General Cryptographic Protocols: The Very Basics

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
oded.goldreich@weizmann.ac.il

March 13, 2010

Abstract

We survey basic definitions and results concerning secure multi-party computations, where the two-party case is an important special case. In a nutshell, these results assert that, under a variety of reasonable settings and/or assumptions, it is possible to construct protocols for securely computing *any* desirable multi-party functionality. Confining ourselves to the very basics of this vast area of study, we focus on the stand-alone setting, while leaving the survey of the study of the security of concurrent executions to other surveys.

Contents

1 Introduction

The modern society is quite preoccupied with various statistics like the average, median, and deviation of various attributes (e.g., salary) of its members.[1] On the other hand, individuals often wish to keep their own attributes secret (although they are interested in the aforementioned statistics). Furthermore, on top of being suspicious of other people, individuals are growing to be suspicious of all (the society's) establishments and are unwilling to trust the latter with their secrets. Under these circumstances it is not clear whether there is a way for the members of the society to obtain various statistics (regarding all secrets) without revealing their individual secrets to other people.

The foregoing question is a special case of a general problem. We are talking about computing some (predetermined) function of inputs that are scattered among different parties, without having these parties reveal their individual inputs. The mutually suspicious parties have to employ some distributed protocol in order to compute the function value, without leaking any other information regarding their inputs to one another. Furthermore, in some settings, some of the parties may deviate from the protocol, and it is desired that such malfunctioning will not be of any advantage to them. At best, we would like to "emulate" a trusted party (which collects the inputs from the parties, computes the corresponding outputs, and hand them to the corresponding parties), and do so in a distributed setting in which no trusted parties exist. This, in a nutshell, is what secure cryptographic protocols are all about.

The results surveyed in this article describe a variety of reasonable models in which such an "emulation" is possible. The models vary by the underlying assumptions regarding the communication channels, numerous parameters relating to the extent of adversarial behavior, and the desired level of emulation of the trusted party (i.e., level of "security"). Our focus is on general results regarding secure multi-party (and two-party) computations, where *general* means that we consider arbitrary desired functionalities (rather than specific ones). In a nutshell, these general results assert that *it is possible to construct protocols for securely computing any desired multi-party functionality*. Indeed, what is striking about these results is their generality, and we believe that the wonder is not diminished by the (various alternative) conditions under which these results hold.

1.1 The problem in a nutshell

A general framework for casting (m-party) cryptographic (protocol) problems consists of specifying a random process that maps m inputs to m outputs.[2] The inputs to the process are to be thought of as the local inputs of m parties, and the m outputs are their corresponding (desired) local outputs. The random process describes the desired functionality. That is, if the m parties were to trust each other (or trust some external party), then they could each send their local input to the trusted party, who would compute the outcome of the process and send to each party the corresponding output. A pivotal question in the area of cryptographic protocols is to what extent can this (imaginary) trusted party be "emulated" by the mutually distrustful parties themselves.

[1] We comment that it seems that more socially useful statistics concern the correlation between various attributes. Needless to say, these two are covered by the current discussion.

[2] That is, we consider the secure evaluation of randomized functionalities, rather than "only" the secure evaluation of functions. Specifically, we consider an arbitrary (randomized) process F that on input $(x_1, ..., x_m)$, first selects at random (depending only on $\ell \stackrel{\text{def}}{=} \sum_{i=1}^{m} |x_i|$) an m-ary function f, and then outputs the m-tuple $f(x_1, ..., x_m) = (f_1(x_1, ..., x_m), ..., f_m(x_1, ..., x_m))$. In other words, $F(x_1, ..., x_m) = F'(r, x_1, ..., x_m)$, where r is uniformly selected in $\{0,1\}^{\ell'}$ (with $\ell' = \text{poly}(\ell)$), and F' is a function mapping $(m+1)$-sequences to m-sequences.

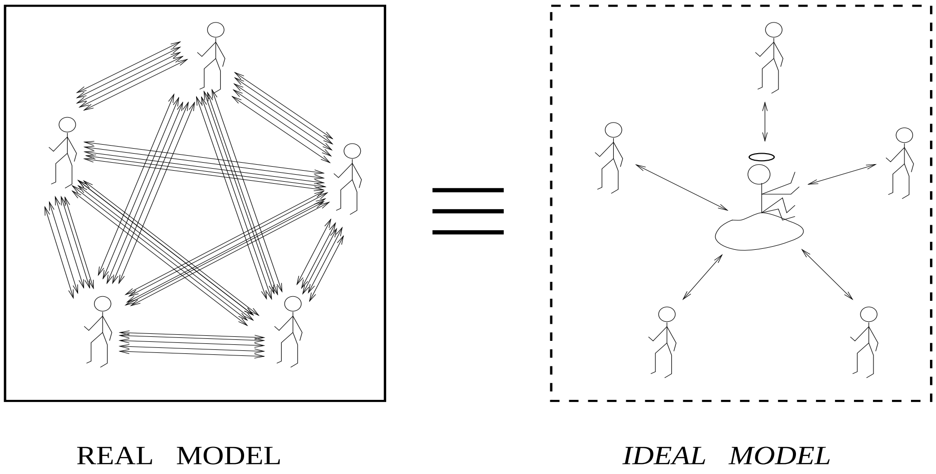

Figure 1: Secure protocols emulate a trusted party – an illustration.

The results surveyed below describe a variety of models in which such an "emulation" is possible. This means that in each of these models the services of an (imaginary) trusted party can be "emulated" by the mutually distrustful parties themselves. In particular, any desired functionality, which is trivially computed with the help of a trusted party, can be securely computed by these mutually distrustful parties.

1.2 Organization and prerequisites

Section 2 provides a rather comprehensive survey of the various definitions used in the area of secure multi-party computation, Section 3 surveys the main known results, and Section 2.2 describes the main ideas that underly these results.

Some readers may prefer to consider one concrete case of the definitional approach before encountering the general approach. Such readers are encouraged to start with Section 2.2, and possibly proceed to Section 4 before returning to Section 2.1. We mention that on top of presenting the basic ideas that underly the general constructions, Section 4 also provides sketches of a couple of concrete protocols.

All the above refers to the security of stand-alone executions. The preservation of security in an environment in which many executions of many protocols are being attacked is briefly considered in Section 5.

Prerequisites and suggestions for further reading: We assume basic familiarity with the foundations of cryptography. Specifically, the more technical parts of the exposition assume basic familiarity with the notions of trapdoor permutations, computational indistinguishability, zero-knowledge, encryption schemes, and commitment schemes. For an introduction to these foundations, at the level of the current article, we recommend our own primer [37].[3] A much more comprehensive treatment can be found in the two-volume work [35, 36]. (We also mention that a tutorial of zero-knowledge, which suffices for our purposes, appears in this volume [38].)

[3] In fact, the current article is a revision of [37, Sec. 7].

1.3 Three advanced comments

Before actually embarking, we address three advanced issues.

1.3.1 Relation to the rest of modern cryptography

The design of secure protocols that implement arbitrary desired functionalities is a major part of modern cryptography. Taking the opposite perspective, the design of any cryptographic scheme may be viewed as the design of a secure protocol for implementing a suitable functionality. Still, we believe that it makes sense to differentiate between basic cryptographic primitives (which involve little interaction) like encryption and signature schemes, on the one hand, and general cryptographic protocols on the other hand.

1.3.2 Relevance to practice

Our focus on the *general* study of secure multi-party computation (rather than on protocols for solving specific problems) is natural in the context of the theoretical treatment of the subject matter. We wish to highlight the importance of this *general* study to practice. Firstly, this study clarifies fundamental issues regarding security in a multi-party environment. Secondly, it draws the lines between what is possible in principle and what is not. Thirdly, it develops general techniques for designing secure protocols. And last, sometimes, it may even yield schemes (or modules) that may be incorporated in practical systems.

1.3.3 The issue of concurrent executions

The bulk of this article is devoted to the "stand-alone" setting. That is, except in Section 5, we presuppose that *during the execution of the (secure) protocol the parties that participate in the execution do not participate in any other protocol execution.* Thus, it is not guaranteed that the aforementioned protocol maintains its security when executed concurrently with other protocols (or even with other instances of the same protocol): Conceivably, an adversary that *controls* parties in several concurrent executions, may gain some illegitimate advantage. Thus, it is desirable (and in some settings imperative) to design protocols that maintain their security also when executed concurrently to other protocols (or to other instances of themselves). In Section 5, we briefly and partially survey the known results regarding security under concurrent executions. At this point, however, we wish to make several comments:

When do concurrent executions pose a security problem? The issue of security under concurrent execution arises *only* if the adversary may initiate and *control* several concurrent executions. In contrast, concurrent executions that are not controlled by the same adversary (or set of coordinating adversaries) do not introduce any new security problem (beyond stand-alone security).

An asymmetry between legitimate behavior and adversarial one. Preservation of security under concurrent executions seems essential in settings, such as the Internet, in which many (distributed) processes do take place concurrently and it is unreasonable to require these processes to coordinate their actions (in order to counter possible attacks of an adversary that may control several processes). We stress that although inter-process coordination cannot be required of the legitimate processes, it cannot be assumed that the adversary does not coordinate its attacks on the various processes. (Coordination is possible, but too expensive to be required in normal operation. Still the adversary may be willing to invest the necessary effort if, by coordinating its attack on the various processes, it can obtain substantial gain.)

When may stand-alone security suffice? It is hasty to conclude that "stand-alone security" is worthless in all distributed systems (i.e., is unsatisfactory in all reasonable settings). We believe that stand-alone security may be sufficient in some (typically, small) distributed systems.

- On the one extreme, stand-alone security suffices in distributed systems in which executions of secure multi-party computations are rare and can be coordinated such that they do not take place concurrently.

- On the other extreme, in distributed systems in which executions of secure multi-party computations involving all (or most) the processors take place all the time, it may be reasonable to "lump together" all these computations into a single (reactive) multi-party computation that supports on-line requests for various individual multi-party computations.

  As another (related) example, consider a (small) distributed system that operates under a single distributed operating system. The desired functionality of such an operating system can be casted as a (reactive) multi-party functionality, and as such one can design a secure implementation of it. This means that we obtain a secure distributed operating system that maintains its functionality even if some of the processors behave in a malicious way (e.g., are governed by an adversary).[4]

2 The Definitional Approach and Some Models

Before describing the aforementioned results, we further discuss the notion of "emulating a trusted party", which underlies the definitional approach to secure multi-party computation (as initiated and developed in [45, 56, 4, 5, 14, 15]) The approach can be traced back to the definition of zero-knowledge (cf. [47]), and even to the definition of secure encryption (cf. [33], rephrasing [46]). The underlying paradigm (called the simulation paradigm) is that a scheme is secure if whatever a feasible adversary can obtain after attacking it, is also feasibly attainable "from scratch". In the case of zero-knowledge this amounts to saying that whatever a (feasible) verifier can obtain after interacting with the prover on a prescribed valid assertion, can be (feasibly) computed from the assertion itself. In the case of multi-party computation we compare the effect of adversaries that participate in the execution of the actual protocol to the effect of adversaries that participate in an imaginary execution of a trivial (ideal) protocol for computing the desired functionality with the help of a trusted party. If whatever the adversaries can feasibly obtain in the former real setting can also be feasibly obtained in the latter ideal setting then the protocol "emulates the ideal setting" (i.e., "emulates a trusted party"), and so is deemed secure. This basic approach can be applied in a variety of models, and is used to define the goals of security in these models.[5]

[4] We comment that in a secure distributed operating system as suggested above, all (or most) parties will have to actively participate in each action taken by the system. Actually, if one assumes that at most t parties may be controlled by the adversary then it suffices to have $O(t)$ parties participate in each action taken by the system.

[5] A few technical comments are in place. Firstly, we assume that the inputs of all parties are of the same length. We comment that as long as the lengths of the inputs are polynomially related, the above convention can be enforced by padding. On the other hand, some length restriction is essential for the security results, because in general it is impossible to hide all information regarding the length of the inputs to a protocol. Secondly, we assume that the desired functionality is computable in probabilistic polynomial-time, because we wish the secure protocol to run in

We first discuss some of the parameters used in defining various models, and next demonstrate the application of this approach in two important models. For further details, see [15] or [36, Sec. 7.2 and 7.5.1].

2.1 Some parameters used in defining security models

The following parameters are described in terms of the actual (or real) computation. In *some cases*, the corresponding definition of security is obtained by imposing some restrictions or provisions on the ideal model. For example, in the case of two-party computation (see below), secure computation is possible only if premature termination is *not* considered a breach of security. In that case, the suitable security definition is obtained (via the simulation paradigm) by allowing (an analogue of) premature termination in the ideal model. In *all cases*, the desired notion of security is defined by requiring that for any adequate adversary in the real model, there exist a corresponding adversary in the corresponding ideal model that obtains essentially the same impact (as the real-model adversary).

2.1.1 The communication channels

The parameters of the model include questions like whether or not the channels may be tapped by an adversary, whether or not they are tamper-free, and questions referring to the network behavior (in the case of multi-party protocols).

Wire-tapping versus the private-channel model. The standard assumption in cryptography is that the adversary may tap all communication channels (between honest parties). In contrast, one may *postulate* that the adversary cannot obtain messages sent between a pair of honest parties, yielding the so-called *private-channel model* (cf. [11, 20]). The latter postulate may be justified in some settings. Furthermore, it may be viewed as a useful abstraction that provides a clean model for the study and development of secure protocols. In this respect, it is important to mention that, in a variety of settings of the other parameters, private channels can be easily emulated by ordinary "tapped channels".

Broadcast channel. In the multi-party context, one may postulate the existence of a *broadcast channel* (cf. [61]), and the motivation and justifications are as in the case of the private-channel model.

The tamper-free assumption. The standard assumption in the area is that the adversary cannot modify, duplicate, or generate messages sent over the communication channels (between honest parties). Again, this assumption can be justified in some settings and can be emulated in others (cf. [8, 16]).

Network behavior. Most works in the area assume that communication is *synchronous* and that point-to-point channels exist between every pair of processors (i.e., a *complete network*). However, one may also consider *asynchronous communication* (cf. [10]) and *arbitrary networks* of point-to-point channels (cf. [27]).

---

probabilistic polynomial-time (and a protocol cannot be more efficient than the corresponding centralized algorithm). Clearly, the results can be extended to functionalities that are computable within any given (time-constructible) time bound, using adequate padding.

2.1.2 Set-up assumptions

Unless stated differently, we make no set-up assumptions (except for the obvious assumption that all parties have identical copies of the protocol's program). However, in some cases it is assumed that each party knows a verification-key corresponding to each of the other parties (or that a public-key infrastructure is available). Another assumption, made more rarely, is that all parties have access to some common (trusted) random string.

2.1.3 Computational limitations

Typically, we consider computationally-bounded adversaries (e.g., probabilistic polynomial-time adversaries). However, the private-channel model allows for the (meaningful) consideration of computationally-unbounded adversaries.

We stress that, also in the case of computationally-unbounded adversaries, security should be defined by requiring that for every real adversary, whatever the adversary can compute after participating in the execution of the actual protocol is computable *within comparable time* by an imaginary adversary participating in an imaginary execution of the trivial ideal protocol (for computing the desired functionality with the help of a trusted party). That is, although no computational restrictions are made on the real-model adversary, it is required that the ideal-model adversary that obtains the same impact does so within comparable time (i.e., within time that is polynomially related to the running time of the real-model adversary being simulated). Thus, any construction proven secure in the computationally-unbounded adversary model is (trivially) secure with respect to computationally-bounded adversaries.

2.1.4 Restricted adversarial behavior

The parameters of the model include questions like whether or not the adversary is "adaptive" and "active" (where these terms are discussed next).

Adaptive versus non-adaptive. The most general type of an adversary considered in the literature is one that may corrupt parties to the protocol while the execution goes on, and does so based on partial information it has gathered so far (cf. [17]). A somewhat more restricted model, which seems adequate in many settings, postulates that the set of dishonest parties is fixed (arbitrarily) before the execution starts (but this set is, of course, not known to the honest parties). The latter model is called *non-adaptive* as opposed to the *adaptive* adversary discussed first. Although the adaptive model is stronger, the author believes that the non-adaptive model provides a reasonable level of security in many applications.

Active versus passive. An orthogonal parameter of restriction refers to whether a dishonest party takes active steps to disrupt the execution of the protocol (i.e., sends messages that differ from those specified by the protocol), or merely gathers information (which it may latter share with the other dishonest parties). The latter adversary has been given a variety of names such as *semi-honest*, *passive*, and *honest-but-curious*. This restricted model may be justified in certain settings, since launching an unrestricted attack may not be feasible in some cases.[6] Furthermore, the passive adversary model provides a useful methodological locus (cf. [42, 43, 34] and Section 4).

[6]Note that deviation from the prescribed program requires replacing the provided software by an alternative one, whereas passive attacks can be conducted by merely monitoring the execution of the provided software. Thus, passive attacks are much easier to launch, whereas designing harmful active attacks seems much harder.

Below we refer to the adversary of the unrestricted model as to *active*; another commonly used name is *malicious*. We also mention the intermediate model of *covert* adversaries (cf. [1]: Covert adversaries may deviate arbitrarily from the prescribed behavior as long as they do not run a risk of being caught doing so.

2.1.5 Restricted notions of security

One important example is the willingness to tolerate "unfair" protocols in which the execution can be suspended (at any time) by a dishonest party, provided that it is detected doing so. We stress that in case the execution is suspended, the dishonest party does not obtain more information than it could have obtained when not suspending the execution. (What may happen is that the honest parties will not obtain their desired outputs, but rather will detect that the execution was suspended.) We stress that the motivation to this restricted model is the impossibility of obtaining general secure two-party computation in the unrestricted model.

Additional weaker (than standard) notions of security were proposed with similar motivation and include relaxing the simulation requirement (by allowing quasi-polynomial-time simulation, cf. [2]) and relaxing the indistinguishability requirement (by allowing a small but noticeable probabilistic gap, cf. [41, 52, 49]).

2.1.6 Upper bounds on the number of dishonest parties

In some models, secure multi-party computation is possible only if a majority of the parties is honest (cf. [11, 22]). Sometimes even a special majority (e.g., 2/3) is required. General "(resilient) adversarial-structures" have been considered too (cf. [51]).

Mobile adversary. In most works, once a party is declared dishonest it remains so throughout the execution. More generally, one may consider transient adversarial behavior (i.e., an adversary seizes control of some site and later withdraws from it). This model, introduced in [59], allows to construct protocols that remain secure even in case the adversary may seize control of all sites during the execution (but never control concurrently, say, more than 10% of the sites). We comment that schemes secure in this model were later termed "proactive" (cf. [18]).

2.2 Example: Multi-party protocols with honest majority

Here we consider an active, non-adaptive, computationally-bounded adversary, and do not assume the existence of private channels. Our aim is to define multi-party protocols that remain secure provided that the honest parties are in majority. (The reason for requiring a honest majority will be discussed at the end of this subsection.)

Consider any multi-party protocol. We first observe that each party may change its local input before even entering the execution of the protocol. However, this is unavoidable also when the parties utilize a trusted party. Consequently, such an effect of the adversary on the real execution (i.e., modification of its own input prior to entering the actual execution) is not considered a breach of security. In general, whatever cannot be avoided when the parties utilize a trusted party, is not considered a breach of security. We wish secure protocols (in the real model) to suffer only from whatever is unavoidable also when the parties utilize a trusted party. Thus, the basic paradigm underlying the definitions of *secure multi-party computations* amounts to requiring that the only situations that may occur in the real execution of a secure protocol are those that can also occur in a corresponding ideal model (where the parties may employ a trusted party). In other words, the "effective malfunctioning" of parties in secure protocols is restricted to what is postulated in the corresponding ideal model.

When defining secure multi-party protocols with honest majority, we need to pin-point what cannot be avoided in the ideal model (i.e., when the parties utilize a trusted party). This is easy, because the ideal model is very simple. Since we are interested in executions in which the majority of parties are honest, we consider an ideal model in which any minority group (of the parties) may collude as follows:

1. Firstly this dishonest minority shares its original inputs and decides together on replaced inputs to be sent to the trusted party. (The other parties send their respective original inputs to the trusted party.)

2. Upon receiving inputs from all parties, the trusted party determines the corresponding outputs and sends them to the corresponding parties. (We stress that the information sent between the honest parties and the trusted party is not seen by the dishonest colluding minority.)

3. Upon receiving the output-message from the trusted party, each honest party outputs it locally, whereas the dishonest colluding minority may determine their outputs based on all they know (i.e., their initial inputs and their received outputs).

Note that the foregoing behavior of the minority group is unavoidable in any execution of any protocol (even in presence of trusted parties). This is the reason that the ideal model was defined as above. Now, a *secure multi-party computation with honest majority* is required to emulate this ideal model. That is, the effect of any feasible adversary that controls a minority of the parties in a real execution of the actual protocol, can be essentially simulated by a (different) feasible adversary that controls the corresponding parties in the ideal model. That is:

Definition 1 (secure protocols – a sketch): *Let f be an m-ary functionality and Π be an m-party protocol operating in the real model.*

- *For a real-model adversary A, controlling some minority of the parties* (and tapping all communication channels)*, and an m-sequence $\overline{x}$, we denote by $\text{REAL}_{\Pi,A}(\overline{x})$ the sequence of m outputs resulting from the execution of Π on input $\overline{x}$ under attack of the adversary A.*

- *For an ideal-model adversary A', controlling some minority of the parties, and an m-sequence $\overline{x}$, we denote by $\text{IDEAL}_{f,A'}(\overline{x})$ the sequence of m outputs resulting from the ideal process described above, on input $\overline{x}$ under attack of the adversary A'.*

We say that Π securely implements f with honest majority *if for every feasible real-model adversary A, controlling some minority of the parties, there exists a feasible ideal-model adversary A', controlling the same parties, so that the probability ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$ and $\{\text{IDEAL}_{f,A'}(\overline{x})\}_{\overline{x}}$ are computationally indistinguishable.*[7]

Thus, security means that the effect of each minority group in a real execution of a secure protocol is "essentially restricted" to replacing its own local inputs (independently of the local inputs of the majority parties) before the protocol starts, and replacing its own local outputs (depending only on its local inputs and outputs) after the protocol terminates. (We stress that in the real execution

[7]Note that, as in the case of zero-knowledge, the notion of indistinguishability used here refers to probability ensembles indexed by strings and to distinguishers that are arbitrary polynomial-size circuits. That is, we refer to the definition presented in [35, Def. 3.2.7 (2)] and in [38, Def. 3].

the minority parties do obtain additional pieces of information; yet in a secure protocol they gain nothing from these additional pieces of information, because they can actually reproduce those by themselves.)

The fact that Definition 1 refers to a model without private channels is due to the fact that our (sketchy) definition of the real-model adversary allowed it to tap the channels, which in turn effects the set of possible ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$. When defining security in the private-channel model, the real-model adversary is not allowed to tap channels between honest parties, and this again effects the possible ensembles $\{\text{REAL}_{\Pi,A}(\overline{x})\}_{\overline{x}}$. On the other hand, when we wish to define security with respect to passive adversaries, both the scope of the real-model adversaries and the scope of the ideal-model adversaries changes. In the real-model execution, all parties follow the protocol but the adversary may alter the output of the dishonest parties arbitrarily depending on all their intermediate internal states (during the execution). In the corresponding ideal-model, the adversary is not allowed to modify the *inputs* of dishonest parties (in Step 1), but is allowed to modify their outputs (in Step 3).

We comment that a definition analogous to Definition 1 can be presented also in case the dishonest parties are not in minority. In fact, such a definition seems more natural, but the problem is that it cannot be satisfied in general. Furthermore, most natural functionalities do not have a protocol for computing them securely (in the foregoing sense) when at least half of the parties are dishonest and employ an adequate adversarial strategy. This follows from an impossibility result regarding two-party computation, which essentially asserts that there is no way to prevent a party from prematurely suspending the execution [24].[8] On the other hand, secure multi-party computation with dishonest majority is possible if premature suspension of the execution is not considered a breach of security (see Section 2.3).

2.3 Another example: Two-party protocols allowing abort

In light of the last paragraph, we now consider multi-party computations in which premature suspension of the execution is not considered a breach of security. For concreteness, we focus here on the special case of two-party computations.[9]

Intuitively, in any two-party protocol, each party may suspend the execution at any point in time, and furthermore it may do so as soon as it learns the desired output. Thus, in case the output of each parties depends on both inputs, it is always possible for one of the parties to obtain the desired output while preventing the other party from fully determining its own output. The same phenomenon occurs even in case the two parties just wish to generate a common random value. Thus, when considering active adversaries in the two-party setting, we do not consider such premature suspension of the execution a breach of security. Consequently, we consider an ideal model where each of the two parties may "shut-down" the trusted (third) party at any point in time. In particular, this may happen after the trusted party has supplied the outcome of the computation to one party but before it has supplied it to the other. That is, an execution in the ideal model proceeds as follows:

1. Each party sends its input to the trusted party, where the dishonest party may replace its input or send no input at all (which can be treated as sending a default value).

[8] We stress that although the foregoing impossibility result applies to many natural functionalities (e.g., coin tossing [24]), it may not apply to other natural functionalities (as demonstrated in [48]). Furthermore, partial fairness can be obtained in many other cases (cf. [49]).

[9] As in Section 2.2, we consider a non-adaptive, active, computationally-bounded adversary.

2. Upon receiving inputs from both parties, the trusted party determines the corresponding outputs, and sends the first output to the first party.

3. In case the first party is dishonest, it may instruct the trusted party to halt, otherwise it always instructs the trusted party to proceed. If instructed to proceed, the trusted party sends the second output to the second party.

4. Upon receiving the output-message from the trusted party, the honest party outputs it locally, whereas the dishonest party may determine its output based on all it knows (i.e., its initial input and its received output).

A secure two-party computation allowing abort is required to emulate this ideal model. That is, as in Definition 1, security is defined by requiring that for every feasible real-model adversary A, there exists a feasible ideal-model adversary A', controlling the same party, so that the probability ensembles representing the corresponding (real and ideal) executions are computationally indistinguishable. This means that each party's "effective malfunctioning" in a secure protocol is restricted to supplying an initial input of its choice and aborting the computation at any point in time. (Needless to say, the choice of the initial input of each party may *not* depend on the input of the other party.)

We mention that an alternative way of dealing with the problem of premature suspension of execution (i.e., abort) is to restrict our attention to single-output functionalities; that is, functionalities in which only one party is supposed to obtain an output. The definition of secure computation of such functionalities can be made identical to Definition 1, with the exception that no restriction is made on the set of dishonest parties (and in particular one may consider a single dishonest party in the case of two-party protocols). For further details, see [36, Sec. 7.2.3].

3 Some Known Results

We next list some of the models for which general secure multi-party computation is known to be attainable (i.e., models in which one can construct secure multi-party protocols for computing any desired functionality). We mention that the first set of results of this type were obtained by Goldreich, Micali, Wigderson and Yao [42, 64, 43].

3.1 In the standard cryptographic model

Assuming the existence of enhanced trapdoor permutations[10], secure multi-party computation is possible in the following models (cf. [42, 64, 43] and details in [34, 36]):

1. Passive adversary, for any number of dishonest parties (cf. [36, Sec. 7.3]).

2. Active adversary that may control only a minority of the parties (cf. [36, Sec. 7.5.4]).

3. Active adversary, for any number of bad parties, provided that suspension of execution is not considered a violation of security (i.e., as discussed in Section 2.3). (See [36, Sec. 7.4 and 7.5.5].)

[10] Loosely speaking, the enhancement refers to the hardness condition of a standard collection of trapdoor permutations, denoted $\{f_i : D_i \to \{0,1\}^*\}_{i\in\overline{I}}$, and requires that it be hard to recover $f_i^{-1}(y)$ also when given the coins used to sample y (rather than merely y itself). See [36, Apdx. C.1].

In all these cases, the adversary is computationally-bounded and non-adaptive. On the other hand, the adversary may tap the communication lines between honest parties (i.e., we do not assume "private channels" here).

The results for active adversaries assume a broadcast channel. Indeed, the latter can be implemented (while tolerating any number of bad parties) using a signature scheme and assuming a public-key infrastructure (or that each party knows the verification-key corresponding to each of the other parties).

3.2 In the private channels model

Making no computational assumptions and allowing computationally-unbounded adversaries, but *assuming private channels*, secure multi-party computation is possible in the following models (cf. [11, 20]):

1. Passive adversary that may control only a minority of the parties.

2. Active adversary that may control only less than one third of the parties.[11]

In both cases the adversary may be adaptive (cf. [11, 17]).

3.3 Additional models

Secure multi-party computation is possible against an active, adaptive and *mobile* adversary that may control a small constant fraction of the parties at any point in time [59]. This result makes no computational assumptions, allows computationally-unbounded adversaries, but *assumes private channels*.

Assuming the existence of trapdoor permutations, secure multi-party computation is possible in a model allowing an active and *adaptive* computationally-bounded adversary that may control only less than one third of the parties [17, 25]. We stress that this result does not assume "private channels".

Results for asynchronous communication and arbitrary networks of point-to-point channels were presented in [10, 12] and [27], respectively.

3.4 Additional comments

Note that the implementation of a broadcast channel can be cast as a cryptographic protocol problem (i.e., for the functionality $(v, \lambda, ..., \lambda) \mapsto (v, v, ..., v)$, where λ denotes the empty string). Thus, it is not surprising that the results regarding active adversaries either assume the existence of such a channel or require a setting in which the latter can be implemented.

Secure reactive computation: All the above results extend to a reactive model of computation in which each party interacts with a high-level process (or application). The high-level process supplies each party with a sequence of inputs, one at a time, and expect to receive corresponding outputs from the parties. That is, a reactive system goes through (a possibly unbounded number of) iterations of the following type:

- Parties are given inputs for the current iteration.

[11] Fault-tolerance can be increased to a regular minority if a broadcast channel exists [61].

- Depending on the current inputs, the parties are supposed to compute outputs for the current iteration. That is, the outputs in iteration j are determined by the inputs of the jth iteration.

A more general formulation allows the outputs of each iteration to depend also on a global state, which is possibly updated in each iteration. The global state may include all inputs and outputs of previous iterations, and may only be partially known to individual parties. (In a secure reactive computation such a global state may be maintained by all parties in a "secret sharing" manner.) For further discussion, see [36, Sec. 7.7.1].

Efficiency considerations: One important efficiency measure regarding protocols is the number of communication rounds in their execution. The aforementioned results were originally obtained using protocols that use an unbounded number of rounds. In some cases, subsequent works obtained secure constant-round protocols: for example, in the case of multi-party computations with honest majority (cf. [6]) and in the case of two-party computations allowing abort (cf. [53]). Other important efficiency considerations include the total number of bits sent in the execution of a protocol, and the local computation time. The (communication and computation) complexities of the aforementioned secure protocols are related to the *computational* complexity of the computation, but alternative relations (e.g., where the complexities of the secure protocols are related to the (insecure) *communication* complexity of the computation) may be possible (cf. [58]).

Theory versus practice (or general versus specific): This article is focused on presenting general notions and general feasibility results. Needless to say, practical solutions to specific problems (e.g., voting [50], secure payment systems [7], and threshold cryptosystems [31]) are typically derived by specific constructions (and not by applying general results of the abovementioned type). Still, the (abovementioned) general results are of great importance to practice because they characterize a wide class of security problems that are solvable in principle, and provide techniques that may be useful also towards constructing reasonable solutions to specific problems.

4 Construction Paradigms and Two Simple Protocols

We briefly sketch a couple of paradigms used in the construction of secure multi-party protocols. We focus on the construction of secure protocols for the model of computationally-bounded and non-adaptive adversaries [42, 64, 43]. These constructions proceed in two steps (see details in [34, 36]). First a secure protocol is presented for the model of passive adversaries (for any number of dishonest parties), and next such a protocol is "compiled" into a protocol that is secure in one of the two models of active adversaries (i.e., either in a model allowing the adversary to control only a minority of the parties or in a model in which premature suspension of the execution is not considered a violation of security). These two steps are presented in the following two corresponding subsections, in which we also present two relatively simple protocols for two specific tasks, which are used extensively in the general protocols.

Recall that in the model of passive adversaries, all parties follow the prescribed protocol, but at termination the adversary may alter the outputs of the dishonest parties depending on all their intermediate internal states (during the execution). Below, we refer to protocols that are secure in the model of passive (resp., active) adversaries by the term passively-secure (resp., actively-secure).

4.1 Constructing passively-secure protocols

For any $m \geq 2$, suppose that m parties, each having a private input, wish to obtain the value of a predetermined m-argument function evaluated at their sequence of inputs. Below, we outline a passively-secure protocol for achieving this goal. We mention that the design of passively-secure multi-party protocol for any functionality (allowing different outputs to different parties as well as handling also randomized computations) reduces easily to the aforementioned task.

We present two alternative constructions of passively-secure protocols, where the first construction applies to any $m \geq 2$ and the second construction applies only to the two-party case (i.e., $m = 2$). Furthermore, while the protocols resulting from the first construction are symmetric with respect to the operation of the m parties, the protocols resulting from the second construction are highly asymmetric. This asymmetry offers various advantages (cf. [55] and the references therein).

4.1.1 Passively-secure computation with shares

We assume that the parties hold a circuit for computing the value of the function on inputs of the adequate length, and that the circuit contains only `and` and `not` gates. The key idea is to have each party "secretly share" its input with everybody else, and "secretly transform" shares of the input wires of the circuit into shares of the output wires of the circuit, thus obtaining shares of the outputs (which allows for the reconstruction of the actual outputs). The value of each wire in the circuit is shared in a way such that all shares yield the value, whereas lacking even one of the shares keeps the value totally undetermined. That is, we use a simple secret sharing scheme (cf. [63]) such that a bit b is shared by a random sequence of m bits that sum-up to $b \bmod 2$. First, each party shares each of its input bits with all parties (by secretly sending each party a random value and setting its own share accordingly). Next, all parties jointly scan the circuit from its input wires to the output wires, processing each gate as follows:

- When encountering a gate, the parties already hold shares of the values of the wires entering the gate, and their aim is to obtain shares of the value of the wires exiting the gate.
- For a `not`-gate this is easy: the first party just flips the value of its share, and all other parties maintain their shares.
- Since an `and`-gate corresponds to multiplication modulo 2, the parties need to securely compute the following randomized functionality (in which the x_i's denote shares of one entry-wire, the y_i's denote shares of the second entry-wire, the z_i's denote shares of the exit-wire, and the shares indexed by i belongs to Party i):

$$((x_1, y_1), ..., (x_m, y_m)) \;\mapsto\; (z_1, ..., z_m) \tag{1}$$

$$\text{where } \textstyle\sum_{i=1}^{m} z_i = (\sum_{i=1}^{m} x_i) \cdot (\sum_{i=1}^{m} y_i), \tag{2}$$

  and all arithmetic operations are mod 2. That is, the z_i's are random subject to Eq. (2).

Finally, the parties send their shares of each circuit-output wire to the designated party, which reconstructs the value of the corresponding bit. Thus, the parties have propagated shares of the input wires into shares of the output wires, by repeatedly conducting privately-secure computation of the m-ary functionality of Eq. (1) & (2). That is, securely evaluating the entire (arbitrary) circuit "reduces" to securely conducting a specific (very simple) multi-party computation. But things get even simpler: the key observation is that

$$\left(\sum_{i=1}^{m} x_i\right) \cdot \left(\sum_{i=1}^{m} y_i\right) = \sum_{i=1}^{m} x_i y_i + \sum_{1 \leq i < j \leq m} (x_i y_j + x_j y_i) \tag{3}$$

Thus, the m-ary functionality of Eq. (1) & (2) can be computed as follows:

1. Each Party i locally computes $z_{i,i} \stackrel{\text{def}}{=} x_i y_i$.

2. Next, each pair of parties (i.e., Parties i and j) securely compute random shares of $x_i y_j + y_i x_j$. That is, Parties i and j (holding (x_i, y_i) and (x_j, y_j), respectively), need to securely compute the randomized two-party functionality $((x_i, y_i), (x_j, y_j)) \mapsto (z_{i,j}, z_{j,i})$, where the z's are random subject to $z_{i,j} + z_{j,i} = x_i y_j + y_i x_j$. Equivalently, Party j uniformly selects $z_{j,i} \in \{0,1\}$, and Parties i and j securely compute the deterministic functionality $((x_i, y_i), (x_j, y_j, z_{j,i})) \mapsto (z_{j,i} + x_i y_j + y_i x_j, \lambda)$, where λ denotes the empty string.

   The latter simple two-party computation can be securely implemented using a 1-out-of-4 Oblivious Transfer (cf. [44] and [36, Sec. 7.3.3]), which in turn can be implemented using enhanced trapdoor permutations (see below). Loosely speaking, a 1-out-of-k Oblivious Transfer is a protocol enabling one party to obtain one of k secrets held by another party, without the second party learning which secret was obtained by the first party. That is, we refer to the two-party functionality
$$(i, (s_1, ..., s_k)) \mapsto (s_i, \lambda) \tag{4}$$
   Note that any deterministic functionality of the form $f : [k] \times \{0,1\}^* \to \{0,1\}^* \times \{\lambda\}$ can be privately-computed by invoking a 1-out-of-k Oblivious Transfer on inputs i and $(f(1, y), ..., f(k, y))$, where i (resp., y) is the initial input of the first (resp., second) party.

3. Finally, for every $i = 1, ..., m$, summing-up all the $z_{i,j}$'s yields the desired share of Party i.

The above construction is analogous to a construction that was briefly described in [43]. A detailed description and full proofs appear in [34, 36].

We mention that an analogous construction has been subsequently used in the private channel model and withstands computationally unbounded active (resp., passive) adversaries that control less than one third (resp., a minority) of the parties [11]. The basic idea is to use a more sophisticated secret sharing scheme; specifically, via a low degree polynomial [63]. That is, the Boolean circuit is viewed as an arithmetic circuit over a finite field having more than m elements, and a secret element s of the field is shared by selecting uniformly a polynomial of degree $d = \lfloor (m-1)/3 \rfloor$ (resp., degree $d = \lfloor (m-1)/2 \rfloor$) having a free-term equal to s, and handing each party the value of this polynomial evaluated at a different (fixed) point (e.g., party i is given the value at point i). Addition is emulated by (local) point-wise addition of the (secret sharing) polynomials representing the two inputs (using the fact that for polynomials p and q, and any field element e (and in particular $e = 0, 1, ..., m$), it holds that $p(e) + q(e) = (p + q)(e)$). The emulation of multiplication is more involved and requires interaction (because the product of polynomials yields a polynomial of higher degree, and thus the polynomial representing the output cannot be the product of the polynomials representing the two inputs). Indeed, the aim of the interaction is to turn the shares of the product polynomial into shares of a degree d polynomial that has the same free-term as the product polynomial (which is of degree $2d$). This can be done using the fact that the coefficients of a polynomial are a linear combination of its values at sufficiently many arguments (and the other way around), and the fact that one can privately-compute any linear combination (of secret values). For details see [11, 32].

A passively-secure 1-out-of-k Oblivious Transfer. Using a collection of enhanced trapdoor permutations, denoted $\{f_\alpha : D_\alpha \to D_\alpha\}_{\alpha \in \overline{I}}$ (along with a corresponding hard-core predicate [40], denoted b), we outline a passively-secure implementation of the functionality of Eq. (4). The implementation originates in [30] (and a full description is provided in [36, Sec. 7.3.2]).[12]

Inputs: The sender has input $(\sigma_1, \sigma_2, ..., \sigma_k) \in \{0,1\}^k$, the receiver has input $i \in \{1, 2, ..., k\}$.

Step S1: The sender selects at random permutations $f_{\alpha_1}, .., f_{\alpha_k}$ along with corresponding trapdoors, denoted $t_1, ..., t_k$, and sends the permutations (i.e., their indices $\alpha_1, ..., \alpha_k$) to the receiver.

Step R1: Upon receiving $(\alpha_1, ..., \alpha_k)$, the receiver uniformly and independently selects $x_j \in D_{\alpha_j}$ for every $j \in \{1, ..., k\}$, sets $y_i = f_{\alpha_i}(x_i)$ and $y_j = x_j$ for every $j \neq i$, and sends $(y_1, y_2, ..., y_k)$ to the sender.

Thus, the receiver knows $f_{\alpha_i}^{-1}(y_i) = x_i$, but cannot predict $b(f_{\alpha_j}^{-1}(y_j))$ for any $j \neq i$. Of course, the last assertion presumes that the receiver follows the protocol (i.e., is semi-honest).

Step S2: Upon receiving $(y_1, y_2, ..., y_k)$, using the inverting-with-trapdoor algorithm (and the trapdoors $t_1, ..., t_k$), the sender computes $z_j = f_{\alpha_j}^{-1}(y_j)$, for every $j \in \{1, ..., k\}$. It sends the k-tuple $(\sigma_1 \oplus b(z_1), \sigma_2 \oplus b(z_2), ..., \sigma_k \oplus b(z_k))$ to the receiver.

Step R2: Upon receiving $(c_1, c_2, ..., c_k)$, the receiver locally outputs $c_i \oplus b(x_i)$.

We first observe that the above protocol correctly computes 1-out-of-k Oblivious Transfer; that is, the receiver's local output (i.e., $c_i \oplus b(x_i)$) indeed equals $(\sigma_i \oplus b(f_{\alpha_i}^{-1}(f_{\alpha_i}(x_i)))) \oplus b(x_i) = \sigma_i$. Next, we offer some intuition as to why the above protocol constitutes a privately-secure implementation of 1-out-of-k Oblivious Transfer. Intuitively, the sender gets no information from the execution because, for any possible value of i, the senders sees the same distribution; specifically, a k-sequence that is uniformly distributed in $D_{\alpha_1} \times \cdots \times \cdots \times D_{\alpha_k}$. (Indeed, the key observation is that applying f_α to a uniformly distributed element of D_α yields a uniformly distributed element of D_α.) Intuitively, the receiver gains no computational knowledge from the execution because, for $j \neq i$, the only information that the receiver has regarding σ_j is the triplet $(\alpha_j, x_j, \sigma_j \oplus b(f_{\alpha_j}^{-1}(x_j)))$, where x_j is uniformly distributed in D_α, and from this information it is infeasible to predict σ_j better than by a random guess. The latter intuition presumes that sampling D_α is trivial (i.e., that there is an easily computable correspondence between the coins used for sampling and the resulting sample), whereas in general the coins used for sampling may be hard to compute from the corresponding outcome (which is the reason that an enhanced hardness assumption is used in the general analysis of the the above protocol). (See [36, Sec. 7.3.2] for an actual proof of security.)

4.1.2 Passively-secure computation with "scrambled circuits"

The following technique refers mainly to two-party computation; that is, we assume here that $m = 2$. The idea is to have one party construct an "scrambled" form of the circuit so that the other party can propagate encrypted values through the "scrambled gates" and obtain the output in the clear (while all intermediate values remain secret). Note that the roles of the two parties are not symmetric, and recall that we are describing a protocol that is secure (only) with respect to passive adversaries. An implementation of this idea proceeds as follows:

- *Constructing a "scrambled" circuit*: The first party constructs a "scrambled" form of the original circuit. The "scrambled" circuit consists of *pairs of encrypted secrets* that correspond

[12] The following presentation differs from the one in [36, Sec. 7.3.2] in that k different permutations are used rather than one. As pointed out by Ron Rothblum, the version of [36, Sec. 7.3.2] is secure only in the case that $k = 2$ (which does suffice via additional reductions).

to the wires of the original circuit and *gadgets* that correspond to the gates of the original circuit. The secrets associated with the wires entering a gate are used (in the gadget that corresponds to this gate) as keys in the encryption of the secrets associated with the wire exiting this gate. Furthermore, there is a *random correspondence* between each pair of secrets and the Boolean values (of the corresponding wire). That is, wire w is assigned a pair of secrets, denoted (s'_w, s''_w), and there is a random 1-1 mapping, denoted ν_w, between this pair and the pair of Boolean values (i.e., $\{\nu_w(s'_w), \nu_w(s''_w)\} = \{0,1\}$).

Each gadget is constructed such that knowledge of a secret that correspond to each wire entering the corresponding gate (in the circuit) yields a secret corresponding to the wire that exits this gate. Furthermore, the reconstruction of secrets using each gadget respects the functionality of the corresponding gate. For example, if one knows the secret that corresponds to the 1-value of one entry-wire and the secret that corresponds to the 0-value of the other entry-wire, and the gate is an OR-gate, then one obtains the secret that corresponds to the 1-value of exit-wire.

Specifically, each gadget consists of 4 templets that are presented in a random order, where each templet corresponds to one of the 4 possible values of the two entry-wires. A templet may be merely a double encryption of the secret that corresponds to the appropriate output value, where the double encryption uses as keys the two secrets that correspond to the input values. That is, suppose a gate computing $f : \{0,1\}^2 \to \{0,1\}$ has input wires w_1 and w_2, and output wire w_3. Then, each of the four templets of this gate has the form $E_{s_{w_1}}(E_{s_{w_2}}(s_{w_3}))$, where $f(\nu_{w_1}(s_{w_1}), \nu_{w_2}(s_{w_2})) = \nu_{w_3}(s_{w_3})$.

- *Sending the "scrambled" circuit*: The first party sends the "scrambled" circuit to the second party. In addition, the first party sends to the second party the secrets that correspond to its own (i.e., the first party's) input bits (but not the values of these bits). The first party also reveals the correspondence between the pair of secrets associated with each output (i.e., circuit-output wire) and the Boolean values.[13] We stress that the random correspondence between the pair of secrets associated with each other wire and the Boolean values is kept secret (by the first party).

- *Oblivious Transfer of adequate secrets*: Next, the first party uses a 1-out-of-2 Oblivious Transfer protocol (see Eq. (4)) in order to hand the second party the secrets corresponding to the second party's input bits (without the first party learning anything about these bits).

- *Locally evaluating the "scrambled" circuit*: Finally, the second party "evaluates" the "scrambled" circuit gate-by-gate, starting from the top (circuit-input) gates (for which it knows one secret per each wire) and ending at the bottom (circuit-output) gates (for which, by construction, the correspondence of secrets to values is known). Thus, the second party obtains the output value of the circuit (but nothing else), and sends it to the first party.

For more details, see [55].

4.2 Compilation of passively-secure protocols into actively-secure ones

We show how to transform any passively-secure protocol into a corresponding actively-secure protocol. The communication model in both protocols consists of a single broadcast channel. Note

[13] This can be done by providing, for each output wire, a succinct 2-partition (of all strings) that separates the two secrets associated with this wire.

that the messages of the original protocol may be assumed to be sent over a broadcast channel, because the adversary may see them anyhow (by tapping the point-to-point channels), and because a broadcast channel is trivially implementable in the case of passive adversaries. As for the resulting actively-secure protocol, the broadcast channel it uses can be implemented via an (authenticated) Byzantine Agreement protocol [28, 54], thus providing an emulation of this model on the standard point-to-point model (in which a broadcast channel does not exist). We mention that authenticated Byzantine Agreement is typically implemented using a signature scheme (and assuming that each party knows the verification-key corresponding to each of the other parties).

Turning to the transformation itself, the main idea is to use zero-knowledge proofs in order to force parties to behave in a way that is consistent with the (passively-secure) protocol. Actually, we need to confine each party to a unique consistent behavior (i.e., according to some fixed local input and a sequence of coin tosses), and to guarantee that a party cannot fix its input (and/or its coins) in a way that depends on the inputs of honest parties. Thus, some preliminary steps have to be taken before the step-by-step emulation of the original protocol may start. Specifically, the compiled protocol (which like the original protocol is executed over a broadcast channel) proceeds as follows:

1. *Committing to the local input*: Prior to the emulation of the original protocol, each party commits to its input (using a commitment scheme [57]). In addition, using a zero-knowledge proof-of-knowledge [47, 9, 42], each party also proves that it knows its own input; that is, that it can decommit to the commitment it sent. (These zero-knowledge proof-of-knowledge are conducted sequentially to prevent dishonest parties from setting their inputs in a way that depends on inputs of honest parties; a more round-efficient method was presented in [23].)

2. *Generation of local random tapes*: Next, all parties jointly generate a sequence of random bits for each party such that only this party knows the outcome of the random sequence generated for it, but everybody gets a commitment to this outcome. These sequences will be used as the random-inputs (i.e., sequence of coin tosses) for the original protocol. Each bit in the random-sequence generated for Party X is determined as the exclusive-or of the outcomes of instances of an (augmented) coin-tossing protocol (cf. [13] and [36, Sec. 7.4.3.5]) that Party X plays with each of the other parties. The latter protocol provides the other parties with a commitment to the outcome obtained by Party X.

3. *Effective prevention of premature termination*: In addition, when compiling (the passively-secure protocol to an actively-secure protocol) *for the model that allows the adversary to control only a minority of the parties*, each party shares its input and random-input with all other parties using a "Verifiable Secret Sharing" (VSS) protocol (cf. [21] and [36, Sec. 7.5.5.1]). Loosely speaking, a VSS protocol allows to share a secret in a way that enables each participant to verify that the share it got fits the publicly posted information, which includes (on top of the commitments posted in Steps 1 and 2) commitments to all shares. The use of VSS guarantees that if Party X prematurely suspends the execution, then the honest parties can together reconstruct all Party X's secrets and carry on the execution while playing its role. This step effectively prevents premature termination, and is not needed in a model that does not consider premature termination a breach of security.

4. *Step-by-step emulation of the original protocol*: After all the foregoing steps were completed, we turn to the main step in which the new protocol emulates the original one. In each step, each party augments the message determined by the original protocol with a zero-knowledge proof that asserts that the message was indeed computed correctly. Recall that the next

message (as determined by the original protocol) is a function of the sender's own input, its random-input, and the messages it has received so far (where the latter are known to everybody because they were sent over a broadcast channel). Furthermore, the sender's input is determined by its commitment (as sent in Step 1), and its random-input is similarly determined (in Step 2). Thus, the next message (as determined by the original protocol) is a function of publicly known strings (i.e., the said commitments as well as the other messages sent over the broadcast channel). Moreover, the assertion that the next message was indeed computed correctly is an NP-assertion, and the sender knows a corresponding NP-witness (i.e., its own input and random-input as well as the corresponding decommitment information). Thus, the sender can prove in zero-knowledge (to each of the other parties) that the message it is sending was indeed computed according to the original protocol.

The above compilation was first outlined in [42, 43]. A detailed description and full proofs appear in [34, 36].

A secure coin-tossing protocol. Using a commitment scheme, we outline a secure (ordinary as opposed to augmented) coin-tossing protocol, which originates in [13].

Step C1: Party 1 uniformly selects $\sigma \in \{0,1\}$ and sends Party 2 a commitment, denoted c, to σ.

Step C2: Party 2 uniformly selects $\sigma' \in \{0,1\}$, and sends σ' to Party 1.

Step C3: Party 1 outputs the value $\sigma \oplus \sigma'$, and sends σ along with the decommitment information, denoted d, to Party 2.

Step C4: Party 2 checks whether or not (σ, d) fit the commitment c it has obtained in Step 1. It outputs $\sigma \oplus \sigma'$ if the check is satisfied and halts with output $\perp$ otherwise (indicating that Party 1 has essentially aborted the protocol prematurely).

Outputs: Party 1 always outputs $b \stackrel{\text{def}}{=} \sigma \oplus \sigma'$, whereas Party 2 either outputs b or $\perp$.

Intuitively, Steps C1–C2 may be viewed as "tossing a coin into the well". At this point (i.e., after Step C2) the value of the coin is determined (essentially as a random value), but only one party (i.e., Party 1) "can see" (i.e., knows) this value. Clearly, if both parties are honest then they both output the same uniformly chosen bit, recovered in Steps C3 and C4, respectively. Intuitively, each party can guarantee that the outcome is uniformly distributed, and Party 1 can cause premature termination by improper execution of Step 3. Formally, we have to show how the effect of every real-model adversary can be simulated by an adequate ideal-model adversary (which is allowed premature termination). This is done in [36, Sec. 7.4.3.1].

5 Security of Concurrent Executions

The definitions and results surveyed so far refer to a setting in which, at each time, only a single execution of a cryptographic protocol takes place (or only one execution may be controlled by the adversary). In contrast, in many distributed settings (e.g., the Internet), many executions are taking place concurrently (and several of them may be controlled by the same adversary). Furthermore, it is undesirable (and sometimes even impossible) to coordinate these executions (so to effectively enforce a single-execution setting). Still, the definitions and results obtained in the single-execution setting serve as a good starting point for the study of security in the setting of concurrent executions.

As in the case of stand-alone security, the notion of zero-knowledge provides a good test case for the study of concurrent security. Indeed, in order to demonstrate the security issues arising from concurrent execution of protocols, we consider the concurrent execution of zero-knowledge protocols. Specifically, we consider a party P holding a random (or rather pseudorandom) function $f:\{0,1\}^{2n} \to \{0,1\}^n$, and willing to participate in the following protocol (with respect to security parameter n).[14] The other party, called A for adversary, is supposed to send P a binary value $v \in \{1,2\}$ specifying which of the following cases to execute:

For $v=1$: Party P uniformly selects $\alpha \in \{0,1\}^n$, and sends it to A, which is supposed to reply with a pair of n-bit long strings, denoted (β,γ). Party P checks whether or not $f(\alpha\beta)=\gamma$. In case equality holds, P sends A some secret information (e.g., the secret-key corresponding to P's public-key).

For $v=2$: Party A is supposed to uniformly select $\alpha \in \{0,1\}^n$, and sends it to P, which selects uniformly $\beta \in \{0,1\}^n$, and replies with the pair $(\beta, f(\alpha\beta))$.

Observe that P's strategy (in each case) is zero-knowledge (even w.r.t auxiliary-inputs): Intuitively, if the adversary A chooses the case $v=1$, then it is infeasible for A to guess a passing pair (β,γ) with respect to a random α selected by P. Thus, except with negligible probability (when it may get secret information), A does not obtain anything from the interaction. On the other hand, if the adversary A chooses the case $v=2$, then it obtains a pair that is indistinguishable from a uniformly selected pair of n-bit long strings (because β is selected uniformly by P, and for any α the value $f(\alpha\beta)$ looks random to A). In contrast, if the adversary A can conduct two concurrent executions with P, then it may learn the desired secret information: In one session, A sends $v=1$ while in the other it sends $v=2$. Upon receiving P's message, denoted α, in the first session, A sends it as its own message in the second session, obtaining a pair $(\beta, f(\alpha\beta))$ from P's execution of the second session. Now, A sends the pair $(\beta, f(\alpha\beta))$ to the first session of P, this pair passes the check, and so A obtains the desired secret.

An attack of the foregoing type is called a *relay attack*: During such an attack the adversary just invokes two executions of the protocol and relays messages between them (without any modification). However, in general, the adversary in a concurrent setting is not restricted to relay attacks. For example, consider a minor modification to the above protocol so that, in case $v=2$, party P replies with (say) the pair $(\beta, f(\overline{\alpha}\beta))$, where $\overline{\alpha} = \alpha \oplus 1^{|\alpha|}$, rather than with $(\beta, f(\alpha\beta))$. The modified strategy P is zero-knowledge and it also withstands a relay attack, but it can be "abused" easily by a more general concurrent attack.

The foregoing example is merely the tip of an iceberg, but it suffices for introducing the main lesson: *an adversary attacking several concurrent executions of the same protocol may be able to cause more damage than by attacking a single execution* (or several sequential executions) of the same protocol. This leads to the need to define resilience to such attacks (i.e., define security of concurrent executions), and provide protocols that satisfy the corresponding definition of security.

5.1 Definitional treatment

One may say that a protocol is concurrently secure if whatever the adversary may obtain by invoking and controlling parties in real concurrent executions of the protocol is also obtainable by a

[14] In fact, assuming that P shares a pseudorandom function f with his friends, the foregoing protocol is an abstraction of a natural "mutual identification" protocol. (The example is adapted from [39].)

corresponding adversary that controls corresponding parties making concurrent functionality calls to a trusted party (in a corresponding ideal model).[15] More generally, one may consider concurrent executions of many sessions of *several* protocols, and say that a *set of protocols* is concurrently secure if whatever the adversary may obtain by invoking and controlling such real concurrent executions is also obtainable by a corresponding adversary that invokes and controls concurrent calls to a trusted party (in a corresponding ideal model). Consequently, a protocol is said to be secure with respect to concurrent compositions if adding this protocol to *any set* of concurrently secure protocols yields a set of concurrently secure protocols.

A much more appealing approach was suggested by Canetti [16]. Loosely speaking, Canetti suggests to consider a protocol to be secure (called *environmentally-secure* (or *Universally Composable secure* [16])) only if it remains secure when executed within any (feasible) environment. Following the simulation paradigm, we get the following definition:

Definition 2 (environmentally-secure protocols [16] – a rough sketch): *Let f be an m-ary functionality and Π be an m-party protocol, and consider the following real and ideal models.*

In the real model *the adversary controls some of the parties in an execution of Π and all parties can communicate with an arbitrary probabilistic polynomial-time process, which is called an* environment (*and possibly represents other executions of various protocols that are taking place concurrently*). *Honest parties only communicate with the environment before the execution starts and when it ends; they merely obtain their inputs from the environment and pass their outputs to it. In contrast, dishonest parties may communicate freely with the environment, concurrently to the entire execution of Π.*

In the ideal model *the* (simulating) *adversary controls the same parties, which use an ideal* (trusted-party) *that behaves according to the functionality f (as in Section 2.2). All parties can communicate with the* (same) *environment* (as in the real model). *Indeed, the dishonest parties may communicate extensively with the environment before and after their single communication with the trusted party.*

We say that Π is an environmentally-secure protocol for computing f *if for every probabilistic polynomial-time adversary A in the real model there exists a probabilistic polynomial-time adversary A' controlling the same parties in the ideal model such that no probabilistic polynomial-time environment can distinguish the case in which it is accessed by the parties in the real execution from the case it is accessed by parties in the ideal model.*

As hinted above, the environment may account for other executions of various protocols that are taking place concurrently to the main execution being considered. The definition requires that such environments cannot distinguish the real execution from an ideal one. This means that anything that the real adversary (i.e., operating in the real model) gains from the execution and some environment, can be also obtained by an adversary operating in the ideal model and having access to the same environment. Indeed, Canetti proves that environmentally-secure protocols are secure with respect to concurrent compositions [16].

[15] One specific concern (in such a concurrent setting) is the ability of the adversary to "non-trivially correlate the outputs" of concurrent executions. This ability, called *malleability*, was first investigated by Dolev, Dwork and Naor [26]. We comment that providing a general definition of what "correlated outputs" means seems very challenging (if at all possible). Indeed the focus of [26] is on several important special cases such as encryption and commitment schemes.

5.2 Some of the known results

It is known is that environmentally-secure protocols for any functionality can be constructed for settings in which more than two-thirds of the active parties are honest [16]. This holds unconditionally for the private channel model, and under standard assumptions (e.g., allowing the construction of public-key encryption schemes) for the standard model (i.e., without private channel). The immediate consequence of this result is that general environmentally-secure multi-party computation is possible, provided that more than two-thirds of the parties are honest.

In contrast, general environmentally-secure *two-party* computation is not possible (in the standard sense).[16] Still, one can salvage general environmentally-secure two-party computation in the following reasonable model: Consider a network that contains servers that are willing to participate (as "helpers", possibly for a payment) in computations initiated by a set of (two or more) users. Now, suppose that two users wishing to conduct a secure computation can agree on a set of servers so that each user believes that more than two-thirds of the servers (in this set) are honest. Then, with the active participation of this set of servers, the two users can compute any functionality in an environmentally-secure manner.

Other reasonable models where general environmentally-secure *two-party* computation is possible include the common random-string (CRS) model [19] and variants of the public-key infrastructure (PKI) model [3]. In the CRS model, all parties have access to a universal random string (of length related to the security parameter). We stress that the entity trusted to post this universal random string is not required to take part in any execution of any protocol, and that all executions of all protocols may use the same universal random string. The PKI models considered in [3] require that each party deposits a public-key with a trusted center, while proving knowledge of a corresponding private-key. This proof may be conducted in zero-knowledge during special epochs in which no other activity takes place.

Acknowledgments

I wish to than Ron Rothblum for discovering an error in the presentation provided in [36, Sec. 7.3.2].

[16] Of course, some specific two-party computations do have environmentally-secure protocols. See [16] for several important examples (e.g., key exchange).

References

[1] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology*, Vol. 23, N. 2, April 2010.

[2] B. Barak and A. Sahai. How To Play Almost Any Mental Game Over The Net – Concurrent Composition via Super-Polynomial Simulation. In *46th IEEE Symposium on Foundations of Computer Science*, pages 543–552, 2005.

[3] B. Barak, R. Canetti and J.B. Nielsen. Universally composable protocols with relaxed set-up assumptions. In *45th IEEE Symposium on Foundations of Computer Science*, pages 186–195, 2004.

[4] D. Beaver. Foundations of Secure Interactive Computing. In *Crypto91*, Springer-Verlag Lecture Notes in Computer Science (Vol. 576), pages 377–391.

[5] D. Beaver. Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology*, Vol. 4, pages 75–122, 1991.

[6] D. Beaver, S. Micali and P. Rogaway. The Round Complexity of Secure Protocols. In *22nd ACM Symposium on the Theory of Computing*, pages 503–513, 1990. See details in [62].

[7] M. Bellare. Electronic Commerce and Electronic Payments. Webpage of a course. `http://www-cse.ucsd.edu/users/mihir/cse291-00/`

[8] M. Bellare, R. Canetti and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key-Exchange Protocols. In *30th ACM Symposium on the Theory of Computing*, pages 419–428, 1998.

[9] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *Crypto92*, Springer-Verlag Lecture Notes in Computer Science (Vol. 740), pages 390–420.

[10] M. Ben-Or, R. Canetti and O. Goldreich. Asynchronous Secure Computation. In *25th ACM Symposium on the Theory of Computing*, pages 52–61, 1993. See details in [14].

[11] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.

[12] M. Ben-Or, B. Kelmer and T. Rabin. Asynchronous Secure Computations with Optimal Resilience. In *13th ACM Symposium on Principles of Distributed Computing*, pages 183–192, 1994.

[13] M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, pages 133–137, February 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[14] R. Canetti. *Studies in Secure Multi-Party Computation and Applications.* Ph.D. Thesis, Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel, June 1995. Available from `http://www.wisdom.weizmann.ac.il/~oded/PS/ran-phd.ps`.

[15] R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, Vol. 13, No. 1, pages 143–202, 2000.

[16] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001. Full version (with different title) is available from *Cryptology ePrint Archive*, Report 2000/067.

[17] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-party Computation. In *28th ACM Symposium on the Theory of Computing*, pages 639–648, 1996.

[18] R. Canetti and A. Herzberg. Maintaining Security in the Presence of Transient Faults. In *Crypto94*, Springer-Verlag Lecture Notes in Computer Science (Vol. 839), pages 425–439.

[19] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *34th ACM Symposium on the Theory of Computing*, pages 494–503, 2002.

[20] D. Chaum, C. Crépeau and I. Damgård. Multi-party unconditionally Secure Protocols. In *20th ACM Symposium on the Theory of Computing*, pages 11–19, 1988.

[21] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th IEEE Symposium on Foundations of Computer Science*, pages 383–395, 1985.

[22] B. Chor and E. Kushilevitz. A Zero-One Law for Boolean Privacy. *SIAM J. on Disc. Math.*, Vol. 4, pages 36–47, 1991.

[23] B. Chor and M.O. Rabin. Achieving independence in logarithmic number of rounds. In *6th ACM Symposium on Principles of Distributed Computing*, pages 260–268, 1987.

[24] R. Cleve. Limits on the Security of Coin Flips when Half the Processors are Faulty. In *18th ACM Symposium on the Theory of Computing*, pages 364–369, 1986.

[25] I. Damgard and J. B. Nielsen. Improved non-committing encryption schemes based on general complexity assumption. In *Crypto00*, Springer-Verlag Lecture Notes in Computer Science (Vol. 1880), pages 432–450.

[26] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, Vol. 30, No. 2, pages 391–437, 2000. Preliminary version in *23rd STOC*, 1991.

[27] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the ACM*, Vol. 40 (1), pages 17–47, 1993.

[28] D. Dolev and H.R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, Vol. 12, pages 656–666, 1983.

[29] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th ACM Symposium on the Theory of Computing*, pages 409–418, 1998.

[30] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, Vol. 28, No. 6, 1985, pages 637–647.

[31] P.S. Gemmell. An Introduction to Threshold Cryptography. In *CryptoBytes*, RSA Lab., Vol. 2, No. 3, 1997.

[32] R. Gennaro, M. Rabin and T. Rabin. Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography. In *17th ACM Symposium on Principles of Distributed Computing*, pages 101–112, 1998.

[33] O. Goldreich. A Uniform Complexity Treatment of Encryption and Zero-Knowledge. *Journal of Cryptology*, Vol. 6, No. 1, pages 21–53, 1993.

[34] O. Goldreich. *Secure Multi-Party Computation.* Working draft, June 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[35] O. Goldreich. *Foundations of Cryptography – Basic Tools.* Cambridge University Press, 2001.

[36] O. Goldreich. *Foundations of Cryptography – Basic Applications.* Cambridge University Press, 2004.

[37] O. Goldreich. Foundations of Cryptography – A Primer. *Foundations and Trends in Theoretical Computer Science*, Volume 1, Issue 1, 2005.

[38] O. Goldreich. Zero-Knowledge Twenty Years After its Invention. *Quaderni di Matematica*, Vol. 13 (Complexity of Computations and Proofs, ed. J. Krajicek), pages 249–304, 2004. See also *ECCC*, TR02-063, 2002.

[39] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, Vol. 25, No. 1, February 1996, pages 169–192.

[40] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[41] O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. *Journal of Cryptology*, Vol. 19, No. 3, pages 241–340, 2006.

[42] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 1, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[43] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing,* pages 218–229, 1987. See details in [34].

[44] O. Goldreich and R. Vainish. How to Solve any Protocol Problem – An Efficiency Improvement. In *Crypto87*, Springer Verlag, Lecture Notes in Computer Science (Vol. 293), pages 73–86.

[45] S. Goldwasser and L.A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Crypto90*, Springer-Verlag Lecture Notes in Computer Science (Vol. 537), pages 77–93.

[46] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[47] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985.

[48] S.D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete Fairness in Secure Two-Party Computation. In *40th STOC*, pages 413–422, 2008.

[49] S.D. Gordon and J. Katz. Partial Fairness in Secure Two-Party Computation. To appear in *EuroCrypt10*, 2010.

[50] R. Greenstadt. Electronic Voting Bibliography, 2000.
`http://theory.lcs.mit.edu/~cis/voting/greenstadt-voting-bibliography.html`.

[51] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. *Journal of Cryptology*, Vol. 13, No. 1, pages 31–60, 2000.

[52] J. Katz. On Achieving the "Best of Both Worlds" in Secure Multiparty Computation. In *39th STOC*, pages 11–20, 2007.

[53] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. In *Crypto01*, Springer Lecture Notes in Computer Science (Vol. 2139), pages 171–189, 2001.

[54] Y. Lindell, A. Lysyanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *34th ACM Symposium on the Theory of Computing*, pages 514–523, 2002.

[55] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Secure Two-Party Computation. *Journal of Cryptology*, Vol. 22, No. 2, pages 161–188, 2009.

[56] S. Micali and P. Rogaway. Secure Computation. In *Crypto91*, Springer-Verlag Lecture Notes in Computer Science (Vol. 576), pages 392–404. Ellaborated working draft available from the authors.

[57] M. Naor. Bit Commitment using Pseudorandom Generators. *Journal of Cryptology*, Vol. 4, pages 151–158, 1991.

[58] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *33rd ACM Symposium on the Theory of Computing*, 2001, pages 590–599.

[59] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In *10th ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[60] M.O. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[61] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In *21st ACM Symposium on the Theory of Computing*, pages 73–85, 1989.

[62] P. Rogaway. The Round Complexity of Secure Protocols. MIT Ph.D. Thesis, June 1991. Available from `http://www.cs.ucdavis.edu/~rogaway/papers`.

[63] A. Shamir. How to Share a Secret. *Communications of the ACM*, Vol. 22, Nov. 1979, pages 612–613.

[64] A.C. Yao. How to Generate and Exchange Secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.