done in the natural way.) That is:

Theorem (universal composition, informal statement): *Let ρ be a protocol that securely realizes some ideal functionality $\mathcal{F}$, and let π be a protocol in the $\mathcal{F}$-hybrid model. Then for any real-life adversary $\mathcal{A}$ there exists an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no environment machine can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π^ρ in the plain real-life model, or with $\mathcal{H}$ and parties running π in the $\mathcal{F}$-hybrid model. In particular,* *if π securely realizes some ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model then π^ρ securely realizes $\mathcal{G}$ from scratch.*

Discussion. We discuss several aspects of the hybrid model and the composition theorem. Additional discussion appears in Section 5.3.

An (over)simplification of the hybrid model. It may seem sufficient at first to formulate a simplified version of the hybrid model and the composition theorem, where only a single copy of the ideal functionality $\mathcal{F}$ is available (instead of multiple copies, as defined above). One could then handle protocols that use multiple copies of $\mathcal{F}$ by repeated applications of the composition theorem until all copies of $\mathcal{F}$ are replaced by calls to a protocol ρ that securely realizes $\mathcal{F}$. This would work even in the concurrent case, since the simplified hybrid model would allow the evaluation of $\mathcal{F}$ to run concurrently with the calling protocol, π.

However, in our setting the composition theorem can be safely repeated only a *constant* number of times. This is so, since the complexity of the hybrid model adversary $\mathcal{H}$ may depend on (in fact, it can be any polynomial in) the complexity of $\mathcal{A}$. Thus, if the theorem is applied a non-constant number of times, the resulting $\mathcal{H}$ may not be polynomial in the security parameter. Consequently, the above simplified hybrid model would result in a composition theorem that applies only to the limited case where at most a constant number of concurrent executions of a protocol take place. In contrast, the composition theorem here handles polynomially many concurrent executions, and even cases where the number of concurrent executions is determined by the adversary in an adaptive manner. In addition, the present hybrid model explicitly addresses some important issues associated with running many protocol-copies concurrently. One such issue it the handling of the session IDs, which is discussed next.

On the role of the session IDs. The composition theorem discusses “composed protocols,” where a single party (i.e., an ITM) runs multiple protocols (or, protocol-copies) concurrently. Full description of how this is done is postponed to subsequent sections. Nonetheless, let us highlight here one relevant issue: a party that runs several protocol-copies at the same time must have a mechanism by which it can tell, upon receiving a message, to which protocol-copy the message is addressed. The mechanism used here is the session IDs (SIDs). That is, it is assumed that a party that runs a composed protocol has some “operating system code” that makes sure that each protocol-copy run by the party has a *unique* SID. (It is stressed that the uniqueness is only guaranteed locally within the party, not globally in the system.) Furthermore, it is assumed that each incoming message contains an SID field. The message (together with the SID) is then directed to the corresponding protocol copy. If no such protocol copy exists then a new one is invoked and immediately activated to receive the incoming message.

The SID mechanism is reflected in the hybrid model by requiring the protocol (in the hybrid model) to include an SID in each message sent to the ideal functionality, and guaranteeing that all such messages that carry the same SID are received by the same copy of the ideal functionality. This provides the necessary “groundwork” for the composition operation (where each message x to the ideal functionality is replaced with an activation of the corresponding protocol copy with input value x).

Capturing standard concurrent composition. The traditional notion of concurrent composition of protocols usually considers a system where many identical copies of a given protocol ρ are running concurrently on adversarially controlled inputs and with an adversarially controlled scheduling of message delivery. (Here there is no "high level protocol" that invokes the various copies.) To see how this traditional notion of concurrent composition is captured by the above composition theorem, assume that protocol ρ securely realizes functionality $\mathcal{F}$, and consider the following protocol $\tilde{\pi}$ in the $\mathcal{F}$-hybrid model. Whenever a party P receives a message saying "Please activate copy s of $\mathcal{F}$ with input x," party P sends the message x to the copy of $\mathcal{F}$ having SID s.

Notice that the composed protocol, $\tilde{\pi}^{\rho}$, allows the adversary to create a scenario that is equivalent to the traditional scenario of adversarially controlled concurrent composition. Thus the composition theorem guarantees "secure concurrent composition" by stating that running protocol $\tilde{\pi}^{\rho}$ in the real-life model is essentially equivalent to running $\tilde{\pi}$ in the $\mathcal{F}$-hybrid model.

We stress that in out setting the SID is always part of the input to each copy of $\mathcal{F}$ (resp., of ρ). In addition, protocol π is assumed to be such that no two copies of $\mathcal{F}$ (resp., no two copies of ρ within the same party) have the same SID. It is interesting to note that if these conditions do not hold then the composition theorem does not necessarily hold. This fact is exemplified in [LLR01] for the basic task of broadcast (or, Byzantine agreement).

The hybrid model as the only model of computation. It will often be convenient, from a technical point of view, to concentrate on the hybrid model and treat the real-life model and the ideal process as special cases of that model. Indeed, when no ideal functionality is present then the hybrid model becomes identical to the real-life model. Similarly, when there is only one copy of the functionality and the parties behave like the "dummy parties" of the ideal process then the hybrid model becomes identical to the ideal process.

On the proof of the composition theorem. We briefly outline the main ideas in the proof of the composition theorem. Let $\mathcal{F}$ be an ideal functionality, let π be a protocol in the $\mathcal{F}$-hybrid model, let ρ be a protocol that securely realizes $\mathcal{F}$ and let π^{ρ} be the composed protocol. Let $\mathcal{A}$ be a real-life adversary, geared towards interacting with parties running π^{ρ}. We wish to construct an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no $\mathcal{Z}$ will be able to tell whether it is interacting with π^{ρ} and $\mathcal{A}$ in the real-life model or with π and $\mathcal{H}$ in the $\mathcal{F}$-hybrid model. For this purpose, we are given an ideal-process adversary $\mathcal{S}_{\rho}$ that works for a single execution of protocol ρ as stand-alone. Essentially, $\mathcal{H}$ will run a simulated copy of $\mathcal{A}$ and will follow the instructions of $\mathcal{A}$. The interaction of $\mathcal{A}$ with the various copies of ρ will be simulated using multiple copies of $\mathcal{S}_{\rho}$. For this purpose, $\mathcal{H}$ will play the environment for the copies of $\mathcal{S}_{\rho}$. The ability of $\mathcal{H}$ to obtain timely information from the multiple copies of $\mathcal{S}_{\rho}$, by playing the environment for them, is at the crux of the proof.

The validity of the simulation is demonstrated via reduction to the validity of $\mathcal{S}_{\rho}$. Dealing with many copies of $\mathcal{S}_{\rho}$ running concurrently is done using a *hybrid argument* in which defines many hybrid executions where in each hybrid execution a different number of copies of the ideal functionality $\mathcal{F}$ are replaced with copies of ρ. This hybrid argument is made possible by the fact that $\mathcal{S}_{\rho}$ must be defined independently of the environment (this is guaranteed by the order of quantifiers), thus it remains unchanged under the various "hybrid environments".

3 Preliminaries and protocol syntax

Probability ensembles and indistinguishability. We review the definitions of probability ensembles and indistinguishability, restricted to the case of binary ensembles. A distribution ensemble $X = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ is an infinite set of probability distributions, where a distribution $X(k,a)$ is associated with each $k \in \mathbf{N}$ and $a \in \{0,1\}^*$. The ensembles considered in this work describe outputs of computations where the parameter a represents input, and the parameter k is taken to be the security parameter. Furthermore, the ensembles in this work are *binary* (i.e., they only contain distributions over $\{0, 1\}$). All complexity characteristics of our constructs are measured in terms of k.

Definition 1 *Two* binary *distribution ensembles X and Y are* indistinguishable *(written $X \approx Y$) if for any $c \in \mathbf{N}$ there exists $k_0 \in \mathbf{N}$ such that for all $k > k_0$ and for all a we have*

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

Interactive Turing machines. We adhere to the standard formalization of algorithms (or, rather, computer programs) in a communication network as interactive Turing machines [GMRa89]. A detailed exposition of interactive Turing machines, geared towards formalizing *pairs* of interacting machines, appears in [G95, Chapter 6.2.1]. For self-containment we recall the definition of [G95], adding some syntax that facilitates modeling of the multi-party setting. The definition contains details that are somewhat arbitrary, and indeed do not add much insight into the nature of the problem. Nonetheless, fixing these details is essential for clarity and unambiguity of the treatment.

Definition 2 (Based on [GMRa89, G95]) *An* interactive Turing machine (ITM) *is a Turing machine that has the following tapes:*

- *A (read only)* identity *tape.*
- *A (read only)* security parameter *tape.*
- *A (read only)* random *tape.*
- *A (read only)* input *tape and a (write once[4])* output *tape.*
- *A (read only)* incoming communication *tape and a (write once)* outgoing communication *tape.*
- *A (read and write) work tape.*
- *A (read and write) one-bit* activation *tape.*

The contents of the identity tape of an ITM M is called the identity *of M. The contents of the incoming communication tape is interpreted (using some standard encoding) to consist of a sequence of strings called* messages, *where each message has two fields: the* sender *field, which is interpreted as an identity of some ITM, followed by an arbitrary* contents *field. The contents of the outgoing communication tape is interpreted in the same way, where the first field of each message is called the* recipient *field.*

We say that M is activated *if its activation tape is set to 1. Once activated, M proceeds according to its program (i.e., its transition function) and the contents of its tapes. Once M completes its processing, it resets its activation tape to 0 and enters one of two states: either a* waiting *state, or a* halt *state. If M enters the waiting state then we say that the activation is complete and that M is waiting for the next activation. If M enters the halt state then we say that it has halted; in this case, it does nothing in all future activations. The contents of the output tape of M at the completion of an activation is called the* output *of M from this activation. The new messages that appear on the outgoing communication tape of M at the end of an activation are called the* outgoing messages *of M at this activation.*

[4]In a write-once tape each cell can be written into only once.

Throughout this work we concentrate on ITMs that operate in (strict) probabilistic polynomial time. That is, we assume that with each ITM M there is an associated constant $c \in \mathbf{N}$ such that in each activation M enters either the halt state or the waiting state within n^c computational steps, where n is the value written on the security parameter tape. Furthermore, M enters the halt state after at most n^c activations. (We use a special tape to bound the running time of an ITM since the standard convention of using the input length as a basis for measuring running time does not apply in our setting where the input length varies from activation to activation.)

Message-driven protocols. Let $n \in \mathbf{N}$. An n-party message-driven protocol is a collection of n ITMs as in Definition 2, where each ITM has unique identity. (That is, no two ITMs have the same value written on their identity tapes.) We envision that each ITM represents a program to be run on a different physical computer, or "host", in a communication network. (We often use the term party to refer to a host.) The identity of each ITM represents the identity of the host in the network, plus possibly some additional information that differentiates between different programs that run on the same host. The input and output tapes model intra-host communication with other programs that are run on the host, as well as interaction with human users. The incoming and outgoing communication tapes model messages from and to other parties in the network, respectively.

Running several sub-protocols simultaneously. In subsequent sections we discuss protocols that run other protocols as "subroutines". That is, we say that some protocol π "invokes protocol ρ". This natural operation implicitly assumes that parties are running several protocols (or *sub-protocols*) simultaneously. There are a number of ways to realize this high-level description of protocols on an actual computer (or, ITM); let us specify one such way for sake of concreteness. The code of the ITM M run by each party is assumed to be structured as follows. There is a "protocol manager" code (representing the "operating system" of that party) and in addition code for each protocol to be run by the party. The protocol manager keeps track of all the protocol copies that are currently running (or, active). Initially there is a single protocol-copy of a "main" protocol that is active. Whenever an existing protocol-copy invokes a new protocol-copy, the protocol manager adds the new protocol-copy to the list of currently active copies. Each active protocol-copy has to have an identifier associated with it. The identifier, called a session ID (SID) can be decided upon in a variety of ways (say, by the session manager, or by another protocol-copy). In any case, it is assumed that the protocol manager guarantees that no two protocol-copies running within the party have the same SID. (For simplicity we assume that SIDs are never repeated for the entire execution of the party, even if the corresponding protocol has already terminated.

Each input and each incoming message to M is assumed to include the name of some protocol and SID of some protocol-copy. When M is activated with input or incoming message x that specifies some protocol π and SID s, the protocol manager first checks if there is an active copy of protocol π with SID s. If such a protocol-copy exists then the protocol manager activates this protocol-copy with input or incoming message (s, x). If no such protocol-copy exists then the protocol manager invokes a fresh copy of protocol π, gives it SID s, and immediately activates it with input or incoming message (s, x).

3.1 Discussion

We motivate some of our definitional choices regarding the syntax of ITMs and protocols, and expand on how this syntax is used in subsequent sections.

Motivating the use of ITMs. Interactive Turing machines are only one of several standard mathematical models aimed at capturing computers in a network (or, more abstractly, interacting computing agents). Other models include the π-calculus of Milner, Parrow and Walker [MPW92] (that is based on the λ-calculus as its basic model of computation), the *spi*-calculus of Abadi and Gordon [AG97] (that is based on π-calculus), the framework of Lincoln et. al. [LMMS98] (that is based on the functional representation of probabilistic polynomial time in [MMS98]), the I/O automata of Lynch [L96], and more. Several reasons motivate the decision to use ITMs as a basis for our framework. First and foremost, ITMs seem to best represent the subtle relationships between communication (often with adversarial scheduling) and the complexity of local computations (which may be randomized). These relationships are at the essence of cryptographic protocols. Furthermore, ITMs seem to best mesh with standard models used in complexity theory (such as standard Turing machines, oracle machines, and circuit families). Lastly, ITMs seem to most faithfully represent the way in which existing computers operate in a network. This faithful representation includes the separation between communication and local inputs/outputs, the identities of parties, the use of a small number of physical communication channels to interact with a large number of other parties, and more.

A disadvantage of the ITM model is that it provides only a very low-level "programming language" for representing computer programs and protocols. Consequently, practically all existing descriptions of protocols are only high-level and informal. One way to mitigate this disadvantage is to develop a library of subroutines that will allow for more convenient representation of protocols as ITMs.

On the role of the communication tapes. In the case of ITMs defined for the purpose of interactive proof-systems [GMRa89, G95], the main motivation for distinguishing between the communication tapes and the input/output tapes is that the input/output tapes are written only once throughout the execution, whereas the communication tapes keep being overwritten. Here we will not assume that the input remains the same across all activations of the machine. In fact, it will be expected that the contents of the input and output tapes will be different in different activations, and that the output tape will be read (by other ITMs) after each activation. In light of this difference, it may appear that the distinction between communication tapes and input/output tapes is no longer necessary. Nonetheless, keeping the distinction seems to be quite convenient. As seen below, the different tapes represent different types of communication of the ITM with the "outside world". Furthermore, our model allows different adversarial entities to have access to different tapes. Specifically, the environment machine will have access to the input and output tapes (which represent the local inputs and outputs of a program), while the adversary will have access to the communication tapes (which represent messages sent to and received from the network).

ITMs as circuit families. ITMs can be equivalently represented via circuit families. Here an evaluation of the circuit corresponds to a *single activation* of the corresponding ITM. That is, the input lines to the circuit represent (in some predefined way) the internal state of the ITM at the beginning of an activation, plus the contents of the incoming communication tape. The output lines of the circuit represent the internal state of the ITM at the end of the activation, plus the contents of the outgoing communication tape. We usually think of ITMs as uniform-complexity ones, thus restricting attention to uniformly generated circuit families. However, non-uniform circuit families are possible as well (and are often used to model adversarial entities).

Protocols with variable number of parties. The above formalization of message-driven protocols is apparently restricted in that it fixes in advance some bound, n, on the number of participants in the protocol. An alternative formalization would allow several copies of a given ITM, with different identities, to interact with each other; here the number of copies may grow with time (say, in an adversarially controlled way). We prefer the above formalization because of its relative simplicity, and since the difference between the two formalizations seems immaterial to the analysis of protocols. (In particular, the programs run by the parties in an n-party protocol need not be aware of n, nor do they necessarily know each other's identities. Thus n can be set to some upper bound on the number of parties expected to take place in a protocol.)

Adversaries and Ideal Functionalities as ITMs. Although the main goal in defining ITMs is to represent the programs run by the actual computers in a network, it will be convenient to use ITMs also to model adversaries and ideal functionalities. These ITMs will be somewhat different in that they will have read and write accesses to tapes that are parts of other ITMs. Also, these ITMs will not always use all their tapes. More importantly, adversarial entities are typically modeled via *non-uniform* ITMs (or, equivalently, circuit families). See more details in Section 4.

4 Defining security of protocols

This section presents a definition of protocols that securely realize a given ideal functionality, in the basic computational model (outlined in Section 2). First we present the "real-life" model of computation (Section 4.1). Next the ideal process for carrying out a given functionality is presented (Section 4.2), followed by the notion of protocol emulation and the definition of security. Section 4.4 presents some alternative formalizations of the definition and demonstrates their equivalence to the main one (Definition 3).

4.1 The real-life model of computation

Let $n \in \mathbf{N}$, and let π be an n-party message-driven protocol. Let P_i denote the identity of the ith ITM in π. (This modeling represents a set of n parties, $P_1 ... P_n$, where P_i runs the ith ITM in π. We will often fail to distinguish between the party and the ITM, or program, run by this party.) In addition, the model includes two adversarial entities, called the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. Both $\mathcal{A}$ and $\mathcal{Z}$ are programs, or ITMs, with interfaces described below. All the participants have (infinitely long) random values written on their random input tapes. In addition, the environment $\mathcal{Z}$ starts with some input value z written on its input tape. ($\mathcal{Z}$'s input represents some initial state of the environment in which the protocol execution takes place. We assume that $\mathcal{Z}$'s input represents all the external inputs to the system, including the local inputs of all parties.) We restrict attention to environments that output a single bit. (As discussed in Section 4.3, no generality is lost by this restriction.) The order of events in an execution of an n-party protocol is described in Figure 1.

We use the following notation. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol π on security parameter k, input z and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \ldots r_n$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{A}}$ for $\mathcal{A}$; r_i for party P_i). Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Protocol execution in the real-life model

Participants: Parties $P_1, ..., P_n$ running protocol π with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ on input z. All participants have the security parameter k.

1. While $\mathcal{Z}$ has not halted do:
   - (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). In addition to its own readable tapes, $\mathcal{Z}$ has read access to the output tapes of all the parties and of $\mathcal{A}$. The activation ends when $\mathcal{Z}$ enters either the halting state or the waiting state. If $\mathcal{Z}$ enters the waiting state then it is assumed to have written some arbitrary value on the input tape of exactly one entity (either $\mathcal{A}$ or of one party out of $P_1, ..., P_n$). This entity is activated next.
   - (b) Once $\mathcal{A}$ is activated, it proceeds according to its program and possibly writes new information on its output tape. In addition, $\mathcal{A}$ can perform one of the following activities:
     - i. $\mathcal{A}$ can ask to see contents of the outgoing communication tapes of all parties. The required information is then written on its incoming communication tape.
     - ii. $\mathcal{A}$ can deliver a message m to party P_i. Delivering m means writing m on the incoming message tape of P_i, together with the identity of some party P_j as the sender of this message. (There are no restrictions on the contents and sender identities of delivered messages.)
     - iii. $\mathcal{A}$ can corrupt a party P_i. Upon corruption $\mathcal{A}$ learns the sequence of all past states of P_i, and $\mathcal{Z}$ learns that P_i was corrupted. (Say, all the past states of P_i are written on $\mathcal{A}$'s input tape, and a note that P_i is corrupted is written on $\mathcal{Z}$'s input tape.)[a] Also, from this point on, P_i may no longer be activated.

     If $\mathcal{A}$ delivered a message to some party in an activation, then this party is activated once $\mathcal{A}$ enters the waiting state. Otherwise, $\mathcal{Z}$ is activated as in Step 1a.
   - (c) Once an uncorrupted party P_i is activated (either due to a new incoming message, delivered by $\mathcal{A}$, or due to a new input, generated by $\mathcal{Z}$), it proceeds according to its program and possibly writes new information on its output tape and its outgoing message tape. Once P_i enters the waiting or the halt states, $\mathcal{Z}$ is activated as in Step 1a.
2. The output of the execution is the first bit of the output tape of $\mathcal{Z}$.

[a]By postulating that the adversary learns not only the current state of the corrupted party but also all the past states, we model a system where parties are not trusted to effectively erase local data. In a setting where erasures are trusted to be effective one may let the adversary see only the *current* state of the corrupted party. See more discussion in [c00].

Figure 1: The order of events in a protocol execution in the real-life model

4.2 The ideal process

We describe the ideal process for evaluating an ideal functionality. First, however, let us define ideal functionalities.

Ideal functionalities. An ideal functionality represents the expected functionality of a certain task, or a protocol problem. This includes both "correctness", namely the expected input-output relations of uncorrupted parties, and "secrecy", or the acceptable leakage of information to the adversary. Technically, an ideal functionality $\mathcal{F}$ is modeled as an ITM (see Definition 2), with the following slight modifications. First, the input and output tapes are not used. Next, in addition to incoming messages whose sender is a party out of $P_1, ..., P_n$, $\mathcal{F}$ takes incoming messages whose sender is the adversary. Similarly, $\mathcal{F}$ may write outgoing messages whose recipient is the adversary. (Typically, useful ideal functionalities will have some more structure. To avoid cluttering the basic definition with unnecessary details, further restrictions on ideal functionalities are postponed to subsequent sections.)

The ideal process. Let $n \in \mathbf{N}$. The ideal process involves an ideal functionality $\mathcal{F}$, an environment $\mathcal{Z}$ and an ideal-process adversary $\mathcal{S}$. In addition, there are n dummy parties $\tilde{P}_1, ..., \tilde{P}_n$; these are (very simple) fixed ITMs, that only forward their inputs to $\mathcal{F}$, and output any value received from $\mathcal{F}$. As in the real-life model, we restrict attention to environments that output a single bit. The order of events in the ideal process is presented in Figure 2.

We use the following notation. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, on security parameter k, input z, and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{S}}, r_{\mathcal{F}}$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{S}}$ for $\mathcal{S}$; $r_{\mathcal{F}}$ for $\mathcal{F}$). Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

4.3 Definition of security

Roughly speaking, we say that a protocol π securely realizes an ideal functionality $\mathcal{F}$ if π emulates the ideal process for $\mathcal{F}$. More specifically, we require that for any real-life adversary $\mathcal{A}$ (from given class of adversaries) there should exist an ideal-process adversary $\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ will hold *for any environment $\mathcal{Z}$*. Note that the environment is the same in the real-life model and the ideal process. This may be interpreted as saying that "For any real-life adversary $\mathcal{A}$ in the given class, there should exist an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running π in the real-life model, or with $\mathcal{S}$ and the dummy parties in the ideal process for $\mathcal{F}$." For sake of concreteness, we restrict attention to classes of adversaries that only restrict the allowed sets of parties to be corrupted. That is, a *class* $\mathcal{C}$ is the set of adversaries that corrupt parties subject to the restriction that the identities of the parties corrupted in each execution belongs a given access structure. Security with respect to adversaries in $\mathcal{C}$ is defined as follows.

Definition 3 *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let π be an n-party protocol. Let $\mathcal{C}$ be some class of real-life adversaries. We say that π securely realizes $\mathcal{F}$ with respect to $\mathcal{C}$ if for any adversary $\mathcal{A} \in \mathcal{C}$ there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{1}$$

4.4 Alternative formulations of Definition 3

We discuss some alternative formalizations of the definition of security.

On the order of quantifiers. Consider a variant of Definition 3, where the ideal-process adversary $\mathcal{S}$ can depend on the code of the environment $\mathcal{Z}$. (That is, for any $\mathcal{A}$ and $\mathcal{Z}$ there should exist a simulator $\mathcal{S}$ that satisfies (1).) As noted in [c00], in the case of computationally unbounded environments the two formalizations are equivalent. However, the argument used there fails (and we are not aware of any other argument) in the case of polynomial-time environments and adversaries.

The ideal process

Participants: Environment $\mathcal{Z}$ and ideal-process adversary $\mathcal{S}$, interacting with ideal functionality $\mathcal{F}$ and dummy parties $\tilde{P}_1, ..., \tilde{P}_n$. All participants have the security parameter k; $\mathcal{Z}$ also has input z.

1. While $\mathcal{Z}$ has not halted do:
   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). $\mathcal{Z}$'s activity remains unchanged from the real-life model (Figure 1). In particular, $\mathcal{Z}$ cannot access $\mathcal{F}$.

   (b) Once a dummy party $\tilde{P}_i$ is activated with a new value on its input tape, it copies this value to the incoming communication tape of $\mathcal{F}$, and enters the waiting state. $\mathcal{F}$ is activated next.

   (c) Once $\mathcal{F}$ is activated it follows its program until it enters either the waiting state or the halt state. In particular, $\mathcal{F}$ may write on its outgoing communication tape messages addressed to the parties and adversary. If $\mathcal{F}$ wrote a message to the adversary then the adversary is activated next. Otherwise, the party that was last activated before $\mathcal{F}$ is activated again.

   (d) Once the adversary $\mathcal{S}$ is activated, it follows its program and possibly writes new information on its output tape. In addition, $\mathcal{S}$ can perform one of the following activities.

      i. $\mathcal{S}$ may ask to see the contents of the outgoing communication tapes of the dummy parties, and the destinations of the outgoing messages generated by $\mathcal{F}$. The desired information is then written on $\mathcal{S}$'s incoming communication tape.

      ii. $\mathcal{S}$ may deliver a message m from $\mathcal{F}$ to some dummy party $\tilde{P}_i$. Here $\mathcal{S}$ does not have access to the contents of m; it only sees the destination of m (unless $\tilde{P}_i$ is corrupted).

      iii. $\mathcal{S}$ may write a message, m, on the incoming communication tape of $\mathcal{F}$. This message appears with sender $\mathcal{S}$.

      iv. $\mathcal{S}$ may corrupt a dummy party $\tilde{P}_i$. Upon corruption, $\mathcal{S}$ learns the history of all past states of $\tilde{P}_i$, and $\mathcal{Z}$ learns that $\tilde{P}_i$ was corrupted. (P_i's history consists of the information received from $\mathcal{Z}$ and from $\mathcal{F}$ throughout the computation.) From this point on, P_i may no longer be activated; also, $\mathcal{S}$ receives all the messages from $\mathcal{F}$ that are addressed to P_i, and may deliver to $\mathcal{F}$ messages whose sender is P_i.

      If some message was delivered to $\tilde{P}_i$ (or $\mathcal{F}$) in this activation then $\tilde{P}_i$ (or $\mathcal{F}$) is activated once $\mathcal{S}$ enters the waiting state. Otherwise, $\mathcal{Z}$ is activated next.

   (e) Once a dummy party $\tilde{P}_i$ is activated with a new incoming message from $\mathcal{F}$ it copies this message to its output tape and enters the waiting state. $\mathcal{Z}$ is activated next.

2. The global output is the first bit of the output tape of $\mathcal{Z}$.

Figure 2: The order of events in the ideal process for a given ideal functionality, $\mathcal{F}$.

We remark that our proof of the universal composition theorem (Theorem 6) uses the stronger formalization in an essential way.

On environments with non-binary outputs. The models of computation and Definition 3 deal only with environments that generate binary outputs. One may consider an extension to the models where the environment has arbitrary output; here the definition of security would require that the two output ensembles $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ (that would no longer be binary) be *computationally indistinguishable,* as defined by Yao [Y82]. It is easy to see, however, that this extra generality results in a definition that is equivalent to Definition 3. We leave the proof as an exercise.

On deterministic environments. Since we allow the environment to receive an arbitrary external input, it suffices to consider only deterministic environments. (That is, the definition that quantifies only over deterministic environments is equivalent to Definition 3.)

Doing without the real-life adversary. Definition 3 can be simplified somewhat, as follows. Fix some class $\mathcal{C}$ of adversaries, and consider the following "dummy adversary", $\tilde{\mathcal{A}}_{\mathcal{C}}$. Once activated, adversary $\tilde{\mathcal{A}}_{\mathcal{C}}$ interprets the contents of its input tape (which was generated by the environment) as one of three possible instructions: either to report all the new messages sent by the parties, to deliver a given message to some party, or to corrupt some party. $\tilde{\mathcal{A}}_{\mathcal{C}}$ carries out the instruction, writes the gathered information on its output tape, and completes its activation. (If $\tilde{\mathcal{A}}_{\mathcal{C}}$ receives a corruption instruction that is inconsistent with the class $\mathcal{C}$ then it ignores this instruction.) Then, instead of quantifying over all possible adversaries $\mathcal{A}$, it suffices to require that the ideal-process adversary $\mathcal{S}$ be able to handle the dummy adversary $\tilde{\mathcal{A}}_{\mathcal{C}}$ (and any environment machine $\mathcal{Z}$). That is, we have:

Definition 4 *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let π be an n-party protocol. Let $\mathcal{C}$ be some class of real-life adversaries. We say that π* securely realizes $\mathcal{F}$ with respect to $\mathcal{C}$ *if there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\tilde{\mathcal{A}}_{\mathcal{C}},\mathcal{Z}} \tag{2}$$

Claim 5 *Definitions 3 and 4 are equivalent.*

Proof: Clearly Definition 3 implies Definition 4. The idea of the derivation in the other direction is that, given access to adversary $\tilde{\mathcal{A}}_{\mathcal{C}}$, the environment can simulate any other adversary by itself. Thus quantifying over all environments essentially implies quantifying also over all real-life adversaries. More formally, assume that there exist an ideal-process adversary $\mathcal{S}$ that satisfies (2) with respect to any environment $\mathcal{Z}$. Then, given an adversary $\mathcal{A}$ we construct an ideal-process adversary $\mathcal{S}'$ that satisfies (1) with respect to $\mathcal{A}$ and any environment. $\mathcal{S}'$ runs simulated copies of $\mathcal{A}$ and $\mathcal{S}$. In addition:

1. $\mathcal{S}'$ forwards any input from the environment to the simulated $\mathcal{A}$, and copies any output of $\mathcal{A}$ to its own output tape (where it will be read by the environment).
2. Whenever the simulated $\mathcal{A}$ expects to see new messages generated by the parties, or wishes to deliver a message or to corrupt a party, $\mathcal{S}'$ activates $\mathcal{S}$ with the corresponding request and forwards the response (i.e., the output) of $\mathcal{S}$ back to $\mathcal{A}$.
3. $\mathcal{S}'$ forwards any message coming from the ideal functionality $\mathcal{F}$ to the simulated $\mathcal{S}$, and forwards to $\mathcal{F}$ any message that $\mathcal{S}$ addresses to its ideal functionality.

Assume for contradiction that there is an adversary $\mathcal{A}$ and environment $\mathcal{Z}'$ such that $\text{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'} \not\approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}'}$. We construct an environment $\mathcal{Z}$ such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \not\approx \text{REAL}_{\pi,\tilde{\mathcal{A}}_{\mathcal{C}},\mathcal{Z}}$. Environment $\mathcal{Z}$ runs an interaction between simulated copies of $\mathcal{Z}'$ and $\mathcal{A}$. Whenever the simulated $\mathcal{A}$ expects to see the messages sent by the parties, $\mathcal{Z}'$ asks the actual adversary it interacts with to provide this information. Similarly, whenever the simulated $\mathcal{A}$ delivers a message or corrupts a party, $\mathcal{Z}$

asks the actual adversary it interacts with to carry out these instructions. $\mathcal{Z}$ outputs whatever the simulated $\mathcal{Z}'$ outputs. It is easy to see that ensembles $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'}$ are identical. Similarly, ensembles $\text{REAL}_{\pi,\tilde{\mathcal{A}}_\mathcal{C},\mathcal{Z}}$ and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}'}$ are identical. The contradiction (and the claim) follows. □

This simplified formulation of the definition can be carried a bit further by restating the real-life model of computation so that the real-life adversary is not mentioned at all. Instead, the environment can directly read the messages sent by the parties, and can deliver messages and corrupt parties by itself. (It is stressed, however, that the ideal process remains unchanged. Here it is the role of the ideal-process adversary to make the interaction indistinguishable, from the point of view of the environment, from an interaction with the real-life process.)

We find Definition 4 less intuitive than Definition 3, thus we do not present it as the main definition. Nonetheless, being less complicated (it uses one less adversarial entity and one less quantifier), it is useful. In particular, it simplifies the proof of the composition theorem. (In addition, in the hybrid model of computation, defined and discussed in the next section, this alternative formulation is no longer equivalent, in general, to Definition 3.)

5 Universal composition of secure protocols

This section states and proves the composition theorem, as sketched in Section 2. Sections 5.1 and 5.2 define the hybrid model and state the composition theorem. Section 5.3 discusses and motivates some aspects of the hybrid model and the theorem. (This discussion is in addition to the discussion in Section 2.2.) Section 5.4 presents the proof.

5.1 Preparing the ground

We "prepare the ground" for the statement of the composition theorem in three steps. First we make a technical restriction on the format of ideal functionalities to which the composition theorem applies. This restriction seems essential when discussing concurrent composition of protocols. (In fact, clear how to even state the composition theorem without it.) Next, we define the hybrid model where a protocol π has access to (many independent copies of) a given ideal functionality $\mathcal{F}$. Next, we define the "mechanics" of replacing π's calls to the ideal functionality with calls to some protocol ρ that securely realizes $\mathcal{F}$.

Session ID's. In order to be able to formalize the hybrid model and state the composition theorem, we make the following syntactic restriction on the ideal functionality. Each incoming message for $\mathcal{F}$ should have a special field, called the Session ID (SID) field. The contents of this field, the SID, should be the same in all incoming messages to a given invocation of $\mathcal{F}$. That is, any incoming message whose SID field is different than that of the first message in an invocation is ignored. Furthermore, all outgoing messages generated by $\mathcal{F}$ have an SID field whose contents is identical to the SID field of the first incoming message. For a string $id \in \{0,1\}^*$ we let $\mathcal{F}_{(id)}$ denote the copy of $\mathcal{F}$ whose SID is id.

This restriction allows us to distinguish between different copies (i.e., invocations) of the ideal functionality that are running at the same time. It is also guarantees that protocols that realize these functionalities will have similar behavior: they will expect an SID in each input, and will include the SID in each output. Note that these conventions are compatible with the basic structure of "composed protocols" (i.e., protocols that consist of several protocol-copies that are in execution at the same time), as described in Section 3.

The hybrid model. We specify the real-life model of computation with the assistance of some ideal functionality $\mathcal{F}$. This model, called the hybrid model with ideal access to $\mathcal{F}$ (or in short the $\mathcal{F}$-hybrid model), is obtained as follows. We start with the real-life model of Section 4.1. To this model we add an unlimited number of copies of the ideal functionality $\mathcal{F}$. In addition to the activities specified in the real-life model, in each activation the parties and adversary may send messages to and receive messages from the various copies of $\mathcal{F}$. The communication between the parties and each copy of $\mathcal{F}$ mimics the ideal process. That is, messages addressed to some copy of $\mathcal{F}$ with SID id (i.e., to $\mathcal{F}_{(id)}$) are written on the incoming communication tape of that copy. If no existing copy of $\mathcal{F}$ has SID id then a new copy of $\mathcal{F}$ is invoked and the message is being written on the incoming communication tape of that copy. Outgoing messages generated by each copy of $\mathcal{F}$ are delivered by the adversary $\mathcal{H}$, without allowing $\mathcal{H}$ access to the contents of the delivered messages. (as usual, $\mathcal{H}$ may choose to deliver these messages at any time during the computation, or even to not deliver these messages at all.)

The order of events in a protocol execution in the hybrid model is described in Figure 3. It is stressed that each copy of $\mathcal{F}$ is a separate ITM with independent random input; furthermore, it does not send messages to other copies of $\mathcal{F}$. (Indeed, $\mathcal{F}$ is defined in a stand-alone ideal process where no other copies exist.)

Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}(k,z)$ denote the random variable describing the output of environment machine $\mathcal{Z}$ on input z, after interacting in the $\mathcal{F}$-hybrid model with protocol π, adversary $\mathcal{H}$, analogously to the definition of $\text{REAL}_{\pi,\mathcal{H},\mathcal{Z}}(k,z)$ in Section 4. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to $\mathcal{F}$.) Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},Z}$ denote the distribution ensemble $\{\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},Z}\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Replacing a call to an ideal functionality with a subroutine call. Next we describe how a call (made by some party running π in the $\mathcal{F}$-hybrid model) to some copy of the ideal functionality $\mathcal{F}$ is replaced with an activation of a message-driven protocol ρ (that presumably realizes $\mathcal{F}$). This is done in a straightforward way. That is, the description of π is modified as follows, to obtain a protocol π^ρ: (The following are protocol instructions, to be carried out by the uncorrupted parties.)

1. Whenever π instructs party P_i to send a message x to $\mathcal{F}_{(id)}$, protocol π^ρ instructs P_i to proceed as follows.
   (a) If this is the first message sent by P_i to $\mathcal{F}_{(id)}$ then invoke a new copy of protocol ρ with SID id, and immediately activate this copy with incoming message x and uniformly distributed string for its random input. That is, a new copy of the ith ITM in ρ is invoked as a subroutine of P_i, and is (locally) given identity id. (See Section 3.1 for more low-level details on how a new protocol invocation runs side-by-side with a an existing invocation.) We say that the new copy of ρ, denoted $\rho_{i,(id)}$, is associated with $\mathcal{F}_{(id)}$.
   (b) Otherwise, hand x to the already-existing copy $\rho_{i,(id)}$.
2. Whenever the copy $\rho_{i,(id)}$ wishes to send a message m to some party $P_{i'}$, write the message $(m,\rho,(id))$ on the outgoing communication tape. Messages that are generated by $\rho_{i,(id)}$ are called $\rho_{i,(id)}$-messages.
3. Whenever activated due to delivery of a $\rho_{i',(id)}$-message $(m,\rho,(id))$ from $P_{i'}$, check whether there is a running copy of ρ that is associated with $\mathcal{F}_{(id)}$. If such a copy, $\rho_{i,(id)}$, exists, then activate this copy with incoming message m. Otherwise, invoke a new copy of ρ with local SID (id) and activate it with incoming message m.

Protocol execution in the $\mathcal{F}$-hybrid model

Participants: Parties $P_1, ..., P_n$ running protocol π with multiple copies of an ideal functionality $\mathcal{F}$, with adversary $\mathcal{H}$, and with environment $\mathcal{Z}$ on input z. All participants have the security parameter k.

1. While $\mathcal{Z}$ has not halted do:
   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). $\mathcal{Z}$'s activity remains unchanged from the real-life model (Figure 1). In particular, $\mathcal{Z}$ cannot access any copy of $\mathcal{F}$.

   (b) Once the adversary $\mathcal{H}$ is activated, it proceeds according to its program and possibly writes new information on its output tape. In addition, $\mathcal{H}$ can perform one out of the following activities:
      i. $\mathcal{H}$ can ask to see contents of the outgoing communication tapes of all parties. The required information is then written on its incoming communication tape, with the exception that for the messages generated by the copies of $\mathcal{F}$ only the *recipient identities* appear.
      ii. $\mathcal{H}$ can deliver a message m to party P_i. This activity remains unchanged from the real-life model (Figure 1).
      iii. $\mathcal{H}$ can deliver a message m from some copy of $\mathcal{F}$ to party P_i. This activity remains unchanged from the ideal process (Figure 2), with the exception that here these messages are written on a distinguished segment of the incoming communication tape of P_i.
      iv. $\mathcal{H}$ can deliver a message, m, to some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$. This message appears with sender $\mathcal{H}$.
      v. $\mathcal{H}$ can corrupt a party P_i. This activity remains unchanged from the real-life model (Figure 1).

      If some message was delivered to P_i (or some $\mathcal{F}_{(id)}$) in this activation then P_i (or $\mathcal{F}_{(id)}$) is activated once $\mathcal{H}$ enters either the waiting state. Otherwise, the entity that was last active before $\mathcal{H}$ is activated again. (This entity is either $\mathcal{Z}$ or $\mathcal{F}$.)

   (c) Once an uncorrupted party P_i is activated (either due to a new input generated by $\mathcal{Z}$, or due to a message delivered by $\mathcal{H}$), it proceeds according to its program and possibly writes new information on its output tape and new messages on its outgoing communication tape. In addition, in each activation P_i may write a single message to a single copy of $\mathcal{F}$. If such message is written, then that copy of $\mathcal{F}$ is activated next. Otherwise, $\mathcal{Z}$ is activated next.

   (d) Once some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$ is activated it proceeds as in the ideal process (Figure 2).
2. The output of the execution is the (one bit) output of $\mathcal{Z}$.

Figure 3: The order of events in a protocol execution in the hybrid model

4. Whenever a copy $\rho_{i,(id)}$ generates an output value y, proceed just as π proceeds with incoming message y from $\mathcal{F}_{(id)}$.

5.2 The composition theorem

We are now ready to state the composition theorem. First we state a more general theorem that makes roughly the following statement: Let π be an arbitrary protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ as in Definition 3. Then the protocol π^ρ, running in the real-life model, has "essentially the same behavior" as protocol π. That is, for any real-life adversary $\mathcal{A}$ there exists a hybrid-model adversary $\mathcal{H}$ such that no environment machine $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and π^ρ in the real-life model or with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model. The statement of the theorem follows.

Theorem 6 (Universal composition: General statement) *Let $\mathcal{C}$ be a class of real-life adversaries, and let $\mathcal{F}$ be an ideal functionality. Let π be an n-party protocol in the $\mathcal{F}$-hybrid model, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ with respect to $\mathcal{C}$. Then for any real-life adversary $\mathcal{A} \in \mathcal{C}$ there exists a hybrid-model adversary $\mathcal{H} \in \mathcal{C}$ such that for any environment machine $\mathcal{Z}$ we have:*

$$\text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}. \tag{3}$$

The second step, which is a corollary from the first one, concentrates on protocols π that securely realize some given functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. The corollary essentially states that if protocol π securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model and if ρ securely realizes $\mathcal{F}$, then π^ρ securely realizes $\mathcal{G}$. To formalize this corollary, we first define protocols for securely realizing a functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. This is done via the usual comparison to the ideal process for $\mathcal{G}$:

Definition 7 *Let $\mathcal{C}$ be a class of real-life adversaries, let π be an n-party protocol, and let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. We say that π* securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model with respect to $\mathcal{C}$ *if for any adversary $\mathcal{H} \in \mathcal{C}$ (in the $\mathcal{F}$-hybrid model) there exists an ideal-process adversary $\mathcal{S}$ such that for any environment machine $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{4}$$

Corollary 8 (Universal composition: Realizing functionalities) *Let $\mathcal{C}$ be a class of real-life adversaries, and let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. Let π be an n-party protocol that realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model with respect to $\mathcal{C}$, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ with respect to $\mathcal{C}$. Then protocol π^ρ securely realizes $\mathcal{G}$ with respect to $\mathcal{C}$.*

Proof: Let $\mathcal{A} \in \mathcal{C}$ be a real-life adversary that interacts with parties running π^ρ. Theorem 6 guarantees that there exists an adversary $\mathcal{H} \in \mathcal{C}$ in the $\mathcal{F}$-hybrid model such that $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}} \approx \text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}}$ for any environment machine $\mathcal{Z}$. The security of π in the $\mathcal{F}$-hybrid model guarantees that there exists an ideal model adversary (a "simulator") $\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}$ for any $\mathcal{Z}$. The corollary follows by combining the two equalities. □

5.3 Discussion

Some aspects of the hybrid model were discussed in Section 2.2. This section highlights and motivates some additional aspects.

Handling multiple functionalities. The hybrid model is defined with respect to (multiple copies of) a *single* functionality, $\mathcal{F}$. One can easily generalize the definition to allow ideal calls to several different ideal functionalities. We chose not to do so for sake of simplicity, and since the more general formalization hardly adds any power to the model: One can always define a single functionality that mimics several different ones, say via a "universal functionality".

Invoking several subroutines in a single activation. The hybrid model specifies that a party may invoke only a single copy of $\mathcal{F}$ in an activation. This may seem somewhat restrictive since protocols may wish to call several copies of $\mathcal{F}$ in a single activation. Recall, however, that whenever some party P_i activates a copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$, then P_i is immediately re-activated once $\mathcal{F}_{(id)}$ completes its activation. This provision allows P_i to activate a number of copies of $\mathcal{F}$, one after the other, without interference of either $\mathcal{H}$ or $\mathcal{Z}$.

Nesting of protocol invocations. Theorem 6 is stated for the case where the "subroutine protocol" (i.e., the protocol ρ that securely realizes $\mathcal{F}$) is a protocol in the real-life model. The theorem can be easily extended to deal also with the case where ρ runs in a hybrid model with ideal access to some functionality $\mathcal{F}'$. Here the composed protocol, π^ρ, is also a protocol in the $\mathcal{F}'$-hybrid model. This extension allows us to apply the theorem iteratively to protocols with multiple "nesting". (For instance, if π is a protocol that securely realizes functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, protocol ρ securely realizes $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model, and protocol ρ' securely realizes $\mathcal{F}'$ in the real-life model, then we obtain a protocol $\pi^{\rho^{\rho'}}$ that securely realizes $\mathcal{G}$ in the real-life model.)

Nonetheless, two caveats need to be stressed here. First, note that this "nesting" of functionalities can have only constant depth. If the composition theorem is iterated more than a constant number of times then the complexity of the hybrid-model adversary guaranteed by the theorem may no longer be bounded by a polynomial in the security parameter.

Second, it is stressed that in protocol $\pi^{\rho^{\rho'}}$ no two copies of ρ may ever call the same copy of ρ', otherwise the composition theorem no longer holds. We note that this restriction may result in inefficiencies and duplications. One way to circumvent this restriction (i.e., to have several copies of ρ use the same copy of ρ') is to define a functionality $\hat{F}$ that captures multiple copies of $\mathcal{F}$. Then one can design a protocol $\hat{\rho}$ that realizes $\hat{\mathcal{F}}$ in the $\mathcal{F}'$-hybrid model. (That is, a single copy of $\hat{\rho}$ would have the same functionality as multiple copies of ρ.) Now, protocol $\pi^{\rho^{\rho'}}$ can be replaced by $\pi^{\hat{\rho}^{\rho'}}$, where protocol $\hat{\rho}^{\rho'}$ is called only once and can use only a single, or small number of copies of ρ'. An example of a context where this technique is used is the case of commitment schemes in the common random string model. See details in Section 8.3.2 and in [CF01]. (There, functionality $\mathcal{F}'$ corresponds to the common randomness functionality and functionality $\mathcal{F}$ corresponds to the commitment functionality.)

On Definition 4 in the hybrid model. Recall that Definition 4 is a somewhat simplified formulation of the main definition (Definition 3: securely realizing a given functionality in the real-life model of computation). Specifically, instead of quantifying over all real-life adversaries, Definition 4 considers only a single "generic" adversary, called the *dummy adversary.* It is then shown that Definition 4 and Definition 3 are equivalent.

It is interesting to note that this equivalence does not hold, in general, with respect to Definition 7 (i.e., with respect to securely realizing an ideal functionality in the $\mathcal{F}$-hybrid model for some $\mathcal{F}$). This is so since in the $\mathcal{F}$-hybrid model the "interface" of the adversary is more complicated than in the real-life model. This is so since the adversary in the $\mathcal{F}$-hybrid model may receive information from $\mathcal{F}$ and may have the opportunity to send information to $\mathcal{F}$. These activities are not captured by the simplistic dummy adversary of Definition 4. (Still, for some specific $\mathcal{F}$'s it may be possible to simplify Definition 7 along the lines of Definition 4 by considering only a restricted class of adversaries in the $\mathcal{F}$-hybrid model.)

5.4 Proof of the composition theorem

A high-level sketch of the proof was presented in in section 2. Section 5.4.1 contains an outline of the proof. A detailed proof appears in Section 5.4.2.

5.4.1 Proof outline

The proof uses the alternative formulation of the definition of security (Definition 4). This formulation considerably simplifies the presentation of the proof. Let $\mathcal{F}$ be an ideal functionality, let π be a protocol in the $\mathcal{F}$-hybrid model, let ρ be a protocol that securely realizes f, let π^ρ be the composed protocol, and let $\mathcal{C}$ be the given class of adversaries (i.e., $\mathcal{C}$ is an access structure for the corruption of parties). Recall the dummy adversary $\tilde{\mathcal{A}}_{\mathcal{C}}$ from Definition 4. We wish to construct an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no $\mathcal{Z}$ will be able to tell whether it is interacting with π^ρ and $\tilde{\mathcal{A}}_{\mathcal{C}}$ in the real-life model or with π and $\mathcal{H}$ in the $\mathcal{F}$-hybrid model. That is, for any $\mathcal{Z}$, $\mathcal{H}$ should satisfy

$$\text{REAL}_{\pi^\rho,\tilde{\mathcal{A}}_{\mathcal{C}},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}. \tag{5}$$

The general outline of the proof proceeds as follows. The security of ρ guarantees that there exists a simulator (i.e., an ideal-process adversary), $\mathcal{S}$, such that for any environment $\mathcal{Z}_\rho$ we have:

$$\text{REAL}_{\rho,\tilde{\mathcal{A}}_{\mathcal{C}},\mathcal{Z}_\rho} \approx \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho}. \tag{6}$$

We construct out of $\mathcal{S}$ an adversary, $\mathcal{H}$, that interacts with some environment $\mathcal{Z}$ and parties running protocol π in the $\mathcal{F}$-hybrid model. We then demonstrate that $\mathcal{H}$ satisfies (5). This is done by reduction: Given an environment $\mathcal{Z}$ that violates (5), construct an environment $\mathcal{Z}_\rho$ that violates (6).

Adversary $\mathcal{H}$ operates as follows. In a nutshell, $\mathcal{H}$ mimics $\tilde{\mathcal{A}}_{\mathcal{C}}$ by following the requests of $\mathcal{Z}$ with respect to the execution of π, and handing the requests of $\mathcal{Z}$ with respect to the copies of ρ to corresponding copies of the ideal-process adversary $\mathcal{S}$. More specifically, recall that $\tilde{\mathcal{A}}_{\mathcal{C}}$ received three types of requests from $\mathcal{Z}$: Either to report the new messages generated by the parties, or to deliver a message, or to corrupt a party. To mimic the behavior of $\tilde{\mathcal{A}}_{\mathcal{C}}$, $\mathcal{H}$ runs copies of the simulator $\mathcal{S}$ that correspond to the current copies of $\mathcal{F}$ in the actual interaction in the $\mathcal{F}$-hybrid model. In response to a request to report the new messages generated by the parties, $\mathcal{H}$ reports the messages that the actual parties generate in their execution of protocol π, and in addition $\mathcal{H}$ uses the copies of $\mathcal{S}$ to simulate the messages generated from the currently active copies of ρ. In response to a request to deliver a π-message (i.e., a message addressed to protocol π within the recipient party), $\mathcal{H}$ delivers this message to the actual party it interacts with. A request to deliver a message addressed to some copy of ρ is forwarded to the relevant copy of $\mathcal{S}$. In response to a request to corrupt a party P_i, $\mathcal{H}$ corrupts the actual P_i it interacts with, and in addition instructs all copies of $\mathcal{S}$ to corrupt P_i. It then combines the gathered information to constitute an internal state of P_i for protocol π^ρ. (Here the internal state from π is real and the internal state from the copies of ρ is simulated.) In addition, $\mathcal{H}$ relays without interference the communication between the simulated copies of $\mathcal{S}$ and the corresponding actual copies of $\mathcal{F}$ that $\mathcal{H}$ interacts with.

The validity of $\mathcal{H}$ is demonstrated, based on the validity of $\mathcal{S}$, via a *hybrid argument*. While the basic logic of the argument is standard, applying the argument to our complex setting requires some work. We sketch this argument. Let m be an upper bound on the number of copies of ρ that are invoked in this interaction. Informally, we let the $\mathcal{F}^{(l)}$-hybrid model denote the model where the interaction with the first l copies of $\mathcal{F}$ works as in the $\mathcal{F}$-hybrid model, whereas the rest of the copies of $\mathcal{F}$ are replaced with an invocation of ρ. In particular, an interaction of $\mathcal{Z}$ with $\mathcal{H}$

in the $\mathcal{F}^{(m)}$-hybrid model is essentially identical to an interaction of $\mathcal{Z}$ with $\mathcal{H}$ in the $\mathcal{F}$-hybrid model. Similarly, an interaction of $\mathcal{Z}$ with $\mathcal{H}$ in the $\mathcal{F}^{(0)}$-hybrid model is essentially identical to an interaction of $\mathcal{Z}$ with $\tilde{\mathcal{A}}_{\mathcal{C}}$ in the real-life model of computation.

Now, assume that there exists an environment $\mathcal{Z}$ that distinguishes with probability ϵ between an interaction with $\mathcal{H}$ in the $\mathcal{F}$-hybrid model and an interaction with $\tilde{\mathcal{A}}_{\mathcal{C}}$ in the real-life model. Then there is an $0 < l \leq m$ such that $\mathcal{Z}$ distinguishes between an interaction with $\mathcal{H}$ in the $\mathcal{F}^{(l)}$-hybrid model and an interaction with $\mathcal{H}$ in the $\mathcal{F}^{(l-1)}$-hybrid model. We then construct an environment $\mathcal{Z}_\rho$, that interacts with parties running a single copy of ρ, and can distinguish with probability ϵ/m between an interaction with $\tilde{\mathcal{A}}_{\mathcal{C}}$ and parties running ρ in the real-life model, and an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}$.

Essentially, $\mathcal{Z}_\rho$ runs a simulated execution of $\mathcal{Z}$, adversary $\mathcal{H}$, and parties running π^ρ in the $\mathcal{F}^{(l)}$-hybrid model, but with the following exception. $\mathcal{Z}_\rho$ uses its actual interaction (which takes place either in the ideal process for $\mathcal{F}$ or in the real-life model) to replace the parts of the simulated execution that have to do with the interaction with the lth copy of $\mathcal{F}$, denoted $\mathcal{F}_{(id_l)}$. (That is, id_l is the SID of the lth copy of $\mathcal{F}$.) A bit more specifically, whenever some simulated party P_i sends a message x to $\mathcal{F}_{(id_l)}$, $\mathcal{Z}_\rho$ activates the corresponding actual party with input x. Outputs generated by an actual party P_i are treated like messages from $\mathcal{F}_{(id_l)}$ to the simulated P_i. Whenever the simulated adversary $\mathcal{H}$ activates the copy $\mathcal{S}_{(id_l)}$ of $\mathcal{S}$ with input value v, $\mathcal{Z}_\rho$ activates the actual adversary it interacts with input v. The new value written on the output tape of the adversary that $\mathcal{Z}_\rho$ interacts with is given to the simulated $\mathcal{H}$ as an output of $\mathcal{S}_{(id_l)}$. Once the simulated $\mathcal{Z}$ halts, $\mathcal{Z}_\rho$ halts and outputs whatever $\mathcal{Z}$ outputs.

The proof is completed by observing that, if $\mathcal{Z}_\rho$ interacts with $\mathcal{S}$ in the ideal process for $\mathcal{F}$, then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ has the same distribution as the view of $\mathcal{Z}$ when interacting with $\mathcal{H}$ in the $\mathcal{F}^{(l)}$-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with $\mathcal{A}$ and parties running ρ in the real-life model, then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ has the same distribution as the view of $\mathcal{Z}$ when interacting with $\mathcal{H}$ in the $\mathcal{F}^{(l-1)}$-hybrid model.

5.4.2 A detailed proof

Terminology. We use the following terminology. An internal state of an ITM M includes the control state and the contents and head locations of all the tapes readable by M. In cases where the model allows M to read tapes that are parts of other ITMs then these tapes become part of the internal state of M. Also, for convenience we include in the internal state the contents of the entire random tape of M, including locations that were not yet read by M.

We assume an encoding convention of internal states into strings. A string $s \in \{0,1\}^*$ is said to be an internal state of machine M if s encodes some internal state of M. (Without loss of generality we can assume that any string s encodes *some* internal state.) In the sequel we often do not distinguish between internal states and their encodings.

A global state of a system is a concatenation of the internal states of the environment, the adversary, and all parties at some point during the protocol execution. In the hybrid model the global state contains also the internal state of all the copies of the ideal functionality. Note that exactly one ITM in a global state has its activation bit set. This ITM is the active entity in the global state.

An execution of a protocol (either in the real-life or in a hybrid model) is the process of running the protocol with a given environment and adversary on given inputs for the environment and random inputs for the all participants. Similarly, "running an activation of machine M from internal state s" means simulating an activation of M starting at the internal state described in s until

M enters the halting or the waiting states. Note that s contains all the information needed for the simulation; in particular, it contains all the necessary randomness. The view of M in a given execution is the sequence of all internal states of M in that execution until it halts.

Adversary $\mathcal{H}$

Adversary $\mathcal{H}$ proceeds as follows, given a value k for the security parameter, and interacting with parties $P_1, ..., P_n$ running protocol π in the $\mathcal{F}$-hybrid model, with an environment $\mathcal{Z}$.
Initially $\mathcal{H}$ keeps no copies of $\mathcal{S}$ running.

1. If the input tape contains an instruction to report to $\mathcal{Z}$ on the new messages sent by the parties, then proceed as follows:
   (a) Read the outgoing communication tapes of the parties, $P_1, ..., P_n$.
   (b) Activate each running copy of $\mathcal{S}$ with a request to report the new messages sent by the parties in the corresponding copy of ρ.
   (c) Combine the information gathered from Steps 1a an 1b to obtain a simulated report of the messages sent by parties running π^ρ, and write this report on the output tape (to be read by $\mathcal{Z}$).
2. If the input contains an instruction to deliver a π-message to some party P_i then deliver the message to P_i. If the instruction is to deliver to P_i a message from some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$ then activate the copy $\mathcal{S}_{(id)}$ of $\mathcal{S}$ with an instruction to deliver the message to P_i. (If there is no active copy of $\mathcal{S}$ with SID id then invoke a new copy of $\mathcal{S}$ and label it as $\mathcal{S}_{(id)}$.)
3. If the input contains an instruction to corrupt a party, P_i, then proceed as follows:
   (a) Corrupt the actual P_i in the hybrid model, and obtain P_i's state regarding to the execution of protocol π.
   (b) Instruct all the running copies of $\mathcal{S}$ to corrupt P_i, and obtain the the simulated internal data of P_i from all the running copies of π.
   (c) Combine the information gathered in Steps 3a and 3b to obtain a simulated state of P_i with respect to protocol π^ρ. Copy this state to the output tape.
4. If the incoming communication tape contains a new message from some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$, then activate the copy $\mathcal{S}_{(id)}$ of $\mathcal{S}$ with incoming message from its ideal functionality. (If there is no active copy of $\mathcal{S}$ with SID id then invoke a new copy of $\mathcal{S}$ and label it as $\mathcal{S}_{(id)}$.)
5. If in any one of its activations some copy $\mathcal{S}_{(id)}$ of $\mathcal{S}$ delivers a message from its ideal functionality to some simulated party P_i then deliver the corresponding message from $\mathcal{F}_{(id)}$ to the actual party P_i. Similarly, if $\mathcal{S}_{(id)}$ sends a message to its ideal functionality, then forward this message to $\mathcal{F}_{(id)}$.

Figure 4: The adversary for protocol π in the $\mathcal{F}$-hybrid model.

Construction of $\mathcal{H}$. Let $\mathcal{C}$ be a class of adversaries (that specifies the allowable sets of parties to be corrupted). Let $\mathcal{F}$ be an ideal functionality, let π be an n-party protocol in the $\mathcal{F}$-hybrid model, let ρ be a protocol that securely realizes f, and let π^ρ be the composed protocol. The security of ρ guarantees that there exists an ideal-process adversary $\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho} \approx \text{REAL}_{\rho,\tilde{\mathcal{A}}_\mathcal{C},\mathcal{Z}_\rho}$ holds for any environment $\mathcal{Z}_\rho$. (Here $\tilde{\mathcal{A}}_\mathcal{C}$ is the dummy adversary from Definition 4.)

Adversary $\mathcal{H}$ is presented in Figure 4. We use the following additional notation. A copy of protocol ρ with SID id, running within party P_i, is labeled $(i,(id))$. A message that is addressed to copy $(i,(id))$ of ρ within the recipient is called a $\rho_{i,(id)}$-message. A message that is addressed to protocol π within the recipient is called a π-message.

Validity of $\mathcal{H}$. Assume that there exists an environment machine $\mathcal{Z}$ that violates the validity of $\mathcal{H}$ (that is, $\mathcal{Z}$ violates Equation (5)). We construct an environment machine $\mathcal{Z}_\rho$ that violates the validity of $\mathcal{S}$ with respect to a single run of ρ. (That is, $\mathcal{Z}_\rho$ violates Equation (6).) Let us first set some conventions that will simplify the presentation and analysis of $\mathcal{Z}_\rho$. Assume that, even in the real-life model of computation, the parties have access to an ideal functionality, denoted $\mathcal{N}$, that, when activated by P_i, responds with $\perp$ and does nothing else. (I.e., $\mathcal{N}$ writes $\perp$ on P_i's incoming tape and enters the waiting state.) Clearly, this convention does not change any substantial aspect of the real-life model.

Next, define the following slight variation of protocol π^ρ. Let $\hat{\pi}^\rho$ be the protocol, running in a hybrid model, where the parties run π^ρ with the following exception. Whenever π^ρ instructs some party P_i to activate a copy $\rho_{i,(id)}$ of ρ with input x, $\hat{\pi}^\rho$ instructs P_i to first activate the corresponding copy of the ideal functionality (i.e., the copy whose SID is id) with input (x, id). If the ideal functionality responds with $\perp$, then P_i proceeds to invoke $\rho_{i,(id)}$ as in π^ρ. Otherwise, $\hat{\pi}^\rho$ proceeds as protocol π would with access to the ideal functionality $\mathcal{F}_{(id)}$.

Let $m = m(k)$ be an upper bound on the number of invocations of ρ within π, given security parameter k. (The bound m is used in the analysis only. The parties need not be aware of m.) Let the $\mathcal{F}^{(l)}$-hybrid model denote the $\mathcal{F}$-hybrid model where all the copies of $\mathcal{F}$, except for the first l copies to be invoked, are replaced with copies of the null functionality $\mathcal{N}$. That is, in the $\mathcal{F}^{(l)}$-hybrid model all the calls to the copies of $\mathcal{F}$ up to the lth copy are answered as in the $\mathcal{F}$-hybrid model; the calls to the remaining copies of $\mathcal{F}$ are answered with $\perp$ (thus forcing $\hat{\pi}^\rho$ to run copies of ρ instead). Then, the $\mathcal{F}^{(m)}$-hybrid model is the same as the $\mathcal{F}$-hybrid model. Similarly, the $\mathcal{F}^{(0)}$-hybrid mode is the same as the real-life model of computation.

Notice that, when $\mathcal{H}$ interacts with parties running $\hat{\pi}^\rho$ in the $\mathcal{F}^{(l)}$-hybrid model, $\mathcal{H}$ invokes only l copies of the simulator $\mathcal{S}$. (These are the copies that correspond to the first l copies of the ideal functionality $\mathcal{F}$.) The rest of the invocations of the ideal functionality are answered with $\perp$ and protocol ρ is invoked instead. In particular, in the $\mathcal{F}^{(m)}$-hybrid model an execution of protocol $\hat{\pi}^\rho$ with adversary $\mathcal{H}$ and environment $\mathcal{Z}$ is identical to an execution of protocol π with $\mathcal{H}$ and $\mathcal{Z}$. Similarly, in the $\mathcal{F}^{(0)}$-hybrid model, an execution of protocol $\hat{\pi}^\rho$ with $\mathcal{H}$ and $\mathcal{Z}$ is in essence identical to an execution of protocol π^ρ with adversary $\tilde{\mathcal{A}}_\mathcal{C}$ and environment $\mathcal{Z}$. (This holds since, in the $\mathcal{F}^{(0)}$-hybrid model, $\mathcal{H}$ will not invoke any copy of $\mathcal{S}$ and will end up running the code of $\tilde{\mathcal{A}}_\mathcal{C}$ without any deviations.) Consequently, Equation (5) can be rewritten as:

$$\text{HYB}^{\mathcal{F}^{(0)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}^{(m)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}. \tag{7}$$

Recall that our goal is to construct, given an environment $\mathcal{Z}$ that violates (7), an environment $\mathcal{Z}_\rho$ that violates (6). Environment $\mathcal{Z}_\rho$ is described in Figure 5. The rest of the proof analyzes the performance of $\mathcal{Z}_\rho$. Let $k \in \mathbf{N}$ be some value of the security parameter, and let $z \in \{0,1\}^*$ be some input value for $\mathcal{Z}$. Assume that, given (k, z), $\mathcal{Z}$ distinguishes with probability ϵ between an interaction in the $\mathcal{F}^{(m)}$-hybrid model and an interaction in the $\mathcal{F}^{(0)}$-hybrid model. (Both interactions are with $\mathcal{H}$ and parties running $\hat{\pi}^\rho$.) That is, assume that:

$$|\text{HYB}^{\mathcal{F}^{(0)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z) - \text{HYB}^{\mathcal{F}^{(m)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z)| \geq \epsilon. \tag{8}$$

Consequently, there exists a value $l \in \{1, ..., m\}$ such that

$$|\text{HYB}^{\mathcal{F}^{(l)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z) - \text{HYB}^{\mathcal{F}^{(l-1)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z)| \geq \epsilon/m. \tag{9}$$

Environment $\mathcal{Z}_\rho$

Environment $\mathcal{Z}_\rho$ proceeds as follows, given a value k for the security parameter, input z_ρ, and interacting with parties $P_1, ..., P_n$ running protocol ρ and with and adversary $\mathcal{A}$. ($\mathcal{A}$ is expected to run the code of the dummy adversary $\tilde{\mathcal{A}}_C$.)

When activated for the first time, interpret the input z_ρ as a pair $z_\rho = (z, l)$ where z is an input for $\mathcal{Z}$, and $l \in \mathbf{N}$. Let variable s hold the initial global state of a system of parties $P'_1, ..., P'_n$ running protocol π^ρ in the $\mathcal{F}^{(l)}$-hybrid model, with the dummy adversary $\tilde{\mathcal{A}}_C$ and environment $\mathcal{Z}$ on input z. (Initially, the active entity in state s is the environment $\mathcal{Z}$.) Proceed to Step 2. In each activation other than the first, start in Step 1.

1. Update the state s:
   (a) If the output tape of P_i contains a new output value y then update the internal state of the simulated P'_i by including a message y received from the lth copy of $\mathcal{F}$, denoted $\mathcal{F}_{(id_l)}$.
   (b) If the output of $\mathcal{A}$ includes new information then treat this information as the output of $\tilde{\mathcal{A}}_C$ and update s to include this information. (This information will be either new outgoing $\rho_{i,(id_l)}$-messages of some party P_i, or the internal state of some newly corrupted party.)
2. Simulate an execution of the system in the $\mathcal{F}^{(l)}$-hybrid model, from state s, until one of the following events occurs:
   (a) Party P'_i sends a message x to $\mathcal{F}_{(id_l)}$. In this case, save the current state of the simulates system in s, write x on the input tape of P_i, and enter the waiting state. *(P_i will be activated next. In the next activation of $\mathcal{Z}_\rho$, which will occur after the activation of P_i is complete, P'_i may have output values. These values will be used to update s in Instruction 1a.)*
   (b) The simulated dummy adversary is activated with some input value v. In this case, save the current state of the simulated system in variable s, write v on the input tape of $\mathcal{A}$, and enter the waiting state. *($\mathcal{A}$ will be activated next. In the next activation of $\mathcal{Z}_\rho$, either $\mathcal{A}$ or one of the parties may have new output values. These values will be used to update s in Instruction 1b.)*
   (c) Environment $\mathcal{Z}$ halts. In this case, output whatever $\mathcal{Z}$ outputs and halt.

Figure 5: The environment for a single copy of ρ.

However, for every $l \in \{1, .., m\}$ we have

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho}(k,(z,l)) = \text{HYB}^{\mathcal{F}^{(l)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z) \tag{10}$$

and

$$\text{REAL}_{\rho,\tilde{\mathcal{A}}_C,\mathcal{Z}_\rho}(k,(z,l)) = \text{HYB}^{\mathcal{F}^{(l-1)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z). \tag{11}$$

Equations (10) and (11) follow from inspecting the code of $\mathcal{Z}_\rho$ and $\mathcal{H}$. In particular, if $\mathcal{Z}_\rho$ interacts with the ideal process for $\mathcal{F}$ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ when interacting with $\hat{\pi}^\rho$ and $\mathcal{H}$ in the $\mathcal{F}^{(l)}$-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running ρ in the real-life model then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ when interacting with $\hat{\pi}^\rho$ and $\mathcal{H}$ in the $\mathcal{F}^{(l-1)}$-hybrid model.

From Equations (9), (10) and (11) it follows that:

$$|\text{REAL}_{\rho,\tilde{\mathcal{A}}_{\mathcal{C}},\mathcal{Z}_\rho}(k,(z,l)) - \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho}(k,(z,l))| \geq \epsilon/m. \quad (12)$$

This contradicts the validity of $\mathcal{S}$, thus completing the analysis of $\mathcal{Z}_\rho$ and the proof of Theorem 6.

6 Other computational models

Definition 3 captures one specific model of computation, which we call the bare model. Essentially, this model captures networks with unauthenticated communication, asynchronous message delivery, and an adversary that corrupts parties in an adaptive way throughout the computation. (See more discussion in Section 2.)

Using the modularity of the framework, the bare model can be extended to capture several other popular computational models by casting them as hybrid models with access to some simple ideal functionalities. Examples include the standard models of ideally authenticated communication, ideally secret communication, communication via broadcast channel, and the common random string model. We postpone the discussion and formal description of these models to Section 8.

This section discusses a number of computational models that cannot naturally be cast as hybrid models as in Section 5.1. Instead, we demonstrate how the these models can be captured by imposing appropriate restrictions on the environments, real-life adversaries, and ideal-process adversaries under consideration. We first address the familiar models of non-blocking networks (i.e., networks with asynchronous communication but guaranteed message delivery), synchronous communication, static corruption of parties and passive control on corrupted parties. Next we present and discuss an extension to the bare model, where ideal functionalities can request to have output values delivered immediately to the recipient parties.

We remark that the composition theorem holds with respect to each one of the variants described below. That is, if the given real-life adversary belongs to the class of adversaries defined by some adaptation of the real-life model then, for any environment within the given class of environments, the resulting hybrid-model adversary belongs to the class of adversaries that correspond to the adapted version of the hybrid model.

The variants are presented in terms of the main definition of security (Definition 3). A presentation in terms of the alternative formalization of the definition (Definition 4) can be easily inferred.

Non-blocking networks. A popular variant of the basic asynchronous model of the previous section is the model where it is guaranteed that any message that was sent is eventually delivered. Say that this is the asynchronous model with non-blocking adversaries. An interesting feature of this variant is that here it is reasonable to require that protocols eventually generate outputs. That is, protocols that "do nothing" should no longer be considered secure. (This stands in contrast to the basic asynchronous model; see discussion in Section 2.)

Our way of forcing protocols to generate outputs is to modify the ideal process, so that the ideal-process adversary is obliged to eventually deliver any message sent by the ideal functionality. That is, say that a ideal-process adversary is non-blocking if any message sent by the ideal functionality to some dummy party is eventually delivered. Say that a protocol securely realizes an ideal functionality $\mathcal{F}$ w.r.t. class $\mathcal{C}$ of adversaries in non-blocking networks if for any *non-blocking* adversary $\mathcal{A} \in \mathcal{C}$ there exists a *non-blocking* ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ holds for any environment $\mathcal{Z}$.

Synchronous networks. A popular and convenient abstraction of communication networks is that of *synchronous* communication. Roughly speaking, here the computation proceeds in *rounds,* where in each round each party receives all the messages that were sent to it in the previous round, and generates outgoing messages for the next round. The order of activation of parties within a round is typically assumed to be under the control of the adversary, thus the messages sent by the corrupted parties may depend on the messages sent by the uncorrupted parties in the *same round.* (This model is often called the "rushing" model for synchronous networks.) Definition 3 can be modified as follows in order to capture synchronous networks:

1. Restrict attention to synchronous environment machines. These are environment machines that start the computation in a sequence of n activations, where in each activation a value is written on the input tape of one of the parties. (The input tape of each party is written to exactly once. The order of activation is up to the environment.)
2. Restrict attention to synchronous real-life adversaries. These are adversaries that deliver messages in rounds, as follows. The messages generated in the first activation of each party are called the round 0 messages. For each round number $\ell = 1, 2, ...$ the adversary activates the parties, in an arbitrary order, to deliver all the round $\ell - 1$ messages. The messages generated in these activations are called the round ℓ messages.
3. Require the ideal-process adversary to be *non-blocking,* as defined above for the case of non-blocking networks. As there, the purpose of this requirement is to disallow protocols that do not generate output.

That is, say that a protocol securely realizes an ideal functionality $\mathcal{F}$ in synchronous networks (w.r.t. class $\mathcal{C}$ of adversaries) if for any *synchronous* adversary $\mathcal{A} \in \mathcal{C}$ there exists a *non-blocking* ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ holds for any *synchronous* environment $\mathcal{Z}$.

Non-adaptive (static) adversaries. Definition 3 postulates adaptive adversaries, namely adversaries that may corrupt parties as the computation proceeds, based on the information gathered so far. Arguably, adaptive corruption of parties is a realistic threat in existing networks. Nonetheless, it is sometimes useful to consider also a weaker threat model, where the identities of the adversarially controlled parties are fixed before the computation starts; this is the case of non-adaptive (or, static) adversaries. See more discussion on the differences between the two models in [C00].

Security against non-adaptive adversaries is captured by considering only environments and real-life adversaries where all the party corruption activities are performed before any party is activated. (Recall that in the case of non-concurrent composition and non-adaptive adversaries, treated in [C00], there was no need to include the environment machine in the model. In the case of security that is closed under concurrent composition the environment machine seems to be needed even with non-adaptive adversaries.)

Passive (eavesdropping) adversaries. Definition 3 postulates active adversaries, namely adversaries that have total control over the behavior of corrupted parties. Another standard corruption model assumes that even corrupted parties continue to follow their prescribed protocol. Here the only advantage the adversary gains from corrupting parties is in learning the internal states of those parties. We call such adversaries passive.

The definition can be adapted in a straightforward way to deal with the case of passive adversaries. Specifically:

- Modify the real-life model as follows: Even after being corrupted by the adversary (Step 1(b)iii in Figure 1), the ITM that corresponds to the corrupted party keeps being activated for receiving inputs and incoming messages, and keeps generating outputs and outgoing messages, with the exception that now the adversary has read access to the memory contents of the corrupted parties.

- Modify the ideal process as follows: Even after being corrupted by the adversary (Step 1(d)iv in Figure 2), the corrupted dummy party $\tilde{P}_i$ can keep being activated for receiving inputs from the environment, and keeps forwarding any received input to the ideal functionality. The adversary cannot send $\mathcal{F}$ messages with source identity $\tilde{P}_i$.

The rest of the definition remains unchanged.

Immediate output generation. Consider some party P_i that runs some protocol π in an asynchronous network with adversarially controlled message delivery. Assume that P_i has just activated π with some input value, and expects an output value. We distinguish between two different types of behavior of π. One option is that π will generate an output value at some future activation of P_i. This behavior is typical when the execution of π involves sending messages and waiting for responses from other parties on the network. Here it is the adversary (who controls the scheduling of message delivery) who decides in which future activation P_i will receive the output value from its local copy of π. We call this type of output generation delayed output. The other option is that π generates its output value *immediately*, that is in the same activation in which the input was given. This behavior occurs when generating the output requires only local computation and there is no need to wait for messages from other parties. We call this type of output generation immediate output. An example of a functionality where the output is typically expected to be immediate is digital signatures: both the signature and the verification processes are typically done locally without any need for communication.

Recall that the ideal process of Section 4.2 allows the ideal-process adversary to arbitrarily delay the messages sent from the ideal functionality to the parties. This means that the definition of security provides no way to require that message delivery be immediate. (That is, there is no way to distinguish, on the requirements level, between immediate and delayed output.) This "slack" in the model is tolerable for most tasks. However, for some tasks it may be important to be able to require that message delivery be immediate. This can be done as follows. Whenever the ideal functionality wishes to have a message (carrying an output value) delivered immediately to some party P_i, it writes the message on its output tape as usual, and in addition sends a message to the adversary with a request to deliver this message immediately. Say that an ideal-model adversary is immediate if, upon receiving such a request from the ideal functionality, it delivers the said message in the same activation. Then, strengthen Definition 3 to quantify only over all *immediate* ideal-process adversaries.

Say that an ideal functionality is immediate if it uses the above mechanism for immediate message delivery. We stress that the general realizability theorem (see Section 7) does not apply to immediate functionalities. In fact, immediate functionalities can be used to write specifications that cannot be realized by any protocol. For instance, the *Random Oracle Model* (see, e.g., [BR93A, CGH98]) can be cast as a hybrid model with access to an immediate ideal functionality $\mathcal{F}_{\text{RO}}$.[5]

[5]When receiving a value x from P_i, $\mathcal{F}_{\text{RO}}$ will proceed as follows. If x was never before given to $\mathcal{F}_{\text{RO}}$, then $\mathcal{F}_{\text{RO}}$ sets a variable y to a random value from an appropriate domain. If x was previously given to $\mathcal{F}_{\text{RO}}$ (by any party) then set y to the value previously answered for that x. Finally, write y on the incoming communication tape of P_i.

7 Realizing general ideal functionalities

This section argues that realizing ideal functionalities in the present formalization is, in general, feasible. That is, we demonstrate that a large class of ideal functionalities can be securely realized in some standard settings, using known techniques, as long as the protocol involves a set of parties such that only a certain fraction of these parties are corrupted.

As remarked in Section 2, in models where message delivery is not guaranteed the protocol that generates no messages and no outputs securely realizes any ideal functionality. Consequently, a general feasibility theorem is of no interest in such cases. In contrast, a general feasibility theorem is indeed of interest in settings where message delivery is guaranteed, and protocols are required to generate output. Two important such settings are the cases of *synchronous* networks and *asynchronous networks with guaranteed delivery* (see Section 6). The rest of this section concentrates on synchronous networks.

We restrict the discussion to *authenticated networks*, i.e. to networks with ideally authenticated communication. (See Section 8.1.1 for more discussion and formal definition.) In this setting, assume that there exists a set A of parties (called "servers") such that less than $|A|/3$ of the servers may be corrupted by the adversary throughout the computation. We show:

Theorem 9 *Consider a synchronous network with a set of servers as described above, and assume that trapdoor permutations exist. Then, for any ideal functionality $\mathcal{F}$ there exists a protocol that securely realizes $\mathcal{F}$ in this network.*

A crucial ingredient in the proposed protocol is to have the protocol include "assistant programs" to be run by the servers. These programs will not have local inputs or outputs; instead, they will be invoked by incoming messages from other parties, and will generate messages to other parties (and other servers). Specifically, we use the construction of [BGW88], with the simplification of [GRR98], and in addition encrypt each message using non-committing encryption [CFGN96].

A sketch of proof appears below. Let us remark that variants of the above theorem hold in most other standard models of computation. In the case of ideally secure communication, encrypting messages (and the assumption that trapdoor permutations exist) is not necessary. In the case of synchronous networks with a broadcast channel, $t < n/2$ is sufficient using the techniques of [RB89, CDDHR99]. In the case of non-blocking asynchronous networks a similar result holds using the techniques of [BCG93, BKR94].

Proof (sketch): We provide a very high level sketch of proof of Theorem 9. Let $t < n/3$ and let $\mathcal{F}$ be an ideal functionality that is not immediate. The first step in constructing a protocol that t-securely realizes $\mathcal{F}$ is to represent $\mathcal{F}$ via a circuit-family $C_{\mathcal{F}}$ as described in Section 3. That is, the kth circuit in the family $C_{\mathcal{F}}$ describes an activation of $\mathcal{F}$ with security parameter k. The inputs to the circuit represent the initial internal state of $\mathcal{F}$ in this activation plus the contents of the incoming message from some party. The outputs of the circuit represent the final internal state of $\mathcal{F}$ in this activation, plus the outgoing messages. Specifically, we use arithmetic circuits over Z_p^* where $p > n$ is a prime and the operations are modular addition and multiplication.

Sketch of the protocol. The protocol for realizing $\mathcal{F}$ proceeds as follows, given security parameter k. Whenever a party P_i is activated with a new input value x, it sends a request to the parties in A to securely evaluate the kth circuit in $C_{\mathcal{F}}$. The evaluation is done in a gate-by-gate manner, as in [BGW88]. The values associated with the input lines to the circuit are determined as follows. Party P_i provides the parties in A with the values associated with the lines that represent the message received from P_i. (This is done using a Verifiable Secret Sharing protocol, as there.) The other input lines to the circuit represent the initial state of $\mathcal{F}$ in this activation. This state equals the final state of $\mathcal{F}$ in the previous activation; thus the parties in A already have the values associated with these lines. (In the first activation, the values of these lines are fixed and known, and the evaluated circuit is degenerated accordingly.)

Once the output lines of the circuit are evaluated, the parties in A proceed as follows. The values associated with each line that represents a message to be sent to party P_j are revealed to P_j. Values addressed to the adversary are revealed to all parties. The values associated with the output lines that represent the state of $\mathcal{F}$ at the end of the activation are kept by the parties in A, to be used in the next evaluation of the circuit. Whenever P_j receives the values associated with an output line assigned to itself, it reconstructs the corresponding output value and writes it on its output tape.

In oder to hide which party receives input at which round, the parties can provide (possibly null) input to $C_{\mathcal{F}}$ at each round. Similarly, $C_{\mathcal{F}}$ is modified so that if the input is null then the activation is aborted. In order to hide which party generates output at which round, $C_{\mathcal{F}}$ can be modified so that each party receives a (possibly null) output at each round. Only non-null values will be copied to the party's output tape.

Ordering the activations. One important missing issue, among the many missing in the above sketch, is how the parties "synchronize" the various evaluations of $\mathcal{F}$. That is, the parties in A need to agree on an ordering of the activations of the circuit. This is done as follows: Evaluations of the circuit are carried out sequentially, one by one. At the end of each evaluation, the parties in A run a protocol for agreement on which evaluation request to fulfill next. This request should be one of those that were made in the earliest communication round and is not yet fulfilled. (There is no required order among the requests made in the same round.)

Adding encryption. In the ideally-secret communication model the above construction suffices. In the authenticated model, where the adversary sees all messages sent by the parties, we encrypt each message using an appropriate encryption protocol: In the case of non-adaptive adversaries, semantically secure encryption [GM84] is sufficient (if a different set of keys is used for each pair or parties). In the case of adaptive adversaries, if one trusts data erasures then simple extensions of semantically secure encryption suffice [BH92]. If data erasures are not trusted then non-committing encryption [CFGN96, B97, DN00] can be used.

Sketch of analysis. Let $\mathcal{F}$ be a non-immediate functionality. Essentially, the proof that the above construction securely realizes $\mathcal{F}$ is a natural extension of the proof that the [BGW88, GRR98] construction securely evaluates any given function, according to known definitions of secure function evaluation. Unfortunately, such a proof does not appear in the literature; nonetheless, the general structure of such a proof (and many of the details) are very similar to the proof of the main theorem in [CKOR00]. There are three main differences between our setting and the setting considered in there. ([CKOR00] consider secure function evaluation in the *secure channels* setting of [C00].) Let us highlight these differences:

1. The model here includes the environment machine, which interacts with the parties and the adversary (both in the real-life model and in the ideal process). However, we claim that the simulator constructed in [CKOR00] can be modified in a straightforward way to accommodate the addition of the environment machine. Recall that in [CKOR00] the simulator $\mathcal{S}$ proceeds via "one pass black-box simulation" of the real-life adversary $\mathcal{A}$. That is, $\mathcal{S}$ runs $\mathcal{A}$ on a simulated interaction with some network, and interprets the instructions of $\mathcal{A}$ in some ways. When $\mathcal{A}$ generates output, $\mathcal{S}$ generates the same output. Such a simulation technique is

readily amenable to adding the environment: whenever the simulated $\mathcal{A}$ sends a message to the environment, $\mathcal{S}$ (treats this message as an output of $\mathcal{A}$ and) forwards the same message to its environment. Whenever $\mathcal{S}$ receives an input value from the environment, it forwards this value to $\mathcal{A}$. Other than that, $\mathcal{S}$ proceeds as in [CKOR00].[6]

2. In [CKOR00] the circuit is evaluated once, whereas here the parties evaluate the circuit many times in a single execution of the protocol, where the circuit evaluations are executed sequentially, and the inputs to an evaluation may depend on the outputs of previous evaluations. This can be dealt with in a straightforward way by running the ideal-process adversary (i.e., the simulator) of [CKOR00] on each of the circuit evaluations, where at the beginning of each evaluation the simulator is given the current internal state of the adversary.

3. Here the secrecy of the communication is protected via encryption, whereas in the [CKOR00] model the communication is ideally secure. This difference can be dealt with using the composition theorem. That is, it suffices to prove security of the protocol in the $\mathcal{F}_{\text{SMT}}$-hybrid model (see Section 8.1.2), which is essentially a reformulation of the model of [CKOR00].[7]

Remark: We stress that the above construction and analysis do not work if the ideal functionality $\mathcal{F}$ is immediate. In particular, if $\mathcal{F}$ is immediate then the environment may activate some party with input value, and then expect the party to generate output *in the same activation.* This cannot be realized using the above construction. □

8 Some ideal functionalities

This section demonstrates the general applicability of the framework by using it to provide universally composable definitions of a number of known cryptographic tasks. To do that, we formulate ideal functionalities that capture the security requirements of these tasks; a protocol is said to securely carry out a task if it securely realizes the corresponding ideal functionality as in Definition 3.

We first formulate ideal functionalities that represent the tasks of guaranteeing authenticated and secure communication (both in a message-by-message fashion and in "secure sessions"), as well as the related task of key-exchange. We then proceed to other basic tasks of cryptographic protocols, such as public-key encryption and digital signatures. Next we formulate functionalities representing two-party primitives such as "coin-tossing" (i.e., generating a common random string), commitment, oblivious transfer, and zero-knowledge. These primitives are treated as two-party protocols in a multi-party setting. Finally, we formulate ideal functionalities that capture Verifiable Secret Sharing and Secure Function Evaluation in *synchronous* networks.

[6]An alternative way to see how the simulator here relates to the simulator of [CKOR00] is to consider the alternative formulation of the definition of security, i.e., Definition 4. Here the simulator $\mathcal{S}$ has to mimic the behavior of the dummy adversary $\tilde{\mathcal{A}}$, or on other words to respond to instructions to deliver messages and report on sent messages and internal states of corrupted parties. This is exactly the functionality provided by the simulator of [CKOR00].

We stress that it is crucial that the simulator of [CKOR00] never "rewinds" the adversary. If it did, then the interaction between our simulator and the environment would be distinguishable from the interaction between the adversary and the environment. Indeed, we chose the construction of [BGW88, GRR98] because it can be proven secure via one-pass black-box simulation. We do not know whether other general constructions for secure function evaluation, such as that of [GMW87, G98] (whose only known proof of security does not proceed via one-pass, black-box simulation), are secure in the present framework.

[7]In fact, the synchronous version of the $\mathcal{F}_{\text{SMT}}$-hybrid model is needed. The solutions proposed in Section 8.1.2 work in this model as well.

Each one of the ideal functionalities presented below corresponds to a *single invocation* of the primitive it represents. This results in functionalities that are relatively simple and easy to work with. The composition theorem guarantees that protocols that securely realize each such functionality remain secure even when many copies are running concurrently, by different subsets of the parties, and in an adversarially scheduled way. Furthermore, these protocols are guaranteed to "faithfully mimic" the corresponding functionality for *any application* that makes use of that functionality.

All the ideal functionalities in this section, with the exception of the one capturing the security requirements of digital signatures, are non-immediate. It thus follows from Section 7 that these ideal functionalities can be realized in settings where trusted third parties (or, servers) are available. In addition, we briefly discuss the feasibility of realizing these ideal functionalities by more efficient protocols (and in particular by two-party protocols, in the relevant cases). In the case of secret communication and digital signatures we demonstrate how protocols that securely realize the proposed ideal functionalities can be constructed from schemes that satisfy known security requirements.

8.1 Secure Communication

Assuming that the underlying communication links are "ideally authenticated", or even "ideally secure", are powerful and popular abstractions for designing cryptographic protocols. This section casts these abstractions in the present framework, and provides universally composable definitions for some salient tasks. We first address the notions of *authentic* and *secure* (i.e., authentic and secret) message transmission. Next, we address *secure sessions,* where secrecy and authenticity is guaranteed for an entire "conversation" (i.e., an aggregate of several messages) between two parties via a single application of the underlying protocol. Finally, we address the task of *key exchange,* which is a useful step towards realizing secure sessions.

8.1.1 Authenticated Message Transmission

Ideally authenticated message transmission means that a party P_i will receive a message m from some party P_j only if P_j has sent the message m to P_i. Furthermore, if P_j sent m to P_i only t times then P_i will receive m from P_j at most t times. (These requirements are of course meaningful only as long as both P_i and P_j are uncorrupted.)

In the present formalization, networks that guarantee ideally authenticated message transmission can be cast as a hybrid model with ideal access to an "authenticated message transmission functionality". This functionality, denoted $\mathcal{F}_{\text{AUTH}}$, is presented in Figure 6. Notice that $\mathcal{F}_{\text{AUTH}}$ deals with authenticated transmission of a single message. Authenticated transmission of multiple messages is obtained via the composition theorem. Also, $\mathcal{F}_{\text{AUTH}}$ reveals the contents of the message to the adversary. This captures the fact that secrecy is not provided. In the rest of this work, we refer to the $\mathcal{F}_{\text{AUTH}}$-hybrid model as the authenticated links model (AM).

Protocols that securely realize $\mathcal{F}_{\text{AUTH}}$ are closely related to the notion of authenticators, that was used in [BCK98] in a setting similar to ours and in [CHH00] in a different setting. These are general "compilers" that transform any protocol that works in a model with ideally authenticated links (e.g., the AM) into a protocol with essentially the same functionality in a model where the communication is not authenticated a-priori. Specifically, applying an authenticator to a protocol π (that was designed for the AM) now amounts to composing π with a protocol ρ that securely realizes $\mathcal{F}_{\text{AUTH}}$.

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

$\mathcal{F}_{\text{AUTH}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a message $(\texttt{send}, id, P_j, m)$ from party P_i, send (id, P_i, P_j, m) to P_j and to the adversary, and halt.

Figure 6: The Message Authentication functionality, $\mathcal{F}_{\text{AUTH}}$

We focus on the authenticators of [BCK98]. Recall that these authenticators require an "initialization function" that models an initial ideally authentic distribution of long-term keys (say, verification keys of a signature scheme). This idealized initialization function can be replaced with a number of set-up protocols, such as a protocol for interacting with a certification authority. Alternatively, the set-up protocol can be treated as an integral part of the authentication protocol. In any case, some type of ideally authenticated communication in some set-up stage is essential for achieving realizing $\mathcal{F}_{\text{AUTH}}$.

Claim 10 *The encryption-based and the signature-based* MT*-authenticators of* [BCK98] *securely realize functionality $\mathcal{F}_{\text{AUTH}}$, given an ideally authenticated initialization stage.*

Proof: The proof is a simple extension of the proofs in [BCK98], and is omitted. (Recall that the proof there proceeds by having the adversary in the $\mathcal{F}_{\text{AUTH}}$-hybrid model carry out a one-pass black-box simulation of the adversary in the bare model. Consequently, adapting the [BCK98] proof can be done by incorporating the environment in a straightforward way, as done in the proof of Theorem 9.) □

8.1.2 Secure Message Transmission

The abstraction of secure message transmission usually means that the transmission is ideally authenticated as in the AM, and in addition that the adversary has no access to the contents of the transmitted message. It is typically assumed that the adversary learns the length of the transmitted message (or some bound on it). In the present framework, networks that guarantee idealized secure message transmission are cast as a hybrid model with ideal access to the "secure message transmission functionality", $\mathcal{F}_{\text{SMT}}$, presented in Figure 7. $\mathcal{F}_{\text{SMT}}$ reveals the exact length of the message to the adversary. Variants that reveal only partial information on the length are of course possible. Also, similarly to $\mathcal{F}_{\text{AUTH}}$, $\mathcal{F}_{\text{SMT}}$ deals with transmission of a single message. Secure transmission of multiple messages is obtained via the composition theorem.

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

$\mathcal{F}_{\text{SMT}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a message $(\texttt{send}, id, P_j, m)$ from party P_i, send (id, P_i, P_j, m) to P_j and $(id, P_i, P_j, |m|)$ to the adversary, and halt.

Figure 7: The Secure Message Transmission functionality, $\mathcal{F}_{\text{SMT}}$

On hiding the length and traffic analysis. Notice that, whenever a party P_i sends a message to some P_j, $\mathcal{F}_{\text{SMT}}$ notifies the adversary that P_i sent a message to P_j. This reflects the common view that encryption does not hide the fact that a message was sent. (Using common terminology, there is no protection against traffic analysis.) Furthermore, $\mathcal{F}_{\text{SMT}}$ reveals the exact length of that message. Encryption schemes that hide information on the length of messages, or schemes that hide the fact that a message was sent, can be modeled by appropriate modifications to $\mathcal{F}_{\text{SMT}}$.

Securely realizing $\mathcal{F}_{\text{SMT}}$. Protocols that securely realize $\mathcal{F}_{\text{SMT}}$ can be constructed in the $\mathcal{F}_{\text{AUTH}}$-hybrid model (i.e., in the AM) based on semantically secure public-key encryption schemes, by using each encryption key for encrypting only a single message. That is, let $S = (gen, enc, dec)$ be an encryption scheme. (Here gen is the key generation algorithm, enc is the encryption algorithm and dec is the decryption algorithm.) Then, consider the following protocol, denoted π_S. When invoked within P_i with a $(\texttt{send}, id, P_j, m)$ request, π_S first sends an $(\texttt{init}, id)$ message to P_j. P_j then runs algorithm gen, gets the secret key sk and the public key pk, and sends (id, pk) back to P_i. Next, P_i sends $(id, enc(pk, m))$ to P_j and returns. Finally, upon receipt of (id, c), π_S within P_j outputs $dec(sk, c)$ and returns.

Choosing new keys for each message to be transmitted is of course highly inefficient and does not capture common practice for achieving secure communication, Nonetheless, it is easy to see that:

Claim 11 *If S is semantically secure as in* [GM84] *then π_S securely realizes $\mathcal{F}_{\text{SMT}}$ in the* AM, *in the presence of non-adaptive adversaries.*

Claim 12 *If S is semantically secure and π_S is modified so that the parties erase the local random bits and secret keys immediately after encryption and decryption, then π_S securely realizes $\mathcal{F}_{\text{SMT}}$ in the* AM *with adaptive adversaries, in a model where data erasures are trusted.*

Furthermore, if S is non-committing as in [CFGN96] *then π_S securely realizes $\mathcal{F}_{\text{SMT}}$ in the* AM *with adaptive adversaries, even if data erasures are not trusted and the adversary sees all the past internal states of the corrupted parties.*

8.1.3 Secure Sessions

A typical communication pattern is to aggregate a number of messages between two parties in a *session.* (Sessions are often the result of running an instance of some higher-level protocol between the two parties.) It thus makes sense to protect (i.e., provide authenticity and secrecy for) all the messages in a session via a single invocation of some protocol. Indeed, protocols for realizing secure sessions are considerably more efficient than protocols that protect each message individually.

We cast the task of providing secure sessions via the ideal secure session functionality, $\mathcal{F}_{\text{SS}}$, presented in Figure 8. This functionality first waits to receive the identities of two parties, P_i and P_j, and then sets up a "communication session" between them (and notifies the adversary that the communication session is established). Next $\mathcal{F}_{\text{SS}}$ simply forwards messages between these two parties while notifying the adversary whenever a message was sent. The functionality also provide a way for a party to *terminate* a session. (A similar functionality, that provides only authenticates sessions, reveals the entire contents of each message m to the adversary rather than just reveling the length of m.)

A typical methodology for securely realizing $\mathcal{F}_{\text{SS}}$ starts with a Key-Exchange protocol, and then encrypts and authenticates each message via symmetric encryption and message authentication

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

$\mathcal{F}_{\text{SS}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value $(\texttt{set-up}, id, P_j)$ from some party, P_i, wait to receive a value $(\texttt{set-up}, id, P_i)$ from P_j. Once this value is received, set a boolean variable `active` and send (id, P_i, P_j) to the adversary.
2. Upon receiving a value $(\texttt{send}, id, m)$ from P_i, and if `active` is set, send (id, m) to P_i and $(id, P_i, |m|)$ to the adversary.
3. Upon receiving a value $(\texttt{send}, id, m)$ from P_j, and if `active` is set, send (id, m) to P_i and $(id, P_j, |m|)$ to the adversary.
4. Upon receiving a value $(\texttt{terminate}, id)$ from either P_i or P_j, unset the variable `active` and halt.

Figure 8: The Secure Session functionality, $\mathcal{F}_{\text{SS}}$

codes using the shared key. An alternative treatment of this methodology, not based on the current framework, is given in [CK01]. (Relations between the two treatments are studied in [CK01a].)

8.1.4 Key Exchange

Key Exchange (KE) is a functionality where two parties interact in order to generate a common random value (a key) that remains unknown to everyone except the two parties. Typically, the key is then used as a "session key" for symmetric encryption and authentication of messages, in a secure session protocol. Furthermore, multiple copies of the protocol may be carried out between different pairs of parties concurrently, in an adversarially controlled way. There exists a large body of work on the security requirements from KE protocols. Here let us mention only [BAN90, DOW92, BR93, BCK98, Sh99]. See more details there.

In the present framework, a secure KE protocol is cast as a protocol that securely realizes functionality $\mathcal{F}_{\text{KE}}$, presented in Figure 9. The functionality provides pairs of parties with a randomly chosen shared key. It also captures the fact that there is no requirement on the distribution of keys in sessions where one of the participants is corrupted. (This is done by allowing parties to fix the generated key to any desired value, via the α field in the $(\texttt{exchange...})$ message. An uncorrupted user is expected to set $\alpha = \perp$.)

As usual, functionality $\mathcal{F}_{\text{KE}}$ is defined for a single invocation of the protocol (i.e., for a single exchange of a key). The composition theorem guarantees that security is maintained when many sessions are running concurrently. Furthermore, it guarantees that the generated key is as secure as an "ideally random key" for any application protocol that uses these keys. In contrast, other definitions of KE incorporate multiple sessions of the protocol *in the actual definition,* and do not rigorously argue on the sufficiency of the definition for guaranteeing the security of protocols that use the shared key. (Exceptions are the work of Goldreich and Lindell [GL00] that defines KE protocols for a single session and then shows secure *non-concurrent* composition, and the work of Canetti and Krawczyk [CK01] that demonstrate the security of the generated key for symmetric encryption and authentication.)

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

$\mathcal{F}_{\text{KE}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Wait to receive a value $(\texttt{exchange}, id, P_i, P_j, \alpha)$ from some party P_i.
2. Wait to receive a value $(\texttt{exchange}, id, P_i, P_j, \alpha')$ from P_j. Then, do:
   (a) If $\alpha = \alpha' = \perp$ then choose $\kappa \overset{R}{\leftarrow} \{0,1\}^k$. Else, if $\alpha \neq \perp$ then set $\kappa = \alpha$; else set $\kappa = \alpha'$.
   (b) Send $(\texttt{key}, id, \kappa)$ to P_i and P_j, send $(\texttt{key}, id, P_i, P_j)$ to the adversary, and halt.

Figure 9: The Key Exchange functionality, $\mathcal{F}_{\text{KE}}$

On the feasibility of realizing $\mathcal{F}_{\text{KE}}$. The requirement that a protocol realizes $\mathcal{F}_{\text{KE}}$ is strictly stronger than other known definitions, such as those of [BR93, CK01]. Nonetheless, several natural and widely used protocols securely realize $\mathcal{F}_{\text{KE}}$. Furthermore, functionality $\mathcal{F}_{\text{KE}}$ can be relaxed so that securely realizing it becomes *equivalent* to the definition of [CK01]. See more details in [CK01a].

From KE to secure sessions. A formalization of secure channels and a proof that standard constructions of secure session protocols using key-exchange are secure is given in [CK01]. We restate the main result of [CK01], cast in the present framework.

Let MAC be a secure Message Authentication Code function, and let ENC be a symmetric encryption scheme that is semantically secure against chosen plaintext attacks. Let $\text{SS}_{\text{MAC,ENC}}$ be the following secure session protocol, that operates in the $\mathcal{F}_{\text{KE}}$-hybrid model. When activated with input $(\texttt{set-up}, id, P_j)$, P_i sends an $(\texttt{exchange}, id, P_i, P_j, \perp)$ request to $\mathcal{F}_{\text{KE}}$. When P_i gets the key κ from $\mathcal{F}_{\text{KE}}$, it derives (using a pseudorandom function) four keys $\kappa_e, \kappa_a, \kappa'_e, \kappa'_a$ from κ. (κ_e and κ_a will be used to encrypt and authenticate the traffic from P_i to P_j and the other two keys will be used to protect the traffic from P_j to P_i.) Next, upon receiving the input value $(\texttt{send}, id, m)$, P_i computes $c = \text{ENC}_{\kappa_e}(m)$, $a = \text{MAC}_{\kappa_a}(c, l)$, and sends (id, c, l, a) to P_j. Here l is a counter that is incremented for each message sent. When receiving a message (id, c, l, a), $\text{SS}_{\text{MAC,ENC}}$ within P_i first verifies that $a = \text{MAC}_{\kappa'_a}(c, l)$ and that no message with counter value l was previously received in this session. If so, then it locally outputs $\text{DEC}_{\kappa'_\epsilon}(c)$. Otherwise, it outputs nothing (or an error message). As soon as $\text{SS}_{\text{MAC,ENC}}$ is activated with a $(\texttt{terminate}, id)$ input, it erases all keys and local randomness used in this invocation of the protocol, and returns.

Claim 13 *Let* MAC *be a secure Message Authentication Code function, and let* ENC *be a symmetric encryption scheme that is semantically secure against chosen plaintext attacks. Then, protocol* $\text{SS}_{\text{MAC,ENC}}$ *securely realizes* $\mathcal{F}_{\text{SS}}$ *in the* $\mathcal{F}_{\text{KE}}$*-hybrid model with adaptive adversaries, in a model where data erasures are trusted.*

8.2 Public-key encryption and signatures

We present ideal functionalities that aim at capturing the desired functionality of public-key encryption and signature schemes when used as *tools* for constructing cryptographic protocols. In contrast to all other functionalities discussed in this work, these functionalities are *immediate*, as defined in Section 6. Specifically, whenever the functionality is activated with an input value generated by some party, it generates an output value only for the calling party. Furthermore, that output is delivered to the calling party within the same activation, without allowing the adversary to delay the delivery of the response. This structure of the encryption and signature functionalities is in line with the tradition of representing encryption and signature schemes as a tuple of non-interactive algorithms rather than as an interactive protocol. Indeed, the protocols that realize these functionalities will not involve any communication among the parties. They will consist of a set of non-interactive algorithms that respond to any input with locally computed output. The "interactive part", namely the use of encryption and signature schemes in interaction is left to the "high-level protocol" that makes use of the schemes.

For signature schemes, realizing the signature functionality in the presence of non-adaptive adversaries turns out to be essentially equivalent to existential unforgeability against chosen message attacks (as defined in [GMRi88]). For encryption schemes, realizing the public-key encryption functionality in the presence of non-adaptive adversaries has the flavor of security against adaptive chosen ciphertext attacks (CCA) [DDN00, RS91, BDPR98], while being less restrictive. It may be argued that the public-key encryption functionality captures the relevant security properties of public-key encryption in a protocol setting, while avoiding some technical restrictions imposed by the notion of CCA security. We exemplify this point by demonstrating that the public-key encryption functionality suffices for realizing a quintessential application of public-key encryption.

8.2.1 Digital Signatures

Digital signature schemes allow one party (the signer) to attach tags (signatures) to documents, so that everyone can verify, by locally running some public algorithm, that the signature was generated by no one else but the signer. Digital signature schemes were first proposed by Diffie and Hellman in [DH76]; their basic security requirements were formulated by Goldwasser, Micali and Rivest in [GMRi88].

Here we treat digital signatures as a cryptographic task within a larger environment, thus providing another view of the security of such schemes. We present the digital signature functionality in two steps. First we present a functionality, $\mathcal{F}_{\text{RSIG}}$, that may look more natural at first, but turns out to impose security requirements that are implied neither by the [GMRi88] notions no by our intuitive notion of unforgeable signatures. Next we present an alternative functionality, $\mathcal{F}_{\text{SIG}}$, which is a relaxation of $\mathcal{F}_{\text{RSIG}}$. We show that securely realizing $\mathcal{F}_{\text{SIG}}$ is essentially *equivalent* to existential unforgeability against chosen message attacks as in [GMRi88]. (Our main motivation in presenting both functionalities is to highlight the effect of relatively small changes in the formalization of ideal functionalities. Nonetheless, as seen below, functionality $\mathcal{F}_{\text{RSIG}}$ may be interesting in its own right.)

Functionalities $\mathcal{F}_{\text{RSIG}}$ and $\mathcal{F}_{\text{SIG}}$ are presented in Figures 10 and 11 below. As discussed in the preamble to Section 8.2, these functionalities are *immediate.* This represents the fact that signature and verification algorithms are typically run locally and provide output immediately, without waiting for any incoming messages from other parties. The two functionalities differ in Steps 2 and 3, namely in the signing and the verification operations. The difference in the signing operation is that, while $\mathcal{F}_{\text{RSIG}}$ generates the signature value at random from some distribution (and provides a unique signature for a message), $\mathcal{F}_{\text{SIG}}$ lets the adversary choose the signature value and allows for multiple signatures per message. This extra leniency of $\mathcal{F}_{\text{SIG}}$ represents the fact that the the common notion of signature schemes imposes no requirement on the distribution of the signature string itself. In particular, it can depend on the signed message (or, in fact, on all the messages signed in the past by the system) in some obvious way.

The difference in the verification process lies in the case where the message m was signed in the past but the signature provided by the verifier is not the one generated by the signer. Functionality

$\mathcal{F}_{\text{RSIG}}$ rejects that message. In contrast, $\mathcal{F}_{\text{SIG}}$ lets the adversary decide whether to accept or reject that signature. This extra leniency of $\mathcal{F}_{\text{SIG}}$ represents the fact that the [GMRi88] notion of security does not prevent the adversary from generating an additional signature to a message that was already signed in the past.

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

$\mathcal{F}_{\text{RSIG}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. In the first activation, expect to receive a value $(\texttt{signer}, id)$ from some party P_i. Then, send $(\texttt{signer}, id, P_i)$ to all parties and the adversary. From now on, ignore all $(\texttt{signer}, id)$ values.
2. Upon receiving a value $(\texttt{sign}, id, m)$ from P_i where m is a new message, choose a value $s_m \overset{\text{R}}{\leftarrow} \{0,1\}^k$, and register the pair (m, s_m) in memory. If m already appears in memory then let s_m be the value registered together with m. Send the message $(\texttt{signature}, id, m, s_m)$ to P_i and request the adversary to deliver this message immediately.
3. Upon receiving a value $(\texttt{verify}, id, m, \sigma)$ from some party P_j, let $f = 1$ if $\sigma = s_m$ and $f = 0$ otherwise. send $(\texttt{verified}, id, m, f))$ to P_j, and request the adversary to deliver this message immediately.

Figure 10: The first (strong) signature functionality, $\mathcal{F}_{\text{RSIG}}$.

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

$\mathcal{F}_{\text{SIG}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. In the first activation, expect to receive a value $(\texttt{signer}, id)$ from some party P_i. Then, send $(\texttt{signer}, id, P_i)$ to all parties and the adversary. From now on, ignore all $(\texttt{signer}, id)$ values.
2. Upon receiving a value $(\texttt{sign}, id, m)$ from P_i, hand $(\texttt{sign}, id, m)$ to the adversary. Upon receiving $(\texttt{signature}, id, m, \sigma)$ from the adversary, set $s_m = \sigma$, send $(\texttt{signature}, id, m, \sigma)$ to P_i, and request the adversary to deliver this message immediately. Save the pair (m, s_m) in memory.
3. Upon receiving a value $(\texttt{verify}, id, m, \sigma)$ from some party P_j, proceed as follows:
   (a) If m was never before signed then let $f = 0$. If m was signed before (i.e., s_m is defined) and $\sigma = s_m$ then let $f = 1$. If m was signed but $s_m \neq \sigma$ then hand $(\texttt{verify}, id, m, \sigma)$ to the adversary. Upon receiving $(\texttt{verified}, id, m, \phi)$ from the adversary let $f = \phi$.
   (b) Once the value of f is set, send $(\texttt{verified}, id, m, f)$ to P_j, and request the adversary to deliver this message immediately.

Figure 11: The second (relaxed) signature functionality, $\mathcal{F}_{\text{SIG}}$.

On $\mathcal{F}_{\text{RSIG}}$ and verifiable random functions. Notice that functionality $\mathcal{F}_{\text{RSIG}}$ realizes in fact a primitive that is somewhat stronger than verifiable random functions (VRF) [MRV99]. As in VRFs, there is a unique, publicly verifiable "signature" value associated with each message. As there, the value is pseudorandom from the point of view of the signature verifier. However, VRFs do not guarantee that the "signature value" appears random to the signer, whereas $\mathcal{F}_{\text{RSIG}}$ provides this guarantee.

On threshold signature schemes. Functionality $\mathcal{F}_{\text{SIG}}$ may be modified somewhat to capture the functionality of threshold signature schemes. See more details in [CHH00].

Equivalence with the [GMRi88] notion of security. We show that securely realizing $\mathcal{F}_{\text{SIG}}$ is equivalent to resilience against existential forgery by chosen message attacks as in [GMRi88]. First, we describe how to translate a signature scheme $S = (gen, sig, ver)$ as in [GMRi88] into a protocol π_S in the present setting. This is done as follows: When P_i, running π_S, receives an input (`signer`,id), it executes algorithm gen, keeps the signing key s and sends the verification key v to all parties. When the signer receives an input (`sign`,id, m), it sets $\sigma = sig(s, m)$ and outputs (`signature`,id, m, σ). When a party gets an input (`verify`,id, m, σ), it outputs (`verified`,$id, m, ver(v, m, \sigma)$).

Claim 14 *Let $S = (gen, sig, ver)$ be a signature scheme as in* [GMRi88]. *Then π_S securely realizes $\mathcal{F}_{\text{SIG}}$ with respect to non-adaptive adversaries if and only if S is existentially unforgeable against chosen message attacks.*

Proof (sketch): Let $S = (gen, sig, ver)$ be a signature scheme, and assume that π_S securely realizes F_{SIG}. We show that S is existentially unforgeable against chosen message attacks. That is, assume that there is a [GMRi88] forger G against S. We construct an environment $\mathcal{Z}$ and a (non-adaptive) real-life adversary $\mathcal{A}$ such that, for any ideal-process adversary $\mathcal{S}$, environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{F}_{\text{SIG}}$ and $\mathcal{S}$ in the ideal process, or with π_S and $\mathcal{A}$ in the real-life model.

Environment $\mathcal{Z}$ proceeds as follows. It first activates some party P_i with input (`signer`,id) for some value of id (say, $id = 0$). From now on, whenever $\mathcal{A}$ asks $\mathcal{Z}$ to sign a message m, $\mathcal{Z}$ will activate the signer with input (`sign`,id, m), and will report the output to $\mathcal{A}$. When $\mathcal{A}$ asks $\mathcal{Z}$ to verify a pair (m, σ), $\mathcal{Z}$ will first verify that m was never signed before. (if m was signed before then $\mathcal{Z}$ outputs 0 and halts.) Next, $\mathcal{Z}$ activates some uncorrupted party with input (`verify`,id, m, σ) and outputs whatever that party outputs.

The real-life adversary $\mathcal{A}$ will first wait to hear some party P_i send a public verification key v. (This will happen when $\mathcal{Z}$ activates P_i to be the signer.) Then, $\mathcal{A}$ will run G on input v. Whenever G generates a message m to be signed, $\mathcal{A}$ asks $\mathcal{Z}$ to sign m. When $\mathcal{A}$ hears the signature σ from $\mathcal{Z}$, it hands σ to G. When G outputs a pair (m^*, σ^*) (supposedly a new message and its forged signature), $\mathcal{A}$ asks $\mathcal{Z}$ to verify (m^*, s^*).

It can be seen that whenever G succeeds (i.e., whenever $ver(m^*, \sigma^*) = 1$ and m^* was not previously signed by P_i), $\mathcal{Z}$ outputs 1. Thus, under the assumption that G succeeds with non-negligible probability, in the real-life model $\mathcal{Z}$ outputs 1 with non-negligible probability. However, in the ideal process with $\mathcal{F}_{\text{SIG}}$ $\mathcal{Z}$ never outputs 1, regardless of what the ideal-process adversary does.

For the other direction, assume that there is a (non-adaptive) real-life adversary $\mathcal{A}$ such that for any ideal-process adversary $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can tell whether it is interacting with $\mathcal{F}_{\text{SIG}}$ and $\mathcal{S}$ in the ideal process, or with π_S and $\mathcal{A}$ in the real-life model. We construct a [GMRi88] forger G against S.

But first consider the following ideal-process adversary $\mathcal{S}$: $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and follows the instructions of $\mathcal{A}$, with the following exceptions. First, when $\mathcal{S}$ receives an output (signer,id, P_i) from $\mathcal{F}_{\text{SIG}}$, it runs the key generation algorithm *gen*, obtains a pair (s, v) of keys, and simulates for $\mathcal{A}$ a message that is sent from P_i to all parties and contains the verification key v. Next, whenever $\mathcal{S}$ receives (sign,id, m) from $\mathcal{F}_{\text{SIG}}$, it computes $\sigma = sig(s, m)$ and hands (signature,id, m, σ) back to $\mathcal{F}_{\text{SIG}}$. Whenever $\mathcal{S}$ receives (verify,id, m, σ) from $\mathcal{F}_{\text{SIG}}$, it hands (verified,$id, m, ver(v, m, \sigma)$) back to $\mathcal{F}_{\text{SIG}}$.[8]

Let B denote the event that, in a run of π_S, $ver(v, m, \sigma) = 1$ for some message m and signature σ, but during the signer never signed m the execution of the protocol. We are guaranteed that there exist a real-life adversary $\mathcal{A}$ and environment $\mathcal{Z}$ that distinguishes between its run with $\mathcal{A}$ in the real life model and its run with $\mathcal{S}$ in the ideal process. However, as long as event B does not occur, the two interactions is identical from the point of view of $\mathcal{Z}$. Thus, we are guaranteed that in the real-life model event B occurs with non-negligible probability.

We turn to constructing the [GMRi88] forger G. G operates as $\mathcal{S}$ does, with the following exceptions. Like $\mathcal{S}$, G interacts with simulated copies of $\mathcal{A}$ and $\mathcal{Z}$. However, instead of running *gen* to obtain the keys (s, v), G will hand $\mathcal{A}$ the public verification key v in its input. Instead of running the signing algorithm to obtain $\sigma = sig(s, m)$, G will ask its oracle to sign m and will thus obtain the signature σ. Whenever the simulated $\mathcal{Z}$ activates some uncorrupted party with input (verify,id, m, σ), G verifies that m was never signed before and $ver(v, m, \sigma) = 1$. Once such a pair (m, σ) is found, G outputs that pair and halts. (Here it is important that $\mathcal{A}$ is non-adaptive. In particular, G may not be able to mimic the behavior of $\mathcal{S}$ in the case where $\mathcal{A}$ corrupts the signer.)

Notice that, from the point of view of $\mathcal{A}$ and $\mathcal{Z}$, the interaction with G looks the same as an interaction in the real-life model with π_S. Thus, we are guaranteed that event B (and, thus, successful forgery by G) will occur with non-negligible probability. □

8.2.2 Public-Key Encryption

One use of public-key encryption is for obtaining protocols for secure message transmission (i.e., protocols for realizing $\mathcal{F}_{\text{SMT}}$) as described in Section 8.1.2. However, that application uses each public key for encrypting only a single message, thus it does not capture in full the security properties of public-key encryption. This section proposes an ideal functionality, $\mathcal{F}_{\text{PKE}}$, that is aimed at capturing the security requirements from public-key encryption schemes when used as tools within cryptographic protocols. In particular, the functionality guarantees the secrecy of encrypted messages in a setting where the same public key is used to encrypt multiple messages by different parties, and where the adversary may obtain the output of the decryption algorithm on inputs of its choice.

We show that any encryption scheme that is secure against adaptive chosen ciphertext attack (CCA) [DDN00, RS91, BDPR98] can be turned in a natural way into a protocol that securely realizes $\mathcal{F}_{\text{PKE}}$ (in the presence of non-adaptive adversaries). We note that the converse does not hold: there exist encryption schemes that are not CCA-secure, and nonetheless result in a protocol that securely realizes $\mathcal{F}_{\text{PKE}}$.

Next, we demonstrate the adequacy of $\mathcal{F}_{\text{PKE}}$ for capturing the security properties of public-key encryption by formalizing an ideal functionality, $\mathcal{F}_{\text{M-SMT}}$, that captures a salient application of public-key encryption, and presenting a simple protocol that realizes $\mathcal{F}_{\text{M-SMT}}$ in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{PKE}})$-hybrid model. Essentially, functionality $\mathcal{F}_{\text{M-SMT}}$ allows multiple parties to send multiple messages

[8]In the construction of $\mathcal{S}$ we assumed that the signer P_i is not corrupted (or, rather, that the environment activates an uncorrupted party to be the signer). If the signer is corrupted then $\mathcal{S}$ simply follows the instructions of $\mathcal{A}$. In this case, the environment's view in the ideal process is identical to its view in the real-life model.

to a single recipient, while guaranteeing the secrecy and integrity of each individual message.

Functionality $\mathcal{F}_{\text{PKE}}$ is presented in Figure 12. Recall that this functionality is *immediate*, as discussed in the preamble of Section 8.2. That is, whenever it hands a value to some party it requests that this value be delivered immediately. As in the case of $\mathcal{F}_{\text{SIG}}$, functionality allows the ideal-process adversary to choose the output values, subject to the restrictions imposed by the desired security properties. Specifically, the ideal-process adversary can choose the value of the public key, the value of each ciphertext (as long as ciphertexts are unique), and the output of the decryption algorithms on ciphertexts that were not legitimately generated by using the functionality. Still, $\mathcal{F}_{\text{PKE}}$ guarantees that the adversary learns nothing on the legitimately encrypted plaintexts, and that the legitimately generated ciphertexts are decrypted correctly. Note that $\mathcal{F}_{\text{PKE}}$ does not guarantee the secrecy of messages that are encrypted with public keys that are different than the one generated by the decrypting party. Also, the secret decryption key is not included in any output of the functionality. It is treated as a secret value that is implicitly kept by the decrypting party.

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

$\mathcal{F}_{\text{PKE}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. In the first activation, expect to receive a value $(\texttt{KeyGen}, id)$ from some party P_i. Then, do:
   (a) Hand $(\texttt{KeyGen}, id)$ to the adversary.
   (b) Receive a value e from the adversary, and hand e to P_i.
2. Upon receiving a value $(\texttt{Encrypt}, id, e', m)$ from some party P_j, proceed as follows:
   (a) Hand $(\texttt{Encrypt}, id, e', |m|)$ to the adversary, where $|m|$ denotes the length of m. (If $e' \neq e$ or e is not yet defined then hand also the entire value m to the adversary.)
   (b) Receive a tag c from the adversary and hand c to P_j. In addition, if $e' = e$ then store the pair (c, m). (If the tag c already appears in a previously stored pair then halt.)
3. Upon receiving a value $(\texttt{Decrypt}, id, c)$ from P_i (and P_i only), proceed as follows:
   (a) If there is a pair (c, m) stored in memory then hand m to P_i.
   (b) Otherwise, hand the value $(\texttt{Decrypt}, id, c)$ to the adversary, receive a value m from the adversary, and hand m to P_i.

Figure 12: The public-key encryption functionality, $\mathcal{F}_{\text{PKE}}$

On $\mathcal{F}_{\text{PKE}}$ and CCA security. Let $S = (gen, enc, dec)$ be an encryption scheme. (Recall that *gen* is the key generation algorithm, *enc* is the encryption algorithm and *dec* is the decryption algorithm.) Very loosely, S is said to be secure against adaptive chosen ciphertext attacks (or, CCA-secure) if no attacker F can succeed in the following experiment. Algorithm *gen* is run to generate an encryption key e and a decryption key d. F is given e and access to a decryption oracle $dec(d, \cdot)$. When F generates a pair m_0, m_1 of messages, a bit $b \overset{\text{R}}{\leftarrow} \{0, 1\}$ is chosen and F is given $c = enc(e, m_b)$. From this point on, F may continue querying its oracle, under the condition that it does not ask for a decryption of c. F succeeds if it guesses b with probability that is non-negligibly more than one half. See more details in [DDN00, RS91, BDPR98].

Consider the following transformation from an encryption scheme S to a protocol π_S that is geared towards realizing $\mathcal{F}_{\text{PKE}}$:

1. When activated, within some P_i and with input $(\texttt{KeyGen}, id)$, run algorithm gen, obtain the encryption key e and decryption key d, and output e.

2. When activated, within some party P_j and with input $(\texttt{Encrypt}, id, e', m)$, return $enc_{e'}(m, r)$ for a randomly chosen r. (Note that it does not necessarily hold that $e' = e$.)

3. When activated, within P_i and with input $(\texttt{Decrypt}, id, c)$, return $dec_d(c)$.

Demonstrating security of π_S with respect to adaptive adversaries is bound to run into problems similar to the ones discussed in [CFGN96], thus known techniques would only allow encryption protocols where the public encryption key is longer than the total number of messages encrypted throughout the lifetime of the system. (This is true even if data erasures are allowed.) We thus concentrate on the non-adaptive case, where the set of corrupted parties is fixed in advance. We show:

Claim 15 *Let S be a CCA-secure encryption scheme. Then π_S securely realizes $\mathcal{F}_{\text{PKE}}$ with respect to non-adaptive adversaries.*

Proof: Let $S = (gen, enc, dec)$ be a CCA-secure encryption scheme, and let π_S be the protocol constructed from S as described above. We show that π_S securely realizes $\mathcal{F}_{\text{PKE}}$. Using the alternative definition of security (Definition 4 on page 22), we construct an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it interacts with $\mathcal{F}_{\text{PKE}}$ and $\mathcal{S}$ in the ideal process or with parties running π_S and the dummy adversary $\tilde{\mathcal{A}}$ in the real-life model.

Recall that $\tilde{\mathcal{A}}$ takes three types of messages from $\mathcal{Z}$: either to corrupt parties, or to report on messages sent in the protocol, or to deliver some message. However, since we are dealing with non-adaptive adversaries, there are no party corruption instructions. Furthermore, since protocol π_S involves no sending of messages, there are no requests to report on or deliver messages. In fact, there is no communication between $\mathcal{Z}$ and $\tilde{\mathcal{A}}$ at all. In fact, the only way in which $\mathcal{S}$ can affect the view of $\mathcal{Z}$ is by communicating with the ideal functionality $\mathcal{F}_{\text{PKE}}$. Consequently, adversary $\mathcal{S}$ proceeds as follows.

1. When $\mathcal{S}$ receives a message $(\texttt{KeyGen}, id)$ from $\mathcal{F}_{\text{PKE}}$, it runs the key generation algorithm gen, obtains the encryption key e and the decryption key d, and returns e to $\mathcal{F}_{\text{PKE}}$.

2. When $\mathcal{S}$ receives a message $(\texttt{Encrypt}, id, e', |m|)$ from $\mathcal{F}_{\text{PKE}}$, it first verifies that $e' = e$. If so, then it computes $c = enc_e(0^{|m|}, r)$ for a random r and returns c to $\mathcal{F}_{\text{PKE}}$. If $e' \neq e$ then $\mathcal{S}$ receives also the full value m from $\mathcal{F}_{\text{PKE}}$. In this case it returns $enc_{e'}(m, r)$ to $\mathcal{F}_{\text{PKE}}$.

3. When $\mathcal{S}$ receives a message $(\texttt{Decrypt}, id, c)$ from $\mathcal{F}_{\text{PKE}}$, it obtains $m = dec_d(c)$ and returns m to $\mathcal{F}_{\text{PKE}}$.

Analyzing $\mathcal{S}$, assume for contradiction that there is an environment $\mathcal{Z}$ that distinguishes between the real and ideal interactions. We use $\mathcal{Z}$ to construct an adversary $\mathcal{F}$ that breaks the CCA security of the encryption scheme S. More precisely, assume that for some value of the security parameter k we have $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k) - \text{REAL}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}(k) > \epsilon$. We show that $\mathcal{F}$ guesses the bit b correctly in the CCA game with probability $1/2 + e/2l$, where l is the total number of messages that were encrypted throughout the run of the system. (Without loss of generality, we assume that in each execution of the protocol $\mathcal{Z}$ asks to encrypt exactly l messages.)

Adversary $\mathcal{F}$ proceeds as follows, given a public key e, and having access to a decryption oracle D and an encryption oracle E. $\mathcal{F}$ first randomly chooses a number $h \stackrel{R}{\leftarrow} \{1,...,l\}$. Next, $\mathcal{F}$ runs $\mathcal{Z}$ on the following simulated interaction with a system running π_S (and the dummy adversary $\tilde{\mathcal{A}}$). Let m_i denote the ith message that $\mathcal{Z}$ asks to encrypt in an execution.[9]

1. When $\mathcal{Z}$ activates some party P_i with input $(\texttt{KeyGen}, id)$, $\mathcal{F}$ lets P_i output the value e from $\mathcal{F}$'s input.
2. For the first $h-1$ times that $\mathcal{Z}$ asks to encrypt some message, m_i, $\mathcal{F}$ lets the encrypting party return $c_i = enc_e(m_i)$.
3. At the hth time that $\mathcal{Z}$ asks to encrypt a message, m_h, $\mathcal{F}$ queries its encryption oracle with the pair of messages $(m_h, 0^{|m_h|})$, and obtains the test ciphertext c_h. It then hands c_h to $\mathcal{Z}$ as the encryption of m_h.
4. For the remaining $l-h$ times that $\mathcal{Z}$ asks to encrypt some message, m_i, $\mathcal{F}$ lets the encrypting party return $c_i = enc_e(0^{|m_i|})$.
5. Whenever the decryptor P_i is activated with input $(\texttt{Decrypt}, id, c)$ where $c = c_i$ for some i, $\mathcal{F}$ lets P_i return the corresponding plaintext m_i. (This holds for the case $i = h$ as well as $i \neq h$.) If c is different from all the c_i's then $\mathcal{F}$ queries its decryption oracle on c, obtains a value v, and lets P_i return v to $\mathcal{Z}$.
6. When $\mathcal{Z}$ halts, $\mathcal{F}$ outputs whatever $\mathcal{Z}$ outputs and halts.

Analyzing the success probability of $\mathcal{F}$ is done via a standard hybrids argument. Let the random variable H_i denote the output of $\mathcal{Z}$ from an interaction that is identical to an interaction with $\mathcal{S}$ in the ideal process, with the exception that the first i ciphertexts are computed as an encryption of the real plaintexts, rather than encryptions of 0^*.

It is easy to see that H_0 is identical to the output of $\mathcal{Z}$ in the ideal process, and H_l is identical the output of $\mathcal{Z}$ in the real-life model. (This follows from the fact that the scheme S guarantees that $dec_d(enc_e(m)) = m$.) Furthermore, in a run of $\mathcal{F}$, if the value c_h that $\mathcal{F}$ obtains from its encryption oracle is an encryption of m_h then the output of the simulated $\mathcal{Z}$ has the distribution of H_{h-1}. If c_h is an encryption of $0^{|m_h|}$ then then the output of the simulated $\mathcal{Z}$ has the distribution of H_h. The claim follows. □

On the necessity of CCA security. It is interesting to note that CCA security of the underlying encryption scheme is not *necessary* for realizing $\mathcal{F}_{\text{PKE}}$. That is, there exist encryption schemes S that are not CCA-secure and yet π_S securely realizes $\mathcal{F}_{\text{PKE}}$. For instance, let S be a CCA-secure encryption scheme, and construct S' from S as follows. The key generation algorithm remains unchanged. The encryption algorithm of S' first runs the encryption algorithm of S and then appends 1 to the ciphertext. The decryption algorithm of S' first deletes the last bit from the received ciphertext, and then runs the decryption algorithm of S. It is easy to see that S' is no longer CCA-secure; nonetheless, $\pi_{S'}$ securely realizes $\mathcal{F}_{\text{PKE}}$ all the same. Indeed, for all practical purposes scheme S' (and protocol π'_S) are just as secure as the original scheme S. (In other words, the differene between S and S' appears to be of no security consequence for the purpose of guaranteeing the secrecy of encrypted data.) We remark that a similar phenomenon occurs in the case of secure sessions and symmetric encryption. See more details in [K01, AB01].

[9]Without loss of generality we assume that $\mathcal{Z}$ only asks to encrypt messages with the public key e that was generated by the decrypting party. Indeed, when $\mathcal{Z}$ asks to encrypt a message m with a public key $e' \neq e$, it receives a value $c = enc_{e'}(m, r)$ that it can compute by itself.

Using public-key encryption for secure communication. Functionality $\mathcal{F}_{\text{PKE}}$ is not aimed at capturing the security requirements of some given application or protocol problem. Rather, it treats public-key encryption as a technical tool to be used within cryptographic protocols. In the rest of this section we demonstrate the use of this technical tool by formulating an ideal functionality that captures a salient application of public-key encryption, and demonstrating how this functionality can be realized with the aid of $\mathcal{F}_{\text{PKE}}$.

The functionality, called $F_{\text{M-SMT}}$, allows multiple parties to send multiple messages to a single recipient, while guaranteeing the secrecy and integrity of each encrypted message. It is presented in Figure 13.

Functionality $\mathcal{F}_{\text{M-SMT}}$

$\mathcal{F}_{\text{M-SMT}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. In the first activation, expect to receive a value $(\texttt{receiver}, id)$ from some party P_i. Then, send $(\texttt{receiver}, id, P_i)$ to all parties and the adversary. From now on, ignore all $(\texttt{receiver}, id)$ values.
2. Upon receiving a value $(\texttt{send}, id, m)$ from some party P_j, send (id, P_j, m) to P_i and $(id, P_j, |m|)$ to the adversary.

Figure 13: The multiple secure message transmission functionality, $\mathcal{F}_{\text{M-SMT}}$

Note that functionality $\mathcal{F}_{\text{M-SMT}}$ does not deal with the delivery of the session ID id to parties that may want to send encrypted data to the receiver. This task is left to the higher-level protocol that uses $\mathcal{F}_{\text{M-SMT}}$. We also remark that functionality $\mathcal{F}_{\text{M-SMT}}$ can be extended in a natural way to represent the security requirements of threshold decryption schemes. See more details in [CG99].

We show a simple protocol that securely realizes $\mathcal{F}_{\text{M-SMT}}$ in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{PKE}})$-hybrid model. That is, on top of assuming ideal access to $\mathcal{F}_{\text{PKE}}$, we assume ideally authenticated communication. (As seen below, the use of authenticated communication is essential for this solution.) Consider the following protocol, σ. When invoked with input $(\texttt{receiver}, id)$ within party P_i, protocol σ invokes $\mathcal{F}_{\text{PKE}}$ with input $(\texttt{KeyGen}, id)$, obtains the public key e and sends (id, e) to all parties.[10] When activated within party P_j with incoming message $(P_i, P_j, (id, e))$ from $\mathcal{F}_{\text{AUTH}}$, party P_j records the triple (P_i, id, e). ¿From now on, when activated within P_j with input $(\texttt{send}, id, m)$, the protocol invokes $\mathcal{F}_{\text{PKE}}$ with input $(\texttt{Encrypt}, id, (P_j, e, m))$, obtains a ciphertext c, and invokes $\mathcal{F}_{\text{AUTH}}$ to send (id, c) from P_j to P_i. Finally, when activated within P_i with incoming message $(\texttt{ciphertext}, id, c)$ from P_j, the protocol invokes $\mathcal{F}_{\text{PKE}}$ with input $(\texttt{Decrypt}, id, c)$, obtains the decryption m', and verifies that m' is of the form $m' = (P_j, m)$. If the verification succeeds, then the protocol outputs m. Otherwise it outputs nothing.

We stress that the sender incorporates its own identity in the encrypted message, and that the receiver verifies that the identity in the decrypted message agrees with the identity of the sender. This precaution is used to prevent copying of messages: Without it, a corrupted party could copy an ciphertext sent by an uncorrupted party, and re-send it as coming from itself. Such copying of messages is not allowed by $\mathcal{F}_{\text{M-SMT}}$. We show:

[10]Sending the public key to all parties is done via $\mathcal{F}_{\text{AUTH}}$. That is, for each party P_j the protocol invokes $\mathcal{F}_{\text{AUTH}}$ with value $(P, P_j, ((id, e))$. This is an over-simplified way to implement authenticated distribution of public keys. In reality, more complex mechanisms would be used. These mechanisms will need to be represented via different ideal functionalities.

Claim 16 *Protocol σ securely realizes $\mathcal{F}_{\text{M-SMT}}$ in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{PKE}})$-hybrid model.*

Proof (sketch): Let $\mathcal{A}$ be an adversary that operates against parties running σ in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{PKE}})$-hybrid model. We construct an ideal process adversary $\mathcal{S}$ such that no environment can tell with non negligible probability whether it is interacting with $\mathcal{A}$ and σ in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{PKE}})$-hybrid model or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{M-SMT}}$. (See Definition 7.) Adversary $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$, and proceeds as follows.

1. Whenever $\mathcal{S}$ is activated with an input value v from $\mathcal{Z}$, it activates the simulated $\mathcal{A}$ with input v. Whenever $\mathcal{A}$ generates an output value v, $\mathcal{S}$ writes v on its output tape.

2. *Key generation:* Initialize a flag $g = 0$. Then:

   (a) *Uncorrupted decryptor:* When $\mathcal{S}$ receives a value $(\texttt{receiver}, id, P_i)$ from $\mathcal{F}_{\text{M-SMT}}$, it checks the value of g. If $g = 1$ then $\mathcal{S}$ does nothing. If $g = 0$ then $\mathcal{S}$ sets $g = 1$ and activates $\mathcal{A}$ with input $(\texttt{KeyGen}, id)$ coming from $\mathcal{F}_{\text{PKE}}$. When $\mathcal{A}$ returns a value e, $\mathcal{S}$ starts a series of activations of $\mathcal{A}$: for each party P_j, $\mathcal{S}$ activates $\mathcal{A}$ with a value $(P_i, P_j, (id, e))$ coming from $\mathcal{F}_{\text{AUTH}}$. (Recall that this value means that P_i sent a message (id, e) to P_j.

   (b) *Corrupted decryptor:* When the simulated $\mathcal{A}$ activates $\mathcal{F}_{\text{PKE}}$ with input $(\texttt{KeyGen}, id)$ in the name of a corrupted party P_i, $\mathcal{S}$ checks the value of g. If $g = 1$ then $\mathcal{S}$ does nothing. If $g = 0$ then $\mathcal{S}$ sets $g = 1$, returns $(\texttt{KeyGen}, id)$ to $\mathcal{A}$, and obtains a value e from $\mathcal{A}$. Next, $\mathcal{S}$ has P_i send a value $(\texttt{receiver}, id, P_i)$ to $\mathcal{F}_{\text{M-SMT}}$. Finally, when the simulated $\mathcal{A}$ activates $\mathcal{F}_{\text{AUTH}}$ with a value $(P_i, P_j, (id, e))$, $\mathcal{S}$ activates $\mathcal{A}$ with a value $(P_i, P_j, (id, e))$ coming from $\mathcal{F}_{\text{AUTH}}$.

3. *Delivery of the public key:* When the simulated $\mathcal{A}$ activates some uncorrupted party P_j to receive a value $(P_i, P_j, (id, e'))$ coming from $\mathcal{F}_{\text{AUTH}}$, $\mathcal{S}$ activates P_j to receive the message $(\texttt{receiver}, id, P_i)$ from $\mathcal{F}_{\text{M-SMT}}$.

4. *Encryption by an uncorrupted party:* When $\mathcal{S}$ receives a value (id, P_j, l) from $\mathcal{F}_{\text{M-SMT}}$, it checks whether the value e' that P_j received in Step 3 equals the value e obtained in Step 2. If $e' \neq e$ then $\mathcal{S}$ does nothing. (The rationale is that in this case P_i is bound to reject the message and output nothing.) If $e' = e$ then $\mathcal{S}$ activates $\mathcal{A}$ with input $(\texttt{Encrypt}, id, e, l')$ coming from $\mathcal{F}_{\text{PKE}}$, where l' is l plus the length of the identifier P_j. When $\mathcal{A}$ returns a value c, $\mathcal{S}$ records the pair (c, m) and activates $\mathcal{A}$ with a value $(P_j, P_i, (id, c))$ coming from $\mathcal{F}_{\text{AUTH}}$.

5. *Encryption by a corrupted party:* When the simulated $\mathcal{A}$ activates $\mathcal{F}_{\text{PKE}}$ with input $(\texttt{Encrypt}, id, e', m')$ in the name of some corrupted party P_i, $\mathcal{S}$ returns $(\texttt{Encrypt}, id, e', |m'|)$ to $\mathcal{A}$ and obtains c from $\mathcal{A}$. It then records the pair (c, m'). If $m' = (P_j, m)$ then $\mathcal{S}$ activates $\mathcal{F}_{\text{M-SMT}}$ with a value $(\texttt{send}, id, m)$ from P_j. (If $e' \neq e$ then $\mathcal{S}$ hands $\mathcal{A}$ also the value m, and does not record the pair (c, m').) When the simulated $\mathcal{A}$ activates $\mathcal{F}_{\text{AUTH}}$ with a value $(P_j, P_i, (id, c))$, $\mathcal{S}$ activates $\mathcal{A}$ with the value $(P_j, P_i, (id, c))$ sent from $\mathcal{F}_{\text{AUTH}}$.

6. *Decryption by an uncorrupted decryptor:* When $\mathcal{A}$ invokes party P_i to receive a message $(P_j, P_i, (id, c))$ from P_j, $\mathcal{S}$ verifies that a pair (c, m') is recorded, and that $m' = (m, P_j)$. If so, then $\mathcal{S}$ activates P_i to receive a value $(\texttt{send}, id, m)$ from P_j. Otherwise, $\mathcal{S}$ does nothing.

7. *Decryption by a corrupted decryptor:* When the simulated $\mathcal{A}$ activates $\mathcal{F}_{\text{PKE}}$ with input $(\texttt{Decrypt}, id, c)$, $\mathcal{S}$ behaves as $\mathcal{F}_{\text{PKE}}$ would. That is, if the pair (c, m) was recorded for some

m then $\mathcal{S}$ hands m to $\mathcal{A}$. Otherwise, $\mathcal{S}$ hands $(\texttt{Decrypt}, id, c)$ to $\mathcal{A}$, obtains a value m, and returns m to $\mathcal{A}$.

It can be verified that the view of $\mathcal{Z}$ in an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{M-SMT}}$ is distributed *identically* to the view of $\mathcal{Z}$ in an interaction with $\mathcal{A}$ and σ in the $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{PKE}})$-hybrid model. This holds even for computationally unbounded $\mathcal{Z}$. □

On the modeling of Pfitzmann and Waidner [PW01]**.** Another modeling of the security provided by public-key encryption is given in [PW01]. Their modeling is similar to that of functionality $\mathcal{F}_{\text{M-SMT}}$ (Figure 13), with the exceptions that they allow the adversary to "replay" old ciphertexts and that they deal with many receivers within the same copy of their functionality. (In contrast, functionality $\mathcal{F}_{\text{M-SMT}}$ does not allow replay, and handles only a single receiver. The case of many receivers is taken care of by the composition theorem.)

Pfitzmann et.al. also propose a protocol for securely realizing their functionality, for the case of non-adaptive adversaries. In contrast to the modular treatment here, they implement protocols directly in the bare unauthenticated-links model. Their protocol calls for a public-key infrastructure, and in addition for signing each message together with the identity of the sender and receiver, and then encrypting it using a CCA-secure encryption scheme.

8.3 Two-party functionalities

We consider four popular two-party tasks: Generating common randomness, Commitment, Zero Knowledge, and Oblivious Transfer. These tasks are treated as tasks for two parties in a larger multiparty network. That is, security should be guaranteed even when many pairs of parties may be executing multiple copies of these protocols concurrently, without being aware of other potential protocol executions. (Indeed, this setting raises numerous security concerns that are not relevant when running two-party protocols in a stand-alone setting. For instance, in addition to malleability and concurrency issue, here the case where both communicating parties are eventually corrupted, perhaps adaptively, becomes relevant.)

Each of these tasks is captured via an appropriate ideal functionality. Let us shortly summarize the results described below regarding realizability of these functionalities:

1. All four functionalities can be realized in the server-assisted setting of Section 7.
2. None of these functionalities can be realized in the bare model (and not even in the AM) by protocols that involve only two parties.[11]
3. The Common Randomness functionality, $\mathcal{F}_{\text{CR}}$, turns out to be instrumental for realizing other functionalities. Specifically:
   (a) The commitment functionality, $\mathcal{F}_{\text{COM}}$, can be securely realized in the $\mathcal{F}_{\text{CR}}$-hybrid model [CF01].
   (b) The Zero-Knowledge functionality, $\mathcal{F}_{\text{ZK}}$, can be securely realized in the $\mathcal{F}_{\text{COM}}$-hybrid model [CF01].
   (c) *Any* two-party functionality can be realized by a two-party protocol in the $\mathcal{F}_{\text{ZK}}$-hybrid model, under standard cryptographic assumptions [CL01].

[11]There is a technical "wrinkle" here: The protocol that generates no messages and no output securely realizes any functionality. We thus restrict attention to protocols that guarantee that parties generate output whenever the adversary delivers all messages sent by the parties and only those messages.

We note that the $\mathcal{F}_{\text{CR}}$-hybrid model coincides with the useful "common random string" model of [BFM89].

8.3.1 Common Randomness

The task of "tossing a common coin" is one of the first and more fundamental protocol problems considered in the literature (see, e.g., [B82]). In its simplest form, the task calls for two mutually distrustful parties to generate a common unbiased random bit. It can be naturally generalized to more than two parties and to having the parties generate a string drawn from an arbitrary (samplable) pre-determined distribution. It can also be somewhat relaxed by allowing the generated string to drawn from some distribution that is adversarially chosen from some family of distribution that satisfy a certain set of conditions.

This task is captured by functionality $\mathcal{F}_{\text{CR}}$, described in Figure 14. For sake of simplicity, $\mathcal{F}_{\text{CR}}$ does not limit the number or identities of parties that obtain the common random value d. It can be modified in straightforward ways in order to incorporate such restrictions.

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

$\mathcal{F}_{\text{CR}}$ proceeds as follows, when parameterized by a distribution D.

1. When activated for the first time on input $(\mathtt{value}, id)$, choose a value $d \overset{\text{R}}{\leftarrow} D$ and send d back to the activating party. In each other activation send the value d to the activating party.

Figure 14: The Common Randomness functionality

Notice that the $\mathcal{F}_{\text{CR}}$-hybrid model coincides with the "common random string model," of [BFM89]. (When D is the uniform distribution this is exactly the [BFM89] model. When D is another distribution than this is the more general "common reference string model". Specifically:

- In the real-life model of computation the parties have access to a common and public string that is chosen in advance according to some distribution (specified by the protocol run by the parties).
- In the ideal process for some functionality (say, for $\mathcal{F}_{\text{COM}}$ defined below), the ideal process adversary that interacts with the environment may "play the role of $\mathcal{F}_{\text{CR}}$" for the environment. This means that the ideal process adversary may choose the common string in any way it wishes (and may in particular keep trapdoor information regarding the generated string).[12]

Unfortunately, $\mathcal{F}_{\text{CR}}$ cannot be realized in the AM. More precisely, say that a protocol is terminating if all participants generate output whenever the adversary corrupts no party, delivers all messages sent by the parties, and only those messages.

Claim 17 *There do not exist terminating two-party protocols that realize $\mathcal{F}_{\text{CR}}$ in the* AM.

[12]In fact, the present re-casting of the common random string model as the $\mathcal{F}_{\text{CR}}$-hybrid model may provide further justification for this salient property of the model: Consider a protocol that realizes some ideal functionality (say, $\mathcal{F}_{\text{COM}}$) in the $\mathcal{F}_{\text{CR}}$-hybrid model. Since the ideal process for $\mathcal{F}_{\text{COM}}$ makes no use of the random string, its validity is not affected by the fact that the protocol runs in the $\mathcal{F}_{\text{CR}}$-hybrid model. We are thus guaranteed that our notion of security and the functionality of $\mathcal{F}_{\text{CR}}$ remains valid whenever the parties are given ideal access to a common and public random string.

Proof: This is a corollary of Claim 18 and the protocols of [CF01] that securely realize $\mathcal{F}_{\text{COM}}$ in the $\mathcal{F}_{\text{CR}}$-hybrid model. □

8.3.2 Commitment[13]

Informally, commitment is a two-party protocol that has two phases: a commit phase, where the receiver of the commitment obtains some information which amounts to a "commitment" to an unknown value, and a reveal phase, where the receiver obtains an "opening" of the commitment to some value, and verifies whether the opening is valid. Roughly speaking, the security guarantee is that once the commit phase ends, there is only a single value that the receiver will consider as a valid opening (and, of course, that an honest committer can convince an honest receiver in the validity of an opening).

Figuratively, we envision that the committer provides the receiver with the digital equivalent of a "sealed envelope" containing a value x. From this point on, the committer cannot change the value inside the envelope, and, as long as the committer does not assist the receiver in opening the envelope, the receiver learns nothing about x. When both parties cooperate, the value x is retrieved in full.

In the present formalization, commitment protocols are protocols that securely realize the following "ideal commitment functionality," denoted $\mathcal{F}_{\text{COM}}$. The commitment phase is modeled by having $\mathcal{F}_{\text{COM}}$ receive a value $(\texttt{Commit}, id, P_i, P_j, x)$, from some party P_i (the committer). Here id is an Session ID (SID), P_j is the identity of another party (the receiver), and x is the value committed to. In response, $\mathcal{F}_{\text{COM}}$ lets the receiver P_j *and the adversary* $\mathcal{S}$ know that P_i has committed to some value, and that this value is associated with SID id. This is done by sending the message $(\texttt{Receipt}, id, P_i, P_j)$ to P_j and $\mathcal{S}$. The opening phase is initiated by the committer sending a value $(\texttt{Open}, id, P_i, P_j)$ to $\mathcal{F}_{\text{COM}}$. In response, $\mathcal{F}_{\text{COM}}$ hands the value $(\texttt{Open}, id, P_i, P_j, x)$ to P_j and $\mathcal{S}$. Functionality $\mathcal{F}_{\text{COM}}$ is presented in Figure 15 below.

We stress that functionality $\mathcal{F}_{\text{COM}}$ corresponds to only a single invocation of a commitment protocol. The composition theorem guarantees that a protocol that securely realizes this functionality will be secure even in the multi-instance cases and in conjunction with any application. This in particular implies that known security requirements, such as non-malleability [DDN00] and security for selective decommitment [DNRS99], are met.

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

$\mathcal{F}_{\text{COM}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value $(\texttt{Commit}, id, P_i, P_j, x)$ from P_i, send the value $(\texttt{Receipt}, id, P_i, P_j)$ to P_j and $\mathcal{S}$. (Ignore any subsequent `Commit` messages.)
2. Upon receiving a value $(\texttt{Open}, id, P_i, P_j)$ from P_i, proceed as follows: If a value $(\texttt{Commit}, id, P_i, P_j, x)$ was previously received from P_i then send the value $(\texttt{Open}, id, P_i, P_j, x)$ to P_j and $\mathcal{S}$. Otherwise, do nothing.

Figure 15: The Ideal Commitment functionality, $\mathcal{F}_{\text{COM}}$

Note that F_{COM} reveals the decommitted value to the adversary, thus capturing the standard assumption that the decommitted value is public. The concern that the opening of the commitment

[13]This section is taken from [CF01] and is included here for self-containment only.

should be available only to the receiver can be captured by modifying $\mathcal{F}_{\text{COM}}$ so that the adversary does not receive the opened value x.

On the feasibility of realizing $\mathcal{F}_{\text{COM}}$. We summarize the relevant results in [CF01]. Their first result is negative:

Claim 18 ([CF01]) *There do not exist two-party protocols that realize $\mathcal{F}_{\text{COM}}$ in the* AM.

Next, they show that in the $\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{CR}}$-hybrid model one can realize $\mathcal{F}_{\text{COM}}$ if trapdoor permutations exist. Their protocol is non-interactive, in the sense that both the commitment and the opening phases consist of a single message sent form the committer to the receiver.

Moreover, they extended the result as follows. As mentioned in Section 8.3.2, in the $\mathcal{F}_{\text{CR}}$-hybrid model the composition theorem works only if each copy of the protocol for realizing the functionality in question ($\mathcal{F}_{\text{COM}}$, in our case) uses a different instance of $\mathcal{F}_{\text{CR}}$. This means that the overall size of the common string must be at least linear in the number of applications of the commitment protocol. In order to benefit from the universal composability property while using only a short common string, they define a variant of $\mathcal{F}_{\text{COM}}$, called $\mathcal{F}_{\text{MCOM}}$, that handles multiple commitment and decommitment operations within the same instance. This means that a single instance of a protocol that realizes $\mathcal{F}_{\text{MCOM}}$ can handle multiple commitment "sessions" based on a single, short, common string. They construct a non-interactive protocol that securely realizes $\mathcal{F}_{\text{MCOM}}$ assuming existence of claw-free trapdoor permutations.

8.3.3 Zero Knowledge

Zero-Knowledge [GMRa89] is a task for two parties, a prover and a verifier, that have a binary relation R (which is polynomial in the length of the first argument) and a common input x. The prover also gets an additional input, w. If $R(x,w) = 1$ then protocol should allow the prover to cause the verifier to output 1. If there exists no w such that $R(x,w) = 1$ then the verifier should output 0. There is no requirement in the case where there exists w s.t. $R(x,w) = 1$ but the prover does not have such w as input.[14] In any case, the verifier should "learn nothing" from the protocol except of whether there exists a w such that $R(x,w) = 1$. See [G95] for many more details on and variants of Zero-Knowledge.

Cast in the present framework, a Zero-Knowledge protocol is a protocol that securely realizes the Zero-Knowledge functionality, $\mathcal{F}_{\text{ZK}}$, described in Figure 16. In fact, $\mathcal{F}_{\text{ZK}}$ provides a somewhat stronger guarantee than informally described above: it requires the prover, P_j, to explicitly provide the "witness" w. This means that $\mathcal{F}_{\text{ZK}}$ has a flavor of a "proof of knowledge", and moreover a non-malleable one, since the verifier knows that it was the specified prover who provided w to $\mathcal{F}_{\text{ZK}}$.

Functionality $\mathcal{F}_{\text{ZK}}$ is parameterized by the relation R. An alternative formalization allows the prover to specify R in its message to $\mathcal{F}_{\text{ZK}}$. However, if this alternative is taken then one has to take care of encoding and complexity issues that are avoided in the present formalization. Notice that $\mathcal{F}_{\text{ZK}}$ does not wait for any input value from the verifier. Instead, it receives the common input value from the prover, and forwards it to the verifier.[15] Furthermore, the adversary is notified whether the verifier accepts. This represents the fact that ZK is not traditionally meant to hide this information.

[14] In the setting of [GMRa89], where the prover is computationally unbounded, this is a moot point (since the prover can always find such w in case it exists).

[15] Indeed, previous formalizations of $\mathcal{F}_{\text{ZK}}$, including the one in [CF01], were more complex. The present formalization was suggested by Yehuda Lindell.

Functionality $\mathcal{F}_{\text{ZK}}^{R}$

$\mathcal{F}_{\text{ZK}}^{R}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$, given a binary relation R.

1. Upon receipt of a value $(\texttt{prover}, id, P_i, P_j, x, w)$ from some party P_i, Send $(id, P_i, x, R(x,w))$ to P_j and $\mathcal{S}$, and halt.

Figure 16: The Zero-Knowledge functionality, $\mathcal{F}_{\text{ZK}}$

As usual, the composition theorem guarantees that, if a protocol securely realizes $\mathcal{F}_{\text{ZK}}$ for some relation R, then security of this protocol is guaranteed even when many copies of a ZK protocol are running concurrently, in an interleaved way and with related inputs. This in particular guarantees that protocols that securely realize $\mathcal{F}_{\text{ZK}}$ are Concurrent Zero Knowledge protocols as in [F91, DNS98]. Moreover, they are strong proofs of knowledge as in [DDOPS01].

On the feasibility of realizing $\mathcal{F}_{\text{ZK}}$. As shown in [CF01], $\mathcal{F}_{\text{ZK}}$ can be securely realized in the $\mathcal{F}_{\text{CR}}$-hybrid model. In fact, it is shown there that in the $\mathcal{F}_{\text{COM}}$-hybrid model one can securely realize $\mathcal{F}_{\text{ZK}}$ for any NP relation, via a three-move protocol, *without any computational assumptions.* (That is, cryptographic assumptions are needed only to realize $\mathcal{F}_{\text{COM}}$.)

Claim 19 ([CF01]) *Let R be an* NP *relation. Then there exists a (three move) protocol for realizing $\mathcal{F}_{\text{ZK}}^{R}$ in the $F_{\text{AUTH}}, F_{\text{COM}}$-hybrid model.*

We complement this result by showing that $\mathcal{F}_{\text{ZK}}$ is impossible to realize in the plain model for non-trivial relations. The proof uses arguments similar to those of [CF01] in the proof of Claim 18.

Claim 20 *There do not exist two-party terminating protocols that realize $\mathcal{F}_{\text{ZK}}^{R}$ in the* AM*, unless the language $L_R = \{x | \exists w \text{ s.t. } R(x,w)\}$ is in* BPP.

Proof: Let R be binary a relation and let π be a protocol that securely realizes $\mathcal{F}_{\text{ZK}}^{R}$. We construct a BPP algorithm for deciding L_R, in three steps as follows.

First, consider the following environment, Z_P. On input x, w, environment $\mathcal{Z}_P$ instructs the adversary to corrupt some P_i. Then it runs the program of the honest prover on input $(\texttt{prover}, id, P_i, P_j, x, w)$ for some id and an uncorrupted P_j, and instructs the adversary to deliver the generated message to P_j and report all of P_j's responses. At the end of the protocol execution, $\mathcal{Z}_P$ outputs 1 iff P_j outputs (id, P_i, x, v) where $v = R(x,w)$. It is easy to see that, when $\mathcal{Z}_P$ interacts with parties running π and the dummy adversary $\tilde{\mathcal{A}}$ of Definition 4 in the real-life model then it outputs 1 almost always. Consequently, the ideal-process adversary $\mathcal{S}_P$ for $\mathcal{Z}_P$ and $\tilde{\mathcal{A}}$ must guarantee that $\mathcal{Z}_P$ almost always outputs 1 in the ideal process. This implies that $\mathcal{S}_P$, after interacting with $\mathcal{Z}_P$, generates a value $(\texttt{prover}, id, P_i, P_j, x, w')$ where $R(x,w) = R(x,w')$. (This is the value to be given to the ideal functionality $\mathcal{F}_{\text{ZK}}^{R}$.)

Next, consider the following environment, $\mathcal{Z}_V$. On input x, w, environment $\mathcal{Z}_V$ instructs the adversary to corrupt some P_j, and invokes some uncorrupted P_i on input $(\texttt{prover}, id, P_i, P_j, x, w)$. Then it runs the program of the honest verifier, and instructs the adversary to deliver the generated message to P_i in the name of P_j, and report all of P_i's responses. At the end of the protocol execution, $\mathcal{Z}_P$ outputs 1 iff the verifier program outputs (id, P_i, x, v) where $v = R(x,w)$. It is easy to see that, when $\mathcal{Z}_V$ interacts with parties running π and the dummy adversary $\tilde{\mathcal{A}}$ in the real-life model then it outputs 1 almost always. Consequently, the ideal-process adversary $\mathcal{S}_V$ for $\mathcal{Z}_V$ and $\tilde{\mathcal{A}}$ must guarantee that $\mathcal{Z}_V$ almost always outputs 1 in the ideal process. This implies that $\mathcal{S}_V$, after receiving (id, P_i, x, v) from $\mathcal{F}_{\text{ZK}}$, interacts with $\mathcal{Z}_V$ in a way that causes the honest verifier code to output (id, P_i, x, v).

The decision procedure for L_R now proceeds as follows. On input x, simulate an interaction between $\mathcal{S}_V$ and $\mathcal{S}_P$, where $\mathcal{S}_V$ plays the environment $\mathcal{Z}_P$ for $\mathcal{S}_P$, and $\mathcal{S}_P$ plays the environment $\mathcal{Z}_V$ for $\mathcal{S}_V$. When $\mathcal{S}_V$ expects to receive a message from $\mathcal{F}_{\text{ZK}}$, provide it with the message $(id, P_i, x, 1)$ (i.e., guess that $x \in L_R$). If at the end of the interaction $\mathcal{S}_P$ outputs a message $(\texttt{prover}, id, P_i, P_j, x, w')$ where $R(x, w')$ holds then accept x. Otherwise reject.

It follows from the correctness of $\mathcal{S}_V$ and $\mathcal{S}_P$ that if $x \in L_R$ then $\mathcal{S}_P$ will output a good witness w' with high probability. If $x \notin R_L$ then no such good witness exists. □

8.3.4 Oblivious Transfer

Oblivious Transfer (OT) is a task for two parties, a sender with input $x_1, ..., x_l$, and a receiver with input $i \in \{1, .., l\}$. The receiver should learn x_l (and nothing else) and the sender should learn nothing. OT was introduced in [R81, EGL85] and studied in many works since.

In the present framework, a secure OT protocol is a protocol that realizes an "ideal OT" functionality. The proposed functionality, $\mathcal{F}_{\text{OT}}$, is presented in Figure 17. (Using standard terminology, $\mathcal{F}_{\text{OT}}$ captures 1-out-of-l OT.)

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

$\mathcal{F}_{\text{OT}}$ proceeds as follows, running with security parameter k, parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Wait to receive a value $(\texttt{sender}, id, P_i, P_j, x_1, ..., x_l)$ from some party P_i, where each $x_m \in \{0,1\}^*$. Once such a value is received, ignore all subsequent $(\texttt{sender}\ldots)$ values.
2. Wait to receive a value $(\texttt{receiver}, id, P_i, P_j, m)$ from P_j, where $m \in \{1..l\}$. Then, send $(\texttt{output}, id, x_m)$ to P_j, send (id, P_i, P_j) to the adversary, and halt.

Figure 17: The Oblivious Transfer functionality, $\mathcal{F}_{\text{OT}}$

On the feasibility of realizing $\mathcal{F}_{\text{OT}}$. $\mathcal{F}_{\text{OT}}$ can be realized either in the server-assisted setting of Theorem 9. Furthermore, the general construction of [CL01] implies that $\mathcal{F}_{\text{OT}}$ can be realized in the $\mathcal{F}_{\text{ZK}}$-hybrid model, if claw-free functions exits. On the other hand, using similar techniques to those of Claim 20, it can be seen that:

Claim 21 *There do not exist two-party terminating protocols that realize $\mathcal{F}_{\text{OT}}$ in the* AM.

8.4 Multiparty functionalities

8.4.1 Verifiable Secret Sharing

Verifiable Secret Sharing (VSS) is a multi-party primitive that consists of two phases. In the sharing phase, a special party (the dealer, denoted D) shares its inputs among the parties. In the opening phase, the parties reconstruct the dealers input. It is required that at the end of the sharing phase the dealers input remains secret. Furthermore, at that time a unique value v should be fixed, such that the value reconstructed by the parties in the opening phase equals v. Finally, if the dealer is uncorrupted then the value reconstructed by the parties should equal the dealers input. We stress that, in contrast with commitment, here the shared value should be reconstructible even without the cooperation of the dealer.

VSS was introduced in [CGMA85]. Several definitions of VSS exist (e.g., [FM97, BGW88, GM95]), but no definition guarantees secure composition with other protocols (nor with other copies of the same protocol) in our setting. Also, since VSS is inherently a "two step process", it cannot be naturally captured as secure evaluation of some function. (Indeed, it is possible to construct valid VSS protocols that do not evaluate any function.)

The ideal VSS functionality, denoted $\mathcal{F}_{\text{VSS}}$, is parameterized by m, the minimum number of parties required in order to open a shared secret. (It is easy to see that $m + t \leq n$ should hold for the functionality to be meaningful, where t is the maximum number of faults.) The functionality is presented in Figure 18.

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

$\mathcal{F}_{\text{VSS}}$ proceeds as follows, running with parties $D, P_1, ..., P_n$ and an adversary $\mathcal{S}$. Let m be the number of parties required to open a shared secret.

1. Upon receiving a value (`input`,id,v) from D, send (`shared`,id) to $\mathcal{S}$. Ignore all future (`input`,id,`...`) values.
2. Upon receiving a value (`open`,id) from m parties, and if a value (`input`,id,v) has been received from D, then send (`open`,id, v) to all parties and $\mathcal{S}$.

Figure 18: The Verifiable Secret Sharing functionality, $\mathcal{F}_{\text{VSS}}$.

Remark: Functionality $\mathcal{F}_{\text{VSS}}$ can be generalized to deal with cases where only a subset of the parties participates in a protocol invocation. In addition, it may be specified that in the opening phase the shared secret is to be given only to a subset of the participants. Also, the identity of the dealer may be determined dynamically via an appropriate message from one or more parties.

On the feasibility of realizing $\mathcal{F}_{\text{VSS}}$. Say that a protocol t-securely realizes $\mathcal{F}$ if it securely realizes $\mathcal{F}$ whenever at most t parties are corrupted. Theorem 9 implies that $\mathcal{F}_{\text{VSS}}$ can be t-securely realized in a synchronous setting whenever $t < n/3$. In fact, the VSS protocols of [BGW88, FM97] by themselves t-securely realize $\mathcal{F}_{\text{VSS}}$ if the communication is ideally secure. Furthermore, we conjecture that the AVSS (Asynchronous VSS) protocols of [BCG93] and [BKR94, CR93] t-securely realize $\mathcal{F}_{\text{VSS}}$ in a non-blocking asynchronous setting with ideally secure communication, respectively for $t < n/4$ and $t < n/3$.

8.4.2 Secure Function Evaluation

Secure Function Evaluation (SFE) is a multi-party primitive for synchronous networks, where each party P_i out of $P_1, ..., P_n$ has an input value x_i, and obtains an output value $f(x_1, ..., x_n, r)_i$, where

$f : (\{0,1\}^*)^n \times R \to (\{0,1\}^*)^n$ is a given function and $r \overset{R}{\leftarrow} R$. Several definitions for SFE exist, e.g. [GL90, MR91, B91, C00].

A first attempt to cast the ideal process of secure function evaluation as an ideal functionality may proceed as follows. First the ideal functionality waits to receive input values from all parties. Once all inputs are received, the functionality evaluates the function on these inputs, and sends each party a message containing its output value.

Indeed, this functionality captures the limitations on the information gathered by the adversary and the correctness requirements from the outputs of the uncorrupted parties, in the case where the uncorrupted parties generate outputs. However, it fails to guarantee that the uncorrupted parties generate any output at all — even in a synchronous system where message delivery is guaranteed. (In fact, the protocol where no party generates any output securely realizes this functionality for any function f to be evaluated: All the ideal-process adversary has to do is to have the corrupted parties not send anything to the ideal functionality.)

The following ideal functionality, denoted $\mathcal{F}_{\text{SFE}}$, provides the additional guarantee that all uncorrupted parties terminate the protocol with output values. It is parameterized by t, the maximum number of corrupted parties. Functionality $\mathcal{F}_{\text{SFE}}$ (presented in Figure 19) expects to receive two different messages from each party P_i: The first message includes the input value contributed by P_i. The second message is a `ready` value, whose interpretation is that P_i is ready to evaluate the function. As soon as $\mathcal{F}_{\text{SFE}}$ receives $t+1$ `ready` messages, it evaluates the function based on the input values it received so far, and sends the corresponding part of the function value to each party. (The "application protocol" that invokes a SFE protocol π is expected to instruct the parties to activate π with the `ready` input at the round following the round where the inputs are given.)

Functionality $\mathcal{F}_{\text{SFE}}$ guarantees that in the synchronous ideal process the parties always output their function values, regardless of what $\mathcal{S}$ does. In particular, at a given round $\mathcal{Z}$ can activate each party P_i with an input value x_i. At subsequent rounds, activate each party with input value `ready`. It is now guaranteed that all uncorrupted parties will generate output.

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

$\mathcal{F}_{\text{SFE}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$. Let t be the maximum number of corrupted parties, and let $f : (\{0,1\}^*)^n \times R \to (\{0,1\}^*)^n$.

Upon activation, set variables $x_1, ..., x_n \leftarrow \perp$ and $\ell \leftarrow 0$. Next:

1. Upon receiving a value (`Input`,id, v) from P_i, set $x_i = v$.
2. Upon receiving a value (`ready`,id) for the first time from some party, increment ℓ. Once $\ell = t+1$, choose $r \overset{R}{\leftarrow} R$, let $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n, r)$, and send a message (`Output`,id, y_i) to each party P_i.

Figure 19: The Secure Function Evaluation functionality for evaluating an n-party function f

Remarks: **1.** Functionality $\mathcal{F}_{\text{SFE}}$ assumes by default that all parties participate in the evaluation of the function. A natural extension is to let F_{SFE} interact only with a subset of the parties. The identities of the parties in the subset can be given to the functionality either as a fixed parameter, or as input.

2. It follows from the general result of Section 7 that functionality $\mathcal{F}_{\text{SFE}}$ can be t-securely realized in a synchronous network, whenever $t < n/3$.

3. We conjecture that functionality $\mathcal{F}_{\text{SFE}}$ can be t-securely realized also in *non-blocking asynchronous networks,* using the techniques of [BCG93, BKR94]. Here, however, the interpretation of this functionality is slightly weaker: the "application" that calls the SFE protocol cannot guarantee that the `ready` values are given only after *all* uncorrupted parties have contributed their inputs. Thus it cannot be guaranteed that the function is evaluated based on the input values of all uncorrupted parties. Instead, it can only be guaranteed that the input values of $n - t$ parties are taken into account in the evaluation of f.

9 Future directions

The present work puts forth a general framework for defining and analyzing security or protocols. This may be regarded as a step towards putting the art of cryptographic protocol design on firm grounds. Let us mention two possible directions for future work towards this goal.

Formulating and realizing functionalities. The most immediate direction for future research is to come up with protocols that securely realize the ideal functionalities presented in Section 8 (or alternatively to show that these functionalities are realizable by existing protocols). In particular, this includes the Commitment, Oblivious Transfer, Zero-Knowledge (with and without the 'designated prover' property), Secret Communication, and Signatures (against adaptive adversaries).

Furthermore, ideal functionalities for capturing the security requirements of other tasks can be written and subsequently realized. Here examples include agreement tasks (such as broadcast and Byzantine agreement); variants of authentication (such as the "one-sided authentication" mentioned in [Sh99]); identification; threshold cryptography tasks such as threshold signatures and threshold decryption; group signatures; electronic-commerce tasks such as auctions, contract signing (either in the *optimistic model* of [ASW97] or in the plain model), anonymous transactions, and more.

A more general approach may be to try to find characterizations of the functionalities that can be realized in certain settings. Two interesting questions are which functionalities can be realized by two-party protocols in the bare model, or the AM, and which functionalities can be realized in the "common random string model" (i.e., in the $\mathcal{F}_{\text{CR}}$-hybrid model).

A related goal is to try to relax the requirements from protocols that realize a certain task, while maintaining the universal composability property. Indeed, the approach of defining security as the ability to "emulate" an ideal process has traditionally resulted in definitions that are more stringent than definitions based on other methods for defining security. Examples include Zero-Knowledge versus Witness-Indistinguishability of Interactive Proof-systems [GMRa89, FS90], Key Exchange protocols [BR93, BCK98] and more. The present framework is even more restrictive in that it requires the ideal-process adversary to mimic the real-life adversary *for any environment machine.* Thus demonstrating security of protocols within the present framework may not be trivial. Let us point to two ways in which the task of constructing and proving security of protocols may be simplified. A first direction may be to modify the framework itself (i.e., to relax Definition 3) in a way that maintains universal composability. A second direction is to try to formulate ideal functionalities in a way that relaxes the requirements as much as possible and simplifies proving security of protocols. (Examples of how this can be done include the treatment of digital signatures in Section 8.2.1 and the treatment of Key-Exchange in [CK01a].)

Exploring connections with formal-methods calculi and automated analysis. Traditionally, proofs of security of cryptographic protocols (say, based on known hardness assumptions) are hand-written and require considerable proficiency and thought. In fact, security analysis of even simple systems in the present framework is a considerable undertaking. It thus seems that the only way to apply the framework to more complex tasks and systems is to *automate* the proof process.

Tools for automating the process of asserting properties of programs and communication protocols exist, say using model-checking techniques (see, e.g., [CGP99]). As mentioned in the Introduction, some of these tools have been applied to verifying security properties of protocols, e.g. [M94, L96, So99]. However, these tools represent cryptographic primitives as symbolic operations and thus lack in soundness: There is no guarantee that a protocol that passes the analysis is indeed 'secure'.

Potentially, definitions of security in the present framework may serve as a computationally-sound basis for automated analysis of protocols based on formal methods. More specifically, ideal functionalities, as defined and used here, may be regarded as idealizations of cryptographic tasks that "hide the dirty details of cryptography" from applications, but at the same time are realizable by actual protocols. Consequently, it may be possible to develop calculi of concurrent protocols where cryptography is represented via ideal functionalities that are in fact securely realizable. Such calculi may lead to automated proofs of security of protocols that maintain cryptographic soundness.

We note that first steps in essentially the same direction have already been taken. In particular, Abadi and Rogaway [AR00] demonstrate the cryptographic soundness of some formal derivation rules for "symbolic encryption". This promising line of research can potentially be extended (say, as described above) to more general and complex cryptographic tasks.

Acknowledgments

Much of the motivation for undertaking this project, and many of the ideas that appear here, come from studying secure key-exchange protocols together with Hugo Krawczyk. I thank him for this long, fruitful, and enjoyable collaboration. I am also grateful to Oded Goldreich who, as usual, gave me both essential moral support and invaluable technical and presentational advice.

Many thanks also to the many people with whom I have interacted over the years on definitions of security and secure composition. A very partial list includes Martin Abadi, Mihir Bellare, Ivan Damgard, Mark Fischlin, Shafi Goldwasser, Rosario Gennaro, Shai Halevi, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, Phil MacKenzie, Tal Malkin, Cathy Meadows, Silvio Micali, Daniele Micciancio, Moni Naor, Rafi Ostrovsky, Tal Rabin, Charlie Rackoff, Phil Rogaway, Victor Shoup, Paul Syverson and Michael Waidner. In particular, Daniele made several observations that have considerably simplified and improved the presentation.

References

[AFG98] M. Abadi, C. Fournet, and G. Gonthier, "Secure implementation of channel abstractions", In Proceedings LICS'98, pages 105–116, 1998.

[AG97] M. Abadi and A. D. Gordon, A calculus for cryptographic protocols: The spi calculus. In *proceedings of the 4th ACM Conference on Computer and Communications Security,* 1997, pp.36-47. Fuller version available at http://www.research.digital.com/SRC/ abadi.

[AR00] M. Abadi and P. Rogaway, Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption), *International Conference on Theoretical Computer Science IFIP TCS 2000,* LNCS, 2000. On-line version at http://pa.bell-labs.com/ abadi/.

[AB01] J. An and M. Bellare, "Does encryption with redundancy provide authenticity?" In Advances in Cryptology - Eurocrypt 2001 Proceedings, Lecture Notes in Computer Science Vol. 2045 , B. Pfitzmann ed, Springer-Verlag, 2001.

[ASW97] N. Ashokan, M. Schunter and M. Waidner, Optimistic protocols for fair exchange, *4th ACM Conference on Computer and Communication Security,* 1997.

[B91] D. Beaver, "Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority", J. Cryptology, Springer-Verlag, (1991) 4: 75-122.

[B96] D. Beaver, "Adaptive Zero-Knowledge and Computational Equivocation", *28th Symposium on Theory of Computing (STOC),* ACM, 1996.

[B97] D. Beaver, "Plug and play encryption", *CRYPTO 97,* 1997.

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology — Eurocrypt '92*, LNCS No. 658, Springer-Verlag, 1992, pages 307–323.

[BCK98] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key-exchange protocols", *30th Symposium on Theory of Computing (STOC),* ACM, 1998.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, "Relations among notions of security for public-key encryption schemes", *CRYPTO '98,* 1998, pp. 26-40.

[BR93] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology, - CRYPTO'93*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed, Springer-Verlag, 1994, pp. 232-249.

[BR93A] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[BCG93] M. Ben-Or, R. Canetti and O. Goldreich, "Asynchronous Secure Computations", *25th Symposium on Theory of Computing (STOC),* ACM, 1993, pp. 52-61.

[BGW88] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th Symposium on Theory of Computing (STOC),* ACM, 1988, pp. 1-10.

[BKR94] M. Ben-Or, B. Kelmer and T. Rabin, Asynchronous Secure Computations with Optimal Resilience, *13th PODC,* 1994, pp. 183-192.

[B82] M. Blum, "Coin flipping by telephone", IEEE Spring COMPCOM, pp. 133-137, Feb. 1982.

[BFM89] M. Blum, P. Feldman and S. Micali, "Non-interactive Zero-Knowledge and its applications," *20th STOC,* 1988.

[BCC88] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, Vol. 37, No. 2, pages 156–189, 1988.

[BAN90] M. Burrows, M. Abadi and R. Needham, "A logic for authentication," DEC Systems Research Center Technical Report 39, February 1990. Earlier versions in *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, and *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles,* 1989.

[C95] R. Canetti, "Studies in Secure Multi-party Computation and Applications",*Ph.D. Thesis,* Weizmann Institute, Israel, 1995.

[C00] R. Canetti, "Security and composition of multi-party cryptographic protocols", *Journal of Cryptology,* Vol. 13, No. 1, winter 2000.

[C01] R. Canetti, "A unified framework for analyzing cryptographic protocols", ECCC TR 01-16. Also available at http://eprint.iacr.org/2000/067.

[CF01] R. Canetti and M. Fischlin, "Universally Composable Commitments", Crypto '01, 2001.

[CFGN96] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th Symposium on Theory of Computing (STOC),* ACM, 1996. Fuller version in MIT-LCS-TR #682, 1996.

[CGH98] R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, Dallas, TX, May 1998. ACM.

[CG99] R. Canetti and S. Goldwasser, A practical threshold cryptosystem resilient against adaptive chosen ciphertext attacks, *Eurocrypt '99,* 1999.

[CHH00] R. Canetti, S. Halevi and A. Herzberg, "How to Maintain Authenticated Communication", *Journal of Cryptology,* Vol. 13, No. 1, winter 2000. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 15-25.

[CK01] R. Canetti and H. Krawczyk, "Analysis of key exchange protocols and their use for building secure channels", Eurocrypt '01, 2001.

[CK01a] R. Canetti and H. Krawczyk, in preparation.

[CKOR00] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen, "Randomness vs. Fault-Tolerance", *Journal of Cryptology,* Vol. 13, No. 1, winter 2000. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997,

[CL01] R. Canetti and Y. Lindell, "Universally Composable Two-Party Computation", in preparation.

[CR93] R. Canetti and T. Rabin, Optimal Asynchronous Byzantine Agreement, *25th STOC,* 1993, pp. 42-51.

[CGMA85] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults", *26th FOCS,* 1985, pp. 383-395.

[CGP99] E. Clarke, O. Grunberg and E. Peled, *Model Checking,* MIT Press, 1999.

[CDDHR99] R. Cramer, I. Damgaard, S. Dziembowski, M. Hirt and T. Rabin ,"Efficient multiparty computations secure against an adaptive adversary", *Eurocrypt*, 1999, pp. 311-326.

[DN00] I. Damgaard and J. B. Nielsen, improved non-committing encryption schemes based on general complexity assumption, CRYPTO 2000, pp. 432-450.

[DOW92] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[DH76] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory* IT-22, November 1976, pp. 644–654.

[DDOPS01] A. De Santis, G. Di Crecenzo, R. Ostrovsky, G. Persiano and A. Sahai, "Robust Non-Interactive Zero-Knowledge," *CRYPTO 01,* 2001.

[DIO98] G. Di Crescenzo, Y. Ishai and R. Ostrovsky, Non-interactive and non-malleable commitment, *30th STOC,* 1998, pp. 141-150.

[DM00] Y. Dodis and S. Micali, "Secure Computation", *CRYPTO '00,* 2000.

[DDN00] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *SIAM. J. Computing,* Vol. 30, No. 2, 2000, pp. 391-437. Preliminary version in *23rd Symposium on Theory of Computing (STOC)*, ACM, 1991.

[DY83] D. Dolev and A. Yao, On the security of public-key protocols, *IEEE Transactions on Information Theory,* 2(29), 1983.

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 523–534. IEEE, 1999.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.

[EGL85] S. Even, O. Goldreich and A. Lempel, "A randomized protocol for signing contracts", *CACM,* vol. 28, No. 6, 1985, pp. 637-647.

[F91] U. Feige. Ph.D. thesis, Weizmann Institute of Science, 1991.

[FS90] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

[FM97] P. Feldman and S. Micali, An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement, *SIAM Journal on Computing, Vol. 26, No. 4,* 1997, pp. 873–933.

[FF00] M. Fischlin and R. Fischlin, "Efficient non-malleable commitment schemes", *CRYPTO '00, LNCS 1880*, 2000, pp. 413-428.

[GM00] J. Garay and P. MacKenzie, "Concurrent Oblivious Transfer", *41st FOCS,* 2000.

[GM95] R. Gennaro and S. Micali, Verifiable Secret Sharing as Secure Computation, *Eurocrypt 95, LNCS 921,* 1995, pp. 168-182.

[GRR98] R. Gennaro, M. Rabin and T Rabin, Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography, *17th PODC,* 1998, pp. 101-112.

[G95] O. Goldreich, *"Foundations of Cryptography (Fragments of a book)"*, Weizmann Inst. of Science, 1995. (Available at http://philby.ucsd.edu)

[G98] O. Goldreich. *"Secure Multi-Party Computation"*, 1998. (Available at http://philby.ucsd.edu)

[GK88] O. Goldreich and H. Krawczyk, On the Composition of Zero-Knowledge Proof Systems, *SIAM. J. Computing,* Vol. 25, No. 1, 1996.

[GL00] O. Goldreich and Y. Lindell, Session-key generation using human passwords only, manuscript, 2000.

[GMW87] O. Goldreich, S. Micali and A. Wigderson, "How to Play any Mental Game", *19th Symposium on Theory of Computing (STOC),* ACM, 1987, pp. 218-229.

[GO94] O. Goldreich and Y. Oren, "Definitions and properties of Zero-Knowledge proof systems", *Journal of Cryptology*, Vol. 7, No. 1, Springer-Verlag, 1994, pp. 1–32. Preliminary version by Y. Oren in *28th Symp. on Foundations of Computer Science (FOCS),* IEEE, 1987.

[GL90] S. Goldwasser, and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO '90, LNCS 537,* Springer-Verlag, 1990.

[GM84] S. Goldwasser and S. Micali, "Probabilistic encryption", *JCSS,* Vol. 28, No 2, April 1984, pp. 270-299.

[GMRa89] S. Goldwasser, S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Journal on Comput.,* Vol. 18, No. 1, 1989, pp. 186-208.

[GMRi88] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, April 1988, pages 281–308.

[HM00] M. Hirt and U. Maurer, "Complete characterization of adversaries tolerable in secure multi-party computation", *Journal of Cryptology,* Vol 13, No. 1, 2000, pp. 31-60. Preliminary version in *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 25–34.

[KMM94] R. Kemmerer, C. Meadows and J. Millen, Three systems for cryptographic protocol analysis, *J. Cryptology,* 7(2):79-130, 1994.

[K01] H. Krawczyk, "The order of encryption and authentication for protecting communications (Or: how secure is SSL?)," *CRYPTO 01,* 2001.

[LLR01] Y. Lindell, A. Lysyanskaya and T. Rabin, "On the composition of authenticated Byzantine agreement", manuscript, 2001.

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, A. Schedrov, "A Probabilistic Poly-time Framework for Protocol Analysis", *5th ACM Conf. on Computer and Communication Security*, 1998, pp. 112-121.

[LMMS99] P. Lincoln, J. Mitchell, M. Mitchell, A. Schedrov, "Probabilistic Polynomial-time equivalence and security analysis", *Formal Methods Workshop,* 1999. Available at ftp://theory.stanford.edu/pub/jcm/papers/fm-99.ps.

[L96] G. Lowe, Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR, *2nd International Workshop on Tools and Algorithms for the construction and analysis of systems,* Springer-Verlag, 1996.

[L96] N. Lynch, *Distributed Algorithms,* Morgan Kaufman, San Francisco, 1996.

[M94] C. Meadows, A model of computation for the NRL protocol analyzer, *Computer Security Foundations Workshop,* IEEE Computer Security Press, 1994.

[M94a] C. Meadows, Formal verification of cryptographic protocols: A survey, *Asiacrypt '94,* LNCS 917, 1995, pp. 133-150.

[MRV99] S. Micali, M. Rabin, and S. Vadhan, "Verifiable Random Functions", *40th Annual Symposium on Foundations of Computer Science,* 1999.

[MR91] S. Micali and P. Rogaway, "Secure Computation", unpublished manuscript, 1992. Preliminary version in *CRYPTO '91, LNCS 576,* Springer-Verlag, 1991.

[MPW92] R. Milner, J. Parrow and D. Walker, A calculus of mobile processes, parts I and II. *Information and computation,* 1992. pp. 1-40 and 41-77.

[MMS98] J. Mitchell, M. Mitchell, A. Schedrov, "A Linguistic Characterization of Bounded Oracle Computation and Probabilistic Polynomial Time," *39th FOCS*, 1998, pp. 725-734.

[NY90] M. Naor and M. Yung, "Public key cryptosystems provably secure against chosen ciphertext attacks", *22nd STOC,* 427-437, 1990.

[PW94] B. Pfitzmann and M. Waidner, "A general framework for formal notions of secure systems", Hildesheimer Informatik-Berichte 11/94, Universitat Hildesheim, 1994. Available at http://www.semper.org/sirene/lit.

[PSW00] B. Pfitzmann, M. Schunter and M. Waidner, "Secure Reactive Systems", IBM Research Report RZ 3206 (#93252), IBM Research, Zurich, May 2000.

[PSW00a] B. Pfitzmann, M. Schunter and M. Waidner, "Provably Secure Certified Mail", IBM Research Report RZ 3207 (#93253), IBM Research, Zurich, August 2000.

[PW00] B. Pfitzmann and M. Waidner, "Composition and integrity preservation of secure reactive systems", *7th ACM Conf. on Computer and Communication Security*, 2000, pp. 245-254.

[PW01] B. Pfitzmann and M. Waidner, "A model for asynchronous reactive systems and its application to secure message transmission", IEEE Symposium on Security and Privacy, May 2001. Preliminary version in http://eprint.iacr.org/2000/066 and IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

[R81] M. Rabin, "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[RB89] T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multi-party Protocols with Honest Majority", *21st Symposium on Theory of Computing (STOC),* ACM, 1989, pp. 73-85.

[RS91] C. Rackoff and D. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack", *CRYPTO '91,* 1991.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt99*, Springer LNCS 1592, pages 415–413.

[Sh99] V. Shoup, "On Formal Models for Secure Key Exchange", manuscript, 1999. Available at: http://www.shoup.org.

[So99] D. Song, Athena: an Automatic Checker for Security Protocol Analysis, *Proc. of 12th IEEE Computer Security Foundation Workshop*, June 1999.

[Y82] A. Yao, Theory and applications of trapdoor functions, In *Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE, 1982.