Relaxing Environmental Security: Monitored Functionalities and Client-Server Computation

Manoj Prabhakaran[1] and Amit Sahai[2]

[1] Princeton University and UCLA
Part of this work was done while the author was an intern at IBM TJ Watson Research Center.
mp@cs.princeton.edu
[2] Princeton University and UCLA
Research done while the author was at Princeton University. Research supported in part by NSF ITR, NSF Cybertrust, and the Alfred P. Sloan Foundation.
sahai@cs.princeton.edu.

Abstract. *Definition of security under the framework of Environmental Security (a.k.a Network-Aware Security or Universally Composable Security) typically requires "extractability" of the private inputs of parties running a protocol. Formalizing concepts that appeared in an earlier work [19], we introduce a framework of "Monitored Functionalities," which allows us to avoid such a requirement from the security definition, while still providing very strong composition properties. We also consider a specialization of the Environmental Security framework by designating one party as a "server" and all other parties as clients. Both these contributions in the work are aimed at being able to provide weaker Environmental Security guarantees to simpler protocols. We illustrate the usability of the Monitored Functionalities framework by providing much simpler protocols in the plain model than in [19] for some limited functionalities in the server-client model.*

1 Introduction

At the onset of theoretical investigations into defining and achieving cryptographic security, idealized settings involving just one protocol were used. A highly successful theory was developed which gave satisfactory solutions to a multitude of cryptographic problems like encryption and multi-party computation, reducing the nebulous questions of security to concrete problems in computational complexity theory. Having successfully tackled these problems, this nascent field of computer science started focusing on more challenging problems, thrown its way by the new requirements of the fast changing world of information technology.

An important challenge was to enhance the theory so that it could handle less idealized settings relevant to the modern world – where there are many computers connected to an unreliable network. Rather annoyingly, this complicated the situation spectacularly. Definitions of security turned out to be inadequate as new attacks came into the picture. Protocols which were provably secure earlier either became insecure, or worse still, remained open challenges to be proven secure or insecure.

Two important works which explored these complications identified *Non-malleability* [8] and *Concurrent simulation* [9] as two most basic problems to be studied. Since then a significant amount of work went into tackling these basic aspects. While there has been quite some success in resolving many challenges, the new protocols kept getting more complicated and less efficient. A fresh look at the problem was taken in [3], which offered a comprehensive definition of security, namely that of *Environmental Security* (ES, for short) or *Network-Aware Security*. (It was introduced in [3] under the name Universally Composable (UC) Security; hence we shall refer to this version of Environmental Security as ES/UC.) ES/UC security, at once subsumes non-malleability and concurrency. This allowed simpler and intuitive compositions of protocols. Using a setup called the "common reference string," (where all parties are provided a string produced by a trusted party), or alternatively using a trust assumption of "honest majority" (where majority of players among a pre-defined subset of parties are trusted to be honest), it was shown how to do "secure multi-party computation," arguably the ultimate cryptographic task [3, 7]. However it fell short of offering a viable solution to the protocol designers: it was shown that to achieve provable security under this new model, some "trust assumption" would be necessary in most cases of interest [3, 4, 6]. Recently [19] showed a way to get around the need for trust assumption by modifying the ES/UC framework to obtain what is called the *generalized Environmental Security* (gES), or *generalized Network-Aware Security*, framework.

This Work. Environmental Security (ES/UC as well as gES) addresses the concern that protocols are run in an arbitrary environment along with other processes and network activity. However the comprehensive definitions of security offered by these frameworks tend to require complex protocols. The thesis of this work is that we need to develop relaxed notions of Environmental Security which will help us prove some level of security for simpler protocols, at least for certain limited applications. We explore two separate directions simultaneously.

First, we develop a model intended to remove some of the concerns of universal composability from environmental security. The model restricts the protocol executions for which security analysis will be applied, by requiring "fixed roles" for the participants across all protocol executions. (This can be viewed as a generalization of the setup introduced in concurrent zero knowledge [9], to the ES setting.) This restriction frees us from concerns of certain "man-in-the-middle" attacks (or malleability of the protocols). Our interest in the client-server model is as a useful theoretical construct in itself – a platform for tackling some of the Environmental Security issues without having to deal with all the composition issues. For the protocols in this work, use of this model is *not crucial*, but it leads to somewhat simpler protocols, simpler assumptions and simpler analysis.

Second – and this is the main focus in this work – we introduce a significant relaxation of the security requirements in the ES framework, in a new framework of *Monitored Functionalities*. Indeed, [19] shows how to relax ES/UC, without much loss in applicability.[3] gES removes the restriction in the ES/UC framework that the "simulation" used to define security should be computationally bounded, and still manages to retain Universal Composability. However the gES protocols in [19] are still complex.

[3] Technically, security in gES framework is not a relaxation of security in ES/UC framework, but involves a relaxation followed by a strengthening, which in general makes it incomparable with the latter.

We go one step further, and redefine security in such a way that one more requirement from the ES/UC framework is removed – namely that of "extractability." We show how to define meaningful notions of security without the extractability requirement, and yet obtain Environmentally Secure and Universally Composable protocols. This is achieved by introducing a new entity alongside the "trusted party" in the ideal world: it is a computationally unbounded *"monitor"* which can watch over the ideal world execution and raise an alarm if some specific condition is violated.

Two of our protocols (for commitment and zero-knowledge proofs) are adapted from [19], with simplifications allowed by the client-server model. The results in [19] do show that those protocols can be used to obtain full-fledged gES security for these tasks (against static adversaries, without the client-server restriction, and without resorting to monitored functionalities). However it is far from a direct use: they are used as subroutines within a larger protocol for commitment. Our attempt is to use these protocols in a more direct fashion and obtain much simpler protocols.

Our Results. We introduce a new framework of Environmental Security, where the correctness and security requirements of a protocol are separately defined (unlike [3, 19]). Further, we consider a model, called "client-server model," which considers restricted patterns of executions of the protocols analysed. Both are aimed at getting relaxations of the existing ES frameworks, so that possibly weaker levels of Environmental Security can be defined and proven for simpler protocols.

Then we show how to realize tools like commitment, zero knowledge proof and commit-and-prove functionalities in this setting. We illustrate the use of these tools in implementing a special class of functionalities called the "server-client" functionalities. All our protocols are very simple and relatively efficient (compared to protocols in ES/UC and gES models). The protocols are all in the "plain-model" (no common reference string, or other trust assumptions), and are much more efficient than even the ones in [19] (which solve a more difficult problem).

We point out that the 4 message zero knowledge protocol we give is, in particular, a *concurrent* zero knowledge argument with only 4 messages, wherein the simulator (and corrupt verifiers) are allowed some super polynomial computational power. Previous results (which worked with polynomial time simulators) gave either protocols with a large number of rounds (dependent on the security parameter), or were dependent on the number of verifiers that the protocol was secure against. Further, our protocol is a simple variant of a well-known simple protocol which has been around for many years, but for which no such strong composability has been proven till now.

Limitations of Our Results. There are some serious limitations to our current results. It is not clear if the approach in this work can directly yield protocols for the most general kind of functionalities. Firstly, our 2-party protocol is for a very special kind of multi-party computations only, which we term the server-client computation. (In a server-client computation, the client receives as output some function of its input and the server's input. But the server receives as output, the client's input.)

But a more serious limitation lies with the nature of security guarantee provided. Along with correctness and secrecy guarantees, one would like to have a guarantee that the server's input to the function is independent of the client's input. The security definition provided in this work does not make this last guarantee. Nevertheless, we sketch how this can be remedied under the condition that the client never *uses* its input previously (the full technical details of which will be published elsewhere).

Despite the limitations, our new framework is a step in the direction of formalizing relaxed notions of security (relaxed, but still accounting for a general environment), which will help prove security guarantees for simpler and more efficient protocols.

Previous Results. As mentioned above [3, 17] introduced the ES/UC framework, as a model to consider general composability and complex environments. But in the plain-model very little was available in terms of positive results. Recently, [19] introduced a modified notion of security, by allowing the IDEAL world environment and adversary access to super-polynomial powers. This made it possible to achieve secure multi-party computation in the plain model. However the protocols in [19] are still much more involved than the ones presented here. An earlier attempt at reducing the requirements of the ES/UC framework was in [5], which also introduced a semi-functionality-like notion in the context of secure Key-Exchange.

Work on concurrent model stretches back to [9], who introduced it in the context of Zero Knowledge proofs, followed by a sequence of works in the same context [20, 11, 18], where an arbitrary polynomial number of concurrent executions can be handled, but with the number of rounds in the protocol growing with the security parameter. [1] gave a constant round protocol for *bounded* concurrent ZK, in which the communication complexity depended on the number of concurrent sessions that can be handled. A similar result, but with similar limitations, was shown for *general* 2-party computations (general, as opposed to our Client-Server Computation) recently [12, 16]. All these protocols are somewhat complicated and conceptually involved. Relaxing the requirement of polynomial time simulation in the definition of security was used in some earlier works [13, 14] too, in the context of zero knowledge proofs.

Connections with [19]. Our starting point is the two "semi-functionalities" for commitment and zero-knowledge proofs, introduced in [19]. There they are used for the specific purpose of implementing a (full-fledged gES) commitment functionality. However we seek to directly use them for "end uses." Our new framework for monitored functionalities lets us extend the approach there to a formal definition of security. We introduce two more semi-functionalities, namely commit-and-prove and server-client computation. We give protocols for these semi-functionalities and also prove that these semi-functionalities have the required correctness property in our framework. We then observe that for such functionalities to be more useful, it would help if the correctness guarantees on the semi-functionalities are strengthened. We show that such a strengthening can indeed be formalized and proven (see Section 6.2).

2 Basic Ideas

The next few subsections introduce the novel tools and concepts we employ. All these are new, except the ideas in Section 2.3 (which were recently introduced in [19]).

2.1 Client-Server Model

We present the *Client-Server model* as a simplified setting to investigate some of the Environmental Security issues, without having to deal with all the composition issues.

In this model, the security guarantees will be given only to sessions of protocols in all of which the participants have the same "fixed roles." The inspiration for this model is the model used for concurrent zero knowledge proofs [9].

The specific fixed role restriction in our model is as follows. There is a special party called the *server* $\mathcal{S}$. All the other parties are considered *clients*. We shall use $\mathcal{K}$ to denote a generic client. We allow only static adversaries, i.e., parties can be corrupted only at the beginning of the system. (Recall that the concurrent ZK-model also has a static adversary.) In this model we shall typically investigate functionalities where $\mathcal{S}$ plays a special role: for instance, a commitment functionality where $\mathcal{S}$ is the committing party (and a client receives the commitment), or a zero knowledge proof where $\mathcal{S}$ is the prover. We also consider a class of multi-party computation problems where $\mathcal{S}$ has a special role. Universal composition in the client-server model is limited to concurrent sessions of the protocols where the same party plays the server in all sessions. Thus, in particular, we do not offer general non-malleability, just as in the case of concurrent zero knowledge. However unlike there, *we require environmental security*: i.e., the security definition incorporates the presence of an arbitrary environment.

Note that the client-server model does not have a different definition of security, but rather inherits it from the underlying ES model. (The new security definition we introduce in this work is given in Section 2.2). It only specifies restrictions on the protocol executions for which security will be defined.

The main purpose of introducing the client-server model is to allow simplification of protocols, by exploiting the asymmetry in the model. It allows us to use simpler assumptions, as will be described in Section 2.3.

2.2 A New Framework for Specifying Security

In the concurrent ZK-model security requirement (for concurrent ZK proofs) is specified by the two properties: zero-knowledge (secrecy) and soundness (correctness). In contrast, in the ES/UC model security requirements are specified by giving an *ideal functionality*, which totally captures both these requirements. We propose a new framework, where we still require Environmental Security for the more subtle secrecy requirement. But the correctness requirement is specified separately, as in the concurrent ZK model. Below we elaborate on this.

"Semi-Functionalities" and "Monitors." In the ES/UC-model, a 2-party IDEAL functionality usually interacts with both the parties in an ideal way. For instance the IDEAL commitment functionality would involve receiving a value from the *sender* secretly, and notifying the *receiver* (and adversary) of the receipt, and later on receiving a command to reveal from the sender, sending the original value to the receiver. This functionality makes sure that the sender is bound to a value on committing (this is the "correctness guarantee") and that the value remains secret ("secrecy guarantee"). The idea behind defining a *semi-functionality* is to free one of these requirements from the IDEAL functionality, and somehow enforce that requirement separately.

A monitored functionality (e.g., $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ described in Figure 1(a)) consists of a semi-functionality ($\mathcal{F}_{\widetilde{\text{COM}}}$ in Figure 1(a)) and some conditions on the semi-functionality. The semi-functionality is syntactically just a functionality, but it is not "ideal" enough. It is

Commitment: $\langle \mathcal{F}_{\widetilde{\mathrm{COM}}} \rangle$

Semi-Functionality $\mathcal{F}_{\widetilde{\mathrm{COM}}}$:

COMMIT PHASE

$$\begin{aligned} \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{COM}}} &: \text{Arbitrary protocol} \\ \mathcal{F}_{\widetilde{\mathrm{COM}}} \to \mathcal{K}, \mathcal{S} &: \texttt{commit} \end{aligned}$$

REVEAL PHASE

$$\begin{aligned} \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{COM}}} &: \text{Arbitrary Protocol} \\ \mathcal{F}_{\widetilde{\mathrm{COM}}} \to \mathcal{K}, \mathcal{S} &: (\texttt{reveal}, b) \\ & \quad b \text{ determined} \\ & \quad \text{arbitrarily by } \mathcal{F}_{\widetilde{\mathrm{COM}}} \end{aligned}$$

Completeness: If $\mathbb{S}$ is honest, $\mathcal{F}_{\widetilde{\mathrm{COM}}}$ never aborts.

Monitor: When $\mathcal{F}_{\widetilde{\mathrm{COM}}}$ sends the message `commit`, the monitor must record one value b^* as the committed value. If $\mathbb{S}$ is honest this value must be the intended value for commitment. Later if $\mathcal{F}_{\widetilde{\mathrm{COM}}}$ sends $(\texttt{reveal}, b)$ but $b \neq b^*$ then the monitor notifies the receiver by raising an alarm.

(a) Commitment

Zero Knowledge Proof: $\langle \mathcal{F}_{\widetilde{\mathrm{ZK}}} \rangle$

Parametrized by a polynomial time relation $R : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2} \to \{0,1\}$. Common input to $\mathbb{S}$ and $\mathcal{K}$: $x \in \{0,1\}^{\ell_1}$.

Semi-Functionality:

$$\begin{aligned} \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{ZK}}} &: \text{Arbitrary protocol} \\ \mathcal{F}_{\widetilde{\mathrm{ZK}}} \to \mathcal{K}, \mathcal{S} &: (\texttt{proven}, x) \end{aligned}$$

Completeness: If $\mathbb{S}$ is honest, $\mathcal{F}_{\widetilde{\mathrm{ZK}}}$ sends the `proven` message to $\mathcal{K}$.

Monitor: If $\mathcal{F}_{\widetilde{\mathrm{ZK}}}$ sends the message $(\texttt{proven}, x)$ to the verifier, *and* there exists no $y \in \{0,1\}^{\ell_2}$ such that $R(x,y) = 1$ then the monitor notifies the verifier by raising an alarm.

(b) Zero Knowledge Proof

Commit and Prove: $\langle \mathcal{F}_{\widetilde{\mathrm{CAP}}} \rangle$

Parametrized by a polynomial time relation $R : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2} \times \{0,1\}^{\ell_3} \to \{0,1\}$.

Semi-Functionality:

COMMIT PHASE:

$$\begin{aligned} \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{CAP}}} &: \text{Arbitrary protocol} \\ \mathcal{F}_{\widetilde{\mathrm{CAP}}} \to \mathcal{K}, \mathcal{S} &: \texttt{commit} \end{aligned}$$

PROOF PHASE (CAN BE MULTIPLE TIMES):

$$\begin{aligned} \mathbb{S} \to \mathcal{F}_{\widetilde{\mathrm{CAP}}} \to \mathcal{K}, \mathcal{S} &: x \\ \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{CAP}}} &: \text{Arbitrary Protocol} \\ \mathcal{F}_{\widetilde{\mathrm{CAP}}} \to \mathcal{K}, \mathcal{S} &: (\texttt{proven}, x) \end{aligned}$$

Completeness: Interacting with an honest prover $\mathbb{S}$, $\mathcal{F}_{\widetilde{\mathrm{CAP}}}$ never aborts.

Monitor: At the end of the commit phase, the monitor internally records a value w. If the $\mathcal{F}_{\widetilde{\mathrm{CAP}}}$ sends $(\texttt{proven}, x)$ later in the proof phase then the monitor checks if there is a value y such that $R(w, x, y) = 1$. If not it raises an alarm.

(c) Commit and Prove

Client-Server Computation: $\langle \mathcal{F}_{\widetilde{\mathrm{CSC}}} \rangle$

Parametrized by a function F.

Semi-Functionality:

$$\begin{aligned} \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{CSC}}} &: \text{Arbitrary Protocol} \\ \mathcal{F}_{\widetilde{\mathrm{CSC}}} \to \mathcal{K}, \mathcal{S} &: \texttt{commit} \\ \mathcal{K} \to \mathcal{F}_{\widetilde{\mathrm{CSC}}} \to \mathbb{S}, \mathcal{S} &: x_{\mathcal{K}} \\ \mathbb{S} \leftrightarrow \mathcal{F}_{\widetilde{\mathrm{CSC}}} &: \text{Arbitrary Protocol} \\ \mathcal{F}_{\widetilde{\mathrm{CSC}}} \to \mathcal{K}, \mathcal{S} &: z \ (z \text{ determined} \\ & \quad \text{arbitrarily by } \mathcal{F}_{\widetilde{\mathrm{CSC}}}) \end{aligned}$$

Completeness: Interacting with an honest prover $\mathbb{S}$ with input $x_{\mathbb{S}}$, if $\mathcal{K}$ sends it input $x_{\mathcal{K}}$, $\mathcal{F}_{\widetilde{\mathrm{CSC}}}$ sends $x_{\mathcal{K}}$ to $\mathbb{S}$ and $z = F(x_{\mathbb{S}}, x_{\mathcal{K}})$ to $\mathcal{K}$.

Monitor: At the end of the first step, the monitor internally records a value x^*. If $\mathcal{F}_{\widetilde{\mathrm{CSC}}}$ sends $x_{\mathcal{K}}$ to $\mathbb{S}$ and z to $\mathcal{K}$, the monitor checks if $z = F(x^*, x_{\mathcal{K}})$. If not it raises an alarm.

(d) 2-Party Client-Server Computation

Fig. 1. Monitored Functionalities $\langle \mathcal{F}_{\widetilde{\mathrm{COM}}} \rangle$, $\langle \mathcal{F}_{\widetilde{\mathrm{ZK}}} \rangle$, $\langle \mathcal{F}_{\widetilde{\mathrm{CAP}}} \rangle$ and $\langle \mathcal{F}_{\widetilde{\mathrm{CSC}}} \rangle$.

typically defined based on an arbitrary protocol. For instance the specification of $\mathcal{F}_{\widetilde{\text{COM}}}$ consists of arbitrary interaction between the server and $\mathcal{F}_{\widetilde{\text{COM}}}$ (which is unspecified in Figure 1(a), but will be later specified in such a way that binding property can be argued separately). Note that the arbitrary protocol is carried out *between the semi-functionality and a party, and not between the two parties*. This is why the semi-functionality guarantees secrecy – in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$, the only message it sends to the receiver and the adversary before the reveal phase is `commit`. To complete the specification of the ideal functionality we need to also give a guarantee that the semi-functionality is *functional* (i.e., it can be used by the server to make commitments) and *correct* (i.e., it is binding on the server). These requirements are specified separately as properties that the semi-functionality needs to satisfy. It is all these three requirements together that make up the specification of the ideal commitment functionality. We shall call such a collection of requirements a *monitored functionality*.

IDEAL *world of Monitored Functionality.* The Monitored Functionality is proposed as an IDEAL functionality, which captures all the security properties of a given task. In Figure 1 we show the four Monitored Functionalities used in this work. We point out some of the important features of this new formalism.

In Figure 1, the semi-functionalities are not fully specified, but allows arbitrary interaction between the server and the semi-functionalities. Once a protocol is chosen, the semi-functionality will be specialized to suit that protocol. That is, the semi-functionality will carry out the client's part in the protocol. Note that the view of the server is unchanged if we replace the interaction with the semi-functionality by the protocol with the client. The important thing here is that irrespective of the protocol that is used, these semi-functionalities are designed to capture the secrecy requirements of the respective tasks. For instance, in commitment, the only messages sent to the client are "`commit`" and "$(\texttt{reveal}, b)$." Indeed, in all four semi-functionalities the messages reaching the client and the IDEAL world adversary are exactly those messages that a corresponding IDEAL functionality in the ES/UC model would specify. The name semi-functionality is to emphasize that they provide only the secrecy guarantee, and correctness needs to be ensured separately. But otherwise there is nothing "semi" about them – technically these are full-fledged functionalities in the ES/UC model.

Next, we draw the reader's attention to the way the correctness requirement is specified. For convenience and concreteness, we employ a new notion, called *monitors*. A monitor is a conceptual device used to specify the security requirements of a functionality. If the security requirement is violated we want the monitor to alert the parties by "raising an alarm." Each session of the functionality has its own monitor. A monitor is a (computationally unbounded) function which can inspect the entire system including all parties and functionalities (except any other monitors) and maintain its own internal state. This is in contrast to the PPT functionalities. There is only one way a monitor can affect the system, namely, by raising an alarm.

Securely Realizing a Monitored Functionality in REAL *world.* In the REAL world we would like to have protocols which can replace Monitored Functionalities. That is, if we replace the IDEAL monitored functionality (i.e., the semi-functionality and monitor) by a protocol, no environment should be able to detect the difference (we are allowed also to replace the REAL adversary $\mathcal{A}$, by an IDEAL adversary $\mathcal{S}$). This involves two things: first the protocol should securely realize the semi-functionality (in the conventional sense of [3]). But in addition, it should be able to mimic being monitored. But clearly there are no monitors in the REAL world. So we require that even in the IDEAL world having the monitor should not be detectable to the environment. Note that this is a requirement on the functionality, and not on the protocol. However, it depends on the protocol in that the functionality is fully specified depending on the protocol.

Definition 1 *We say a protocol securely realizes a monitored functionality if*

1. *for every adversary there exists a simulator such that no environment can tell between interacting with the protocol in the* REAL *world (running the protocol) and interacting with the semi-functionality of the monitored functionality in the* IDEAL *world (this is the condition for the UC theorem to hold), and*
2. *there exists a monitor satisfying the specified requirements, such that for any environment and adversary, the probability that the monitor raises an alarm is negligible, even when there are other protocols, functionalities and monitors in the system.*

Note that in the above definition, the first condition is stated for a stand-alone setting, as the UC theorem [3, 19] ensures that it holds in a composed setting also. But the second condition needs to be met for the composed setting, as we do not have a composition theorem for (computationally unbounded) monitors. (i.e., a monitor may behave entirely differently when, in some part of the system, a REAL protocol is replaced by an IDEAL functionality). Hence we need to show the existence of a monitor for the composed setting- i.e., after all REAL protocols have been transformed to IDEAL functionalities or semi-functionalities. Further there may be other monitors in the system. But the monitors are independent of each other and the only way a monitor interferes with the system is by raising an alarm. Hence other monitors can be ignored for analysing the monitor of a particular session.

Expiring Monitors. For concreteness in our analysis, we shall consider monitors $\mathfrak{M}_\tau$ which *expire* after time τ from the start of the protocol. The guarantee given by the monitor holds only till it expires. But for any τ polynomial in the security parameter, we shall show that the monitor raises an alarm with negligible probability. Thus for any τ polynomially large in the security parameter, the guarantee would hold.

Locked States. For our guarantees to be useful, we would often require that the monitor cannot inspect some parts of the system. In $\mathcal{F}_{\widetilde{\text{CSC}}}$, for example, we would like the monitor to record the server's input independent of the client's input. We cannot make such a guarantee if the client has released its input into the system earlier (either explicitly, or by giving out a commitment, even if the commitment is perfectly hiding). However, if the client's input is kept "locked" and unused until after the sever makes its commitment step, then we should provide the above guarantee. We do this in Section 6.2.

2.3 Generalized Environmental Security

We assume that the reader is somewhat familiar with the ES/UC framework [3]. An IDEAL*functionality* is specified to define a task as well as the security requirements on it. A REAL world protocol is said to *securely realize* the IDEAL functionality, if replacing access to the IDEAL functionality by execution of the REAL protocol does not let the adversaries take advantage of this in any environment. That is, for every REAL world adversary $\mathcal{A}$, there is an IDEAL world adversary $\mathcal{S}$, such that no environment will behave differently when in the REAL world (running the protocol), instead of the IDEAL world (with access to the functionality). All the parties, the adversary, the environment and the IDEAL functionalities are probabilistic polynomial time (PPT) machines.

However for most of the interesting cryptographic tasks, there are no protocols which are secure under the ES/UC model, in the standard model [3, 4, 6]. The only protocols for these tasks, which are ES/UC secure require some strong trust assumptions in the model (eg. honest majority, or a trusted common random string) In [19] this difficulty was overcome by altering the definition of security, to obtain a new framework called the generalized Environmental Security (gES) framework.[4] There the IDEAL world adversary is given extra computational power via oracle access to an exponential time machine (referred to as the "Imaginary Angel," or simply Angel). When an imaginary angel Γ is used, the resulting model will be called the Γ-ES model. A protocol for a functionality is said to Γ-ES-realize the functionality against PPT adversaries, if for every PPT adversary $\mathcal{A}$, we can demonstrate a PPT simulator $\mathcal{S}$ with oracle access to Γ.

As in [19], here the information provided by the imaginary angel will be about a hash function. Suitable assumptions of hardness related to this hash function will be made in Section 2.3. The specification of the imaginary angel is given in Section 2.3. Though our assumptions on the hash function and our imaginary angel are similar to those in [19], they are somewhat simpler in our case. In fact, we avoid a strong "non-malleability flavored" assumption. (Correspondingly, however, we restrict ourselves to the client-server model introduced in Section 2.1.)

[19] proves the composition theorem below for the generalized setting, for any imaginary angel, which when queried, returns a (probabilistic) function of the set of corrupted parties and the query. For further details, we refer the reader to [19].

Theorem 1. (Extended Universal Composition Theorem- Informal Statement) [19] *Let $\mathcal{C}$ be a class of real-world adversaries and $\mathcal{F}$ be an ideal functionality. Let ρ be an n-party protocol that Γ-ES-realizes $\mathcal{F}$ against adversaries of class $\mathcal{C}$. Also, suppose π is an n-party protocol in the $\mathcal{F}$-hybrid model which Γ-ES-realizes a functionality $\mathcal{F}$' against adversaries of class $\mathcal{C}$. Then the protocol π^{ρ} (obtained from π by replacing invocations of $\mathcal{F}$ by invocations of the protocol ρ) Γ-ES-realizes $\mathcal{F}$' against adversaries of class $\mathcal{C}$.*

Hash Function As in [19], our computational assumptions have to do with a "hash function." However our assumptions are weaker than those there. We assume a hash function $\mathcal{H} : \{0,1\}^{k_1} \times \{0,1\}^{k_2} \times \{0,1\} \rightarrow \{0,1\}^{\ell}$, with the following properties:

[4] In [19] it was called generalized *Environmental* Security.

A1 (Difficult to find collisions with same prefix): For all PPT circuits M, for a random $r \leftarrow \{0,1\}^{k_1}$, probability that $M(r)$ outputs (x,y) such that $\mathcal{H}(r,x,0) = \mathcal{H}(r,y,1)$ is negligible.

A2 (Collisions and Indistinguishability): For every $r \in \{0,1\}^{k_1}$, there is a distribution $\mathcal{D}_r$ over the set $\{(x,y,z)|\mathcal{H}(r,x,0) = \mathcal{H}(r,y,1) = z\} \neq \phi$, such that

$$\{(x,z)|(x,y,z) \leftarrow \mathcal{D}_r\} \approx \{(x,z)|x \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(r,x,0)\}$$

$$\{(y,z)|(x,y,z) \leftarrow \mathcal{D}_r\} \approx \{(y,z)|y \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(r,y,1)\}$$

Further, given sampling access to $\mathcal{D}_r$ to a distinguisher, these distributions still remain indistinguishable.

The last condition essentially says that the hash function is "equivocable": i.e., for every r it is possible (but computationally infeasible) to give a z such that z can be explained as a hash of 0 ($\mathcal{H}(r,x,0)$ for some x) as well as a hash of 1 ($\mathcal{H}(r,y,1)$ for some y), and both explanations look as if it came from a uniform choice of x or y. Note that for random oracles all these conditions trivially hold (where k_1, k_2, ℓ are polynomially related).

These assumptions suffice for achieving concurrent Zero Knowledge proofs and commit-and-prove functionality. To securely realize "client-server computation," we make one more assumption (a stronger variety of trapdoor permutations) in Section 5.1. All these assumptions were used in [19] as well (where, in fact, the assumptions used are stronger than the ones here).

Imaginary Angel Γ We specify the imaginary angel Γ that we use through out this work. Γ first checks if the server $\mathcal{S}$ is corrupted or not (recall that we do not allow $\mathcal{S}$ to be adaptively corrupted). If it is corrupted Γ functions as a *null-angel*, i.e., it returns $\perp$ on any query. But if $\mathcal{S}$ is not corrupted, then when queried with a string r Γ draws a sample from $\mathcal{D}_r$ described above and returns it. This is very similar to the imagnary angel used in [19], but slightly simpler.

3 Monitored Commitment and Zero Knowledge Proof

The semi-functionalities $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{ZK}}}$ were introduced in [19] where protocols were given for these semi-functionalities. Further, lemmas were proved there which showed binding and soundness properties of these functionalities. Our protocols in this section are very similar to (but slightly simpler than) the corresponding ones in [19]. The proofs are similar too, except that the binding and soundness properties are now proven in terms of the probability with which a monitor raises alarm.

3.1 Monitored Commitment

The monitored functionality for Commitment $\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle$ was described in Figure 1(a), as composed of the Commitment semi-functionality $\mathcal{F}_{\widetilde{\text{COM}}}$ and a monitor to ensure binding. Now we give a protocol which realizes this functionality, in Figure 2.

Commitment Protocol COM

The committing party is the server $\mathcal{S}$, and the receiving party is some client $\mathcal{K}$.

COMMIT PHASE

$\mathcal{K} \rightarrow \mathcal{S} : r \leftarrow \{0,1\}^{k_1}$

$\mathcal{S} \rightarrow \mathcal{K} : c = \mathcal{H}(r, r', b)$ where $r' \leftarrow \{0,1\}^{k_2}$

REVEAL PHASE

$\mathcal{S} \rightarrow \mathcal{K} : (b, r')$

$\mathcal{K}$: if $\mathcal{H}(r, r', b) = c$ then accept b as revealed

Fig. 2. Commitment Protocol COM

(Given the protocol, we can go back to the specification of $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ and complete the semi-functionality specification, by replacing the "Arbitrary protocol" steps with the corresponding steps from the protocol.)

As mentioned earlier, we call the protocol secure if it achieves the semi-functionality and there exists a monitor as specified by the functionality, which will raise an alarm with negligible probability. The following lemmas which assure us of this.

Lemma 1. *For any polynomial (in the security parameter k) τ, under assumption A1, there is a monitor satisfying the requirements specified by $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$, such that the probability of the monitor raising an alarm within time τ is negligible.*

Proof. Firstly, if the adversary does not corrupt the sender, then the monitor reads the committed value from the honest sender's input to the protocol and sets b^* to that value. It is easy to verify that this monitor meets all the requirements and never raises an alarm.

So suppose the adversary corrupts the sender. In this case the imaginary oracle functions as a null-oracle. Thus the entire system of all the parties and the environment is probabilistic polynomial time. (We need not consider other monitors, as explained above.) The monitor $\mathfrak{M}_\tau$ chooses b^* as follows: examine the state of the entire system and determine the probability p_0 of the sender (legally) revealing this commitment as 0 within time τ, and the probability p_1 of the sender revealing it as 1 within that time. Choose $b^* = 0$ if $p_0 \geq p_1$; else choose $b^* = 1$.

We shall demonstrate a (non-uniform) PPT machine M which accepts $r \leftarrow \{0,1\}^k$ and outputs (x, y) such that $\mathcal{H}(r, x, 0) = \mathcal{H}(r, y, 1)$, with a probability polynomially related to the probability of the monitor raising an alarm.

M simulates the system internally, starting at the point the session is initiated. Recall that this session is to be run with access to the IDEAL semi-functionality. But instead, M will play the role of the semi-functionality for this session. It sends the input it received r as the first message to the sender. Then the sender responds with a string c. At this point M makes two copies of the system, and runs them with independent randomness, for time τ each. If the sender eventually reveals the commitment as $(x, 0)$ in one run and as $(y, 1)$ in the other run, then M outputs (x, y).

Define random variable p_0 (respectively, p_1) as the probability that after sending c the sender reveals the commitment as 0 (respectively, 1) within time τ. The probability that the monitor raises an alarm is at most

$$\mathbf{E}[\min\{p_0, p_1\}] \leq \mathbf{E}[\sqrt{p_0 p_1}] \leq \sqrt{\mathbf{E}[p_0 p_1]} = \sqrt{\frac{1}{2}\mathbf{Pr}\left[M \text{ succeeds}\right]}$$

because after forking two copies of the system, M succeeds (i.e., it manages to output (x, y) such that $\mathcal{H}(r, x, 0) = \mathcal{H}(r, y, 1)$) when in one of the runs the event with probability p_0 occurs and in the other the event with probability p_1.

Since the probability that M succeeds is negligible by assumption on $\mathcal{H}$, so is the probability that the monitor $\mathfrak{M}_\tau$ raises an alarm. Clearly this holds for any τ polynomial in the security parameter.

Lemma 2. COM Γ-*ES-realizes* $\mathcal{F}_{\widetilde{\text{COM}}}$ *against static adversaries, under assumption A2.*

Proof (sketch): For every PPT adversary $\mathcal{A}$ we demonstrate a PPT simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish between interacting with the parties and $\mathcal{A}$ in the REAL world, and interacting with the parties and $\mathcal{S}$ in the IDEAL world. We do this in the presence of the imaginary oracle Γ.
Corrupt Server. Note that the semi-functionality is designed such that choosing the simulator $\mathcal{S}$ to be identical to $\mathcal{A}$ (except that it sends the messages to $\mathcal{F}_{\widetilde{\text{COM}}}$ instead of to $\mathcal{K}$) works.
Honest Server. During simulation $\mathcal{S}$ runs $\mathcal{A}$ internally. When $\mathcal{A}$ starts the commitment protocol, $\mathcal{S}$ initiates a session with the IDEAL functionality. When $\mathcal{A}$ sends out the first message in the protocol r, $\mathcal{S}$ forwards this to the oracle Γ and receives $(x, y, z) \leftarrow \mathcal{D}_r$. Then, when $\mathcal{F}_{\widetilde{\text{COM}}}$ gives the `commit` message, $\mathcal{S}$ provides $\mathcal{A}$ with z as the message from $\mathcal{S}$ to $\mathcal{K}$ (whether $\mathcal{K}$ is corrupted or not). Later if $\mathcal{F}_{\widetilde{\text{COM}}}$ gives the message $(\texttt{reveal}, 0)$, $\mathcal{S}$ provides $\mathcal{A}$ with $(0, x)$ as the REAL message, and if $\mathcal{F}_{\widetilde{\text{COM}}}$ gives the message $(\texttt{reveal}, 1)$, $\mathcal{S}$ provides $(1, y)$ to $\mathcal{A}$. Under the assumption on $\mathcal{D}_r$, it can be shown that this is a good simulation. ■

3.2 Monitored Zero Knowledge Proof

In Figure 3 we show a simple protocol ZK in the $\mathcal{F}_{\widetilde{\text{COM}}}$-hybrid model which Γ-ES-realizes $\langle\mathcal{F}_{\widetilde{\text{ZK}}}\rangle$ against static adversaries. The particular relation R used in $\langle\mathcal{F}_{\widetilde{\text{ZK}}}\rangle$ is of Hamiltonicity (ie, given a graph, whether it contains a Hamiltonian cycle or not). The protocol is a simple adaptation of the well-known zero knowledge protocol for this relation. However that protocol (as well as its previous variants) is not known to be secure in a concurrent setting.
Multi-bit Commitment. Multiple bits can be committed to by running independent copies of the protocol in Section 3.1 in parallel. (Better efficiency can be achieved by making suitable assumptions on the hash function $\mathcal{H}$. But in this work we do not address this aspect of efficiency.) For convenience, we denote this collection of sessions of $\mathcal{F}_{\widetilde{\text{COM}}}$ by $\mathcal{F}^*_{\widetilde{\text{COM}}}$. For simplifying the description, we shall use the notation $\mathcal{S} \leftrightarrow \mathcal{F}^*_{\widetilde{\text{COM}}} \rightarrow \mathcal{K}$: COM(M) to denote a step where $\mathcal{S}$ sends a commitment to the bits of M, to $\mathcal{K}$ through

the semi-functionality $\mathcal{F}^*_{\widetilde{\text{COM}}}$, and $\mathcal{S} \leftrightarrow \mathcal{F}^*_{\widetilde{\text{COM}}} \to \mathcal{K} : \textsc{rev}(M')$ will denote a reveal to $M' \subset M$ later using (the same copy of) $\mathcal{F}^*_{\widetilde{\text{COM}}}$.

Note that we are providing the protocol ZK in the $\mathcal{F}^*_{\widetilde{\text{COM}}}$-hybrid model. So in the semi-functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$, the "arbitrary protocol" will involve interaction of $\mathcal{F}_{\widetilde{\text{ZK}}}$ (and $\mathcal{S}$) with $\mathcal{F}^*_{\widetilde{\text{COM}}}$. This simply means that $\mathcal{F}^*_{\widetilde{\text{COM}}}$ is internally run by $\mathcal{F}_{\widetilde{\text{ZK}}}$, when interacting with $\mathcal{S}$.

The prover receives a Hamiltonian cycle $H \subset G$ as witness. First it verifies that the H is indeed a valid Hamiltonian cycle in G. (Else it aborts the protocol.) We use the above notation for commitments and reveals of $n \times n$ matrices. The adjacency matrix of a graph is naturally represented as an $n \times n$ bit-matrix. For convenience we let a permutation ϕ of $[n]$ also to be represented by an $n \times n$ bit-matrix $\varPhi$ defined as $\varPhi_{ij} = 1$ iff $\phi(i) = j$ (else $\varPhi_{ij} = 0$).

The idea is that the prover has to commit to the pair of $n \times n$ matrices (M_1, M_2) where the verifier expects $M_1 = \phi(G)$ and $M_2 = \varPhi$ for some permutation ϕ with the representation $\varPhi$. In response to sending $b = 0$ the verifier expects the prover to reveal all of M_1 and M_2, where as for $b = 1$ it expects the prover to reveal the bits in M_1 corresponding to a Hamiltonian cycle in $\phi(G)$. An edge is represented by an index (i, j) into this matrix. Given a set of edges ζ, we use $M|_\zeta$ to denote the entries in the matrix M given by the edges in ζ.

Lemma 3. *For any polynomial (in the security parameter k) τ, there is a monitor satisfying the requirements specified by $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$, such that the probability of the monitor raising an alarm within time τ is negligible.*

Proof. Our monitor does exactly what the specification requires: it checks if G is Hamiltonian. If not *and* if $\mathcal{F}_{\widetilde{\text{ZK}}}$ sends the message HAMILTONIAN to $\mathcal{K}$, then the monitor raises an alarm.

We shall use the result that $\mathcal{F}_{\widetilde{\text{COM}}}$ has a monitor, to argue that the probability this monitor raises an alarm is negligible. For each of the n parallel sessions, consider the behaviour of the monitors for $\mathcal{F}_{\widetilde{\text{COM}}}$ for the $2n^2$ sessions of $\mathcal{F}_{\widetilde{\text{COM}}}$. These monitors record values b^*_{ij}, $(i, j) \in [n] \times [2n]$ internally.

For convenience, we define the following events: ALARM is the above event that the monitor raises an alarm; BADCOM is the event that some $\mathcal{F}_{\widetilde{\text{COM}}}$ monitor raises an alarm; ALLGOODQUERIES is the event that in each of the n sessions, for the bit selected by $\mathcal{K}$ the bits recorded by commitment monitors define a valid answer (i.e., for $b = 0$ the monitors have recorded $M_1 = \phi(G)$, $M_2 = \varPhi$ and for $b = 1$ the monitors have recorded an M_1 with Hamiltonian cycle. If any pair (M_1, M_2) recorded by the monitors defines a valid answer for both $b = 0$ and $b = 1$, it implies that the graph is Hamiltonian; else we call the pair "bad." Let ALLBADPAIRS be the event that in all the n sessions, bits recorded by the commitment monitors give bad pairs of matrices. Then it is easy to see (as shown in the soundness proof for the corresponding protocol, in [19]) that

$$\mathbf{Pr}\left[\text{ALARM}\right] \leq \mathbf{Pr}\left[\ \text{ALLGOODQUERIES} | \text{ALLBADPAIRS}\right] + \mathbf{Pr}\left[\text{BADCOM}\right].$$

If a pair is bad it can define a valid answer for at most one of the two possible queries. That is, with probability at most $\frac{1}{2}$, $\mathcal{K}$ makes a good query on that pair. So,

$$\mathbf{Pr}\left[\ \text{ALLGOODQUERIES} | \text{ALLBADPAIRS}\right] \leq 2^{-n}$$

```
Zero Knowledge Proof Protocol: ZK

Common input to $\mathcal{S}$ and $\mathcal{K}$: a graph $G$.

REPEAT IN PARALLEL $n$ TIMES:

$\mathcal{S} \leftrightarrow \mathcal{F}^*_{\widetilde{\text{COM}}} \rightarrow \mathcal{K}$ : $\text{COM}(M_1 = \phi(G), M_2 = \Phi)$, where $\phi$ (represented by the matrix $\Phi$)
            is a randomly chosen permutation of $[n]$.
      $\mathcal{K} \rightarrow \mathcal{S} : b \leftarrow \{0,1\}$
      IF $b = 0$
$\mathcal{S} \leftrightarrow \mathcal{F}^*_{\widetilde{\text{COM}}} \rightarrow \mathcal{K}$ : $\text{REV}(M_1, M_2)$
        ELSE
$\mathcal{S} \leftrightarrow \mathcal{F}^*_{\widetilde{\text{COM}}} \rightarrow \mathcal{K}$ : $\text{REV}(M_1|_\zeta)$
            where $\zeta$ corresponds to the edges of the cycle $\phi(H)$
        ENDIF

END REPEAT

        $\mathcal{K}$ : if in all parallel repetitions
            $b = 0 \implies M_2$ represents a permutation $\phi$, and $M_1$ represents a graph
                    such that $\phi(G) = M_1$
            $b = 1 \implies \zeta$ corresponds to the edges of a Hamiltonian cycle,
                    and $\forall (i,j) \in \zeta, M_{1ij} = 1$
            then ACCEPT
```

Fig. 3. Protocol for the Monitored Functionality for ZK Proof

Since $\mathbf{Pr}[\text{BADCOM}]$ is also negligible, we conclude that $\mathbf{Pr}[\text{ALARM}]$ is negligible.

Lemma 4. ZK Γ*-ES-realizes* $\mathcal{F}_{\widetilde{\text{ZK}}}$ *against static adversaries in the* $\mathcal{F}_{\widetilde{\text{COM}}}$*-hybrid model.*

Proof. Corrupt Server. Just as in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$, if the server is corrupt, a trivial simulator in the IDEAL world, which acts transparently between an internal copy of $\mathcal{A}$ and the semi-functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ perfectly simulates the protocol between the corrupt server and an honest client.
Server not Corrupt. Recall that the protocol is in the $\mathcal{F}_{\widetilde{\text{COM}}}$-hybrid model. If the server is not corrupt, the only protocol messages that $\mathcal{A}$ can see are the statement to be proven, the length of the messages from $\mathcal{S}$ to $\mathcal{F}_{\widetilde{\text{COM}}}$, the `commit` messages from $\mathcal{F}_{\widetilde{\text{COM}}}$, the bit b sent by $\mathcal{K}$ ($\mathcal{K}$ may be corrupt or honest), and the final `proven` message. All these are available to simulator $\mathcal{S}$ in the IDEAL execution too. Note that if $\mathcal{K}$ is not corrupted, the bit b can be chosen uniformly at random during simulation. On the other hand, if $\mathcal{K}$ is corrupted (before it sends out b), then this bit is indeed produced by the copy of $\mathcal{A}$ that $\mathcal{S}$ runs internally. Then it is easily verified that $\mathcal{S}$ can indeed simulate in this case *perfectly*.

The above two lemmas can be summarized as follows.

Lemma 5. *Protocol* ZK *in* $\mathcal{F}_{\widetilde{\text{COM}}}$*-hybrid model* Γ*-ES-realizes* $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$ *against static adversaries.*

Using the composition theorem Theorem 1 and Lemma 2 we get a protocol ZK^{COM} in the REAL world which Γ-ES-realizes $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$ against static adversaries. Note that ZK^{COM} is a 4-round protocol. In the language of Zero-Knowledge proofs, we can state this result as follows.

Theorem 2. *There is a 4-round concurrent Zero Knowledge argument for Hamiltonicity when the simulator (as well as corrupt verifiers) has sampling access to* $\mathcal{D}_r$ *for all* $r \in \{0,1\}^{k_1}$.

4 Monitored Commit and Prove

Somewhat surprisingly, our model of security allows very simple protocols for the commit and prove (monitored) functionality (Figure 1(c)) as well. Below is the semi-functionality $\mathcal{F}_{\widetilde{\text{CAP}}}$, with respect to a relation R, and monitor for it.

For the commit phase, we use a straight-forward extension of the bit commitment protocol COM to multiple bits (see Section 3.2). A transcript of the commitment phase consists of two messages (r, c), where $c = \overline{\mathcal{H}}(r, r', w)$, where r' is a random string privately chosen by $\mathcal{S}$ and w is the string committed to. $\overline{\mathcal{H}} : \{0,1\}^{k_1 t} \times \{0,1\}^{k_2 t} \times \{0,1\}^t \to \{0,1\}^{\ell t}$ is a multi-bit version of $\mathcal{H}$: $\overline{\mathcal{H}}((r_1 \cdots r_t), (r'_1 \cdots r'_t), (w_1 \cdots w_t)) = (\mathcal{H}(r_1, r'_1, w_1), \cdots, \mathcal{H}(r_t, r'_t, w_t))$.

We introduce some more notation to conveniently describe the protocol. Let $\mathcal{S} \leftrightarrow \mathcal{F}_{\widetilde{\text{ZK}}} \rightarrow \mathcal{K} : \text{ZKP}_R(x; r, c)$ denote the following specification: first, parties $\mathcal{S}$ and $\mathcal{K}$ reduce the problem "$\exists w, y, r'$ such that $R'(w, x, r, r', c, y) = 1$" to a Hamiltonicity problem instance $G(x, r, c)$, where $R'(w, x, r, r', c, y) = 1$ if and only if $R(w, x, y) = 1$ and $\overline{\mathcal{H}}(r, r', w) = c$. This reduction is carried out in such a way that given a Hamiltonian cycle in G, it is possible to recover (w, r', y) as above. Then $\mathcal{S}$ uses the semi-functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ to prove to $\mathcal{K}$ that G is Hamiltonian.

The protocol is given in Figure 4. We shall prove the following:

Lemma 6. *Protocol* CAP Γ*-ES-realizes monitored functionality* $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$ *against static adversaries, under assumptions A1 and A2.*

Proof. 1. CAP Γ*-ES-realizes* $\mathcal{F}_{\widetilde{\text{CSC}}}$ *against static adversaries in the* $\mathcal{F}_{\widetilde{\text{ZK}}}$*-hybrid model.*
Corrupt Server. Just as in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{ZK}}}$, a trivial simulator which acts transparently between an internal copy of $\mathcal{A}$ and the semi-functionality $\mathcal{F}_{\widetilde{\text{CAP}}}$ perfectly simulates the protocol between the corrupt server and an honest client.
Server not Corrupt. The protocol is in the $\mathcal{F}_{\widetilde{\text{ZK}}}$-hybrid model, and hence so is the semi-functionality $\mathcal{F}_{\widetilde{\text{CAP}}}$. When the server is not corrupted, the only protocols messages $\mathcal{A}$ can see are the initial commitment messages r and c, lengths of the messages from $\mathcal{S}$ to $\mathcal{F}_{\widetilde{\text{ZK}}}$, and the $(\texttt{proven}, G(x, r, c))$ message from $\mathcal{F}_{\widetilde{\text{ZK}}}$ at the end of each proof phase. The only non-trivial task for the simulator is to produce the commitment text c. Since c will never be revealed (because the server is honest and the adversary cannot adaptively

Commit and Prove Protocol: CAP

COMMIT PHASE:

$$\mathcal{K} \to \mathcal{S} : r \leftarrow \{0,1\}^{k_1}$$

$$\mathcal{S} \to \mathcal{K} : c = \overline{\mathcal{H}}(r, r', w) \text{ where } r' \leftarrow \{0,1\}^{k_2}$$

PROOF PHASE (CAN BE MULTIPLE TIMES):

$$\mathcal{S} \to \mathcal{K} : x$$

$$\mathcal{S} \leftrightarrow \mathcal{F}_{\widetilde{\text{ZK}}} \to \mathcal{K} : \text{ZKP}_R(x; c)$$

$$\mathcal{K} : \text{Accept if accepted in the above protocol}$$

Fig. 4. Protocol for the Monitored Functionality for Commit and Prove

corrupt $\mathcal{S}$), $\mathcal{S}$ can simply use a commitment to a random text using r to produce a purported commitment of w.

Note that assumption A2 implies that the distributions of commitments to 0 and to 1 are indistinguishable (even with access to $\mathcal{D}_r$): that is, for all $r \in \{0,1\}^{k_1}$

$$\{z | x \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(r, x, 0)\} \approx \{z | y \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(r, y, 1)\},$$

because both the distributions are indistinguishable from $\{z | (x, y, z) \leftarrow \mathcal{D}_r\}$. From this, it is a routine exercise to show that no PPT environment (with access to the imaginary oracle Γ) can distinguish between the simulation in the IDEAL world and the execution in the $\mathcal{F}_{\widetilde{\text{ZK}}}$-hybrid model.

2. For any polynomial (in the security parameter k) τ, there is a monitor satisfying the requirements specified by $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$, such that the probability of the monitor raising an alarm within time τ is negligible.

We restrict ourselves to the case when $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$ allows only one proof phase per session. It is possible to extend it to multiple proofs, but the details become lengthy and tedious.

First we describe how a value w^* is recorded by $\mathfrak{M}_\tau$. Consider an "extractor" PPT machine M_τ which simulates the entire (composed IDEAL) system internally, starting at the point where the session of interest running our Commit-and-Prove protocol starts (this start state is given to M_τ as non-uniform advice), for at most τ time-steps. M_τ runs the system until the proof phase of started, and the prover makes the commitment step. At this point M_τ clones the system and runs the two copies independent of each other. If in both the copies the proof is accepted by the verifier, M_τ checks if the n-bit queries made by the verifier in $\text{ZKP}_R(x; r, c)$ are identical or not. If they are not identical this lets M_τ extract a Hamiltonian cycle for G (assuming the monitors for the $\mathcal{F}_{\widetilde{\text{COM}}}$s do not raise any alarm). Then M_τ derives a witness (w, r', y) from this Hamiltonian cycle, and outputs it. Else M_τ outputs $\perp$.

Now we use M_τ to describe the monitor $\mathfrak{M}_\tau$. When $\mathcal{F}_{\widetilde{\text{CAP}}}$ sends `commit` to $\mathcal{K}$, for each w $\mathfrak{M}_\tau$ checks the probability of M_τ outputting w, and records the one with the highest such probability, say w^*. Later if $\mathcal{F}_{\widetilde{\text{CAP}}}$ sends $(\texttt{proven}, x)$ for some x such that for no y $R(w^*, x, y)$ holds, then it raises an alarm. Also, for purposes of analysis, when the prover executes the commitment protocol (semi-functionality) as part of the zero-knowledge proof protocols, $\mathfrak{M}_\tau$ starts the monitors for $\mathcal{F}_{\widetilde{\text{COM}}}$ as sub-monitors. The monitors will also be run when the extractor M_τ runs. If any of these sub-monitors raises an alarm, then too $\mathfrak{M}_\tau$ will raise an alarm.

Clearly $\mathfrak{M}_\tau$ satisfies the requirements of the functionality (up to the time bound τ). We go on to prove that the probability that $\mathfrak{M}_\tau$ raises an alarm (which event we denote by ALARM) is negligible. In the rest of the proof, we condition on the event that none of these sub-monitors raise an alarm. Since we have already shown that this is an event of negligible probability (and only polynomially many such sub-monitors are run), this will not change our conclusions.

Now, consider the point at which M_τ forks the system. Let p_w be the probability that M_τ outputs w starting at (conditioned on) this point, within τ time-steps. Let q be the probability that $\mathcal{K}$ accepts the proof $\text{ZKP}_R(x; c)$ within τ time-steps, but $\nexists (y, r') R'(w^*, x, r, r', c, y) = 1$. Note that $\mathbf{Pr}[\text{ALARM}] = \mathbf{E}[q]$, where the expectation is over the distribution on the state of the system at the point at which M_τ forks.

Since we assume that the sub-monitors do not raise alarm, M_τ outputs some w if the two copies it runs both accept the proof, and in the second copy the verifier sends a query different from the one in the first copy. So, $\sum_{w \neq w^*} p_w \geq q(q - 2^{-n})$. Then,

$$\begin{aligned}
\mathbf{Pr}[M_\tau \text{ outputs } w \neq w^*] &\geq \mathbf{E}[q(q - 2^{-n}] \\
&\geq \mathbf{E}[q]^2 - 2^{-n}\mathbf{E}[q] \\
&\geq \frac{1}{2}\mathbf{E}[q]^2 && \text{if } \mathbf{E}[q] \geq 2^{-n+1}
\end{aligned}$$

If the assumption in the last line above does not hold, we would be done, because $\mathbf{Pr}[\text{ALARM}] = \mathbf{E}[q]$. So we make that assumption and proceed.

Now we shall demonstrate a (non-uniform) PPT machine M'_τ which accepts $r \leftarrow \{0,1\}^k$ and outputs (x, y) such that $\mathcal{H}(r, x, 0) = \mathcal{H}(r, y, 1)$, with a probability polynomially related to the probability of the monitor raising an alarm. $M'_\tau(r)$ starts M_τ and runs the commit phase by sending r. It forks M_τ after the commitment from P arrives. Then it runs the two independent copies of M_τ (which involves forking the system again), and checks if they output different values (w_1, r'_1) and (w_2, r'_2), with $w_1 \neq w_2$. If so, M'_τ derives a collision to the hash function from some bit at which w_1 and w_2 differ, and outputs the corresponding portions of r'_1, r'_2. We say that M'_τ succeeds if it gets (w_1, w_2), such that $w_1 \neq w_2$ from two runs of M_τ. Then,

$$\begin{aligned}
\mathbf{Pr}[M'_\tau \text{ succeeds}] &\geq \sum_{w'} p_{w'} \sum_{w \neq w'} p_w \\
&\geq \sum_{w'} p_{w'} \sum_{w \neq w^*} p_w && \text{because for all } w', p_{w^*} \geq p_{w'} \\
&= (\sum_{w'} p_{w'})(\sum_{w \neq w^*} p_w) \geq (\sum_{w \neq w^*} p_w)^2 \\
&\geq \frac{1}{4}\mathbf{E}[q]^4 = \frac{1}{4}\mathbf{Pr}[\text{ALARM}]^4
\end{aligned}$$

Putting it all together we have that $\mathbf{Pr}\left[\text{ALARM}\right] \leq (4\,\mathbf{Pr}\left[M'_\tau \text{ finds a collision}\right])^{\frac{1}{4}}$, which is negligible by assumption on $\mathcal{H}$.

5 Applications of the New Framework

As we have shown above, theoretically interesting cryptographic tools like commitment and zero-knowledge proofs can be securely realized in the new framework, relatively efficiently (compared to those in previous Environmental Security models). The reason for this is that our security requirements are much more relaxed. However this raises the question if these weakened versions of the above tools are useful to achieve security for practically interesting tasks. In this section we make some progress towards making the new framework usable for multi-party computation problems. We restrict ourselves to 2-party computations of a very specific kind, as described below.

5.1 Client-Server Computation

A 2-party Client-Server Computation functionality $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$ is given earlier in Figure 1(d). Note that the client does not keep any secrets from the server $\mathcal{S}$. But the server must *commit* to its inputs (and the monitor shall record the committed input) before the client sends its inputs. First, we shall give a protocol for this Monitored functionality, before discussing some of its limitations.

Secret Commit and Prove In order to give a protocol for $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$, we need to modify the Commit-and-Prove functionality, so that if both the server and the client are honest, the adversary is not given the statements that the server proves. (This is because the adversary should not learn the client's input.) Such a functionality $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}$ can be securely realized in the $\mathcal{F}_{\text{ENC}}$-hybrid model, where $\mathcal{F}_{\text{ENC}}$ is the encryption functionality. For this $\mathcal{F}_{\widetilde{\text{ZK}}}$ and $\mathcal{F}_{\widetilde{\text{COM}}}$ are modified to the SECRET versions, which do not send the statement proven ($\mathcal{F}_{\text{SECRET-}\widetilde{\text{ZK}}}$) or the bit revealed ($\mathcal{F}_{\text{SECRET-}\widetilde{\text{COM}}}$) to the adversary. $\mathcal{F}_{\text{SECRET-}\widetilde{\text{COM}}}$ can be securely realized by the protocol COM modified to encrypt the reveal step, using the functionality $\mathcal{F}_{\text{ENC}}$. In the static case $\mathcal{F}_{\text{ENC}}$ is known to be easy to implement, using CCA2-secure public-key encryption with new keys each time (see for instance [3]), which in turn can be implemented assuming a family of trapdoor permutations (using the construction in [21], for instance). But since we are in the Γ-ES-model, we need to revisit the assumptions used to securely realize $\mathcal{F}_{\text{ENC}}$, namely the existence of trapdoor permutations.

A3 There exists a family of trapdoor permutations secure against *non-uniform PPT adversaries which are given sampling access to $\mathcal{D}_r$ for all r.*

This is also an assumption made in [19]. With this assumption in place, we get the following result.

Lemma 7. *There are protocols which Γ-ES-realize $\mathcal{F}_{\text{ENC}}$ and $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}$ against static adversaries, under assumptions A1, A2 and A3.*

Client-Server Computation Protocol: CSC

The protocol is parametrized by a function F.

$$\mathcal{S} \leftrightarrow \mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}} \rightarrow \mathcal{K} : \text{COMMIT-PHASE}(x_{\mathcal{S}})$$
$$\mathcal{K} \rightarrow \mathcal{F}_{\text{ENC}} \rightarrow \mathcal{S} : x_{\mathcal{K}}$$
$$\mathcal{S} \rightarrow \mathcal{F}_{\text{ENC}} \rightarrow \mathcal{K} : z = F(x_{\mathcal{S}}, x_{\mathcal{K}})$$
$$\mathcal{S} \leftrightarrow \mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}} \rightarrow \mathcal{K} : \text{PROOF-PHASE}(z == F(x_{\mathcal{S}}, x_{\mathcal{K}}))$$

Fig. 5. Protocol for the Monitored Functionality for Client-Server Computation

5.2 The Protocol

Theorem 3. *The protocol* CSC Γ*-ES-realizes the monitored functionality* $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$ *against static adversaries in the* $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}, \mathcal{F}_{\text{ENC}}$*-hybrid model.*

Proof. 1. For any polynomial (in the security parameter k) τ, there is a monitor satisfying the requirements specified by $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$, such that the probability of the monitor raising an alarm within time τ is negligible.

We can build a monitor $\mathfrak{M}_\tau$ for $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$ using the monitor for $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$. $\mathfrak{M}_\tau$ starts the monitor for $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$, and if the protocol proceeds beyond the first step, it would record a value x^* internally as the committed value. $\mathfrak{M}_\tau$ will copy that value and record it as the input of $\mathcal{S}$. Later if the monitor for $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$ raises an alarm, $\mathfrak{M}_\tau$ will raise an alarm. If $\mathcal{F}_{\widetilde{\text{CSC}}}$ sends the value z to $\mathcal{K}$, then $\mathcal{F}_{\widetilde{\text{CAP}}}$ must return $(\texttt{proven}, z = F(x^*, x_{\mathcal{K}}))$. So if the monitor for $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$ does not raise an alarm, it means indeed $z = F(x^*, x_{\mathcal{K}})$ and $\mathfrak{M}_\tau$ need not raise any alarm either. Thus $\mathfrak{M}_\tau$ does satisfy the reuirements specified by $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$. Further the probability that $\mathfrak{M}_\tau$ raises an alarm is the same as that the monitor for $\langle \mathcal{F}_{\widetilde{\text{CAP}}} \rangle$ raises an alarm. By earlier analysis, this is indeed negligible.

2. CSC Γ-ES-realizes $\mathcal{F}_{\widetilde{\text{CSC}}}$ against static adversaries in the $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}, \mathcal{F}_{\text{ENC}}$-hybrid model.

For every PPT adversary $\mathcal{A}$ we demonstrate a PPT simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish between interacting with the parties and $\mathcal{A}$ in the REAL world, and interacting with the parties and $\mathcal{S}$ in the IDEAL world.

As usual $\mathcal{S}$ internally runs $\mathcal{A}$ (which expects to work in the $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}, \mathcal{F}_{\text{ENC}}$-hybrid with the parties running the CSC protocol), and works as an interface between $\mathcal{A}$ and the parties. When $\mathcal{A}$ starts the CSC protocol, $\mathcal{S}$ initiates a session with the IDEAL functionality $\mathcal{F}_{\widetilde{\text{CSC}}}$.

Corrupt Server. Again, as in the case of all the monitored functionalities introduced in this work, thanks to the way the semi-functionality is designed, a trivial simulator in the IDEAL world, which acts transparently between an internal copy of $\mathcal{A}$ and the semi-functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ perfectly simulates the protocol.

Server not Corrupt. The client $\mathcal{K}$ may or may not be corrupt. We analyse the two cases separately:

- *Honest $\mathcal{K}$:* In this case all that $\mathcal{A}$ can see are the lengths of the messages $x_{\mathcal{S}}, x_{\mathcal{K}}$ and $F(x_{\mathcal{S}}, x_{\mathcal{K}})$, given to it by $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}$ and $\mathcal{F}_{\text{ENC}}$. These are known to $\mathcal{S}$ (because F is publicly known), and it can send them to $\mathcal{A}$.

- *Corrupt $\mathcal{K}$:* In this case $\mathscr{S}$ gets $x_{\mathcal{K}}$ and $F(x_{\mathcal{S}}, x_{\mathcal{K}})$. In addition $\mathcal{A}$ expects to see the messages `commit` and `proven` from $\mathcal{F}_{\text{SECRET-}\widetilde{\text{CAP}}}$ (in the first and last steps of the protocol). These are easily provided by the simulator.

It is easy to see that in all the cases, the simulation is perfect.

From this theorem, using Lemma 7 and the composition theorem Theorem 1, we get the following corollary.

Corollary 4 *There is a protocol which Γ-ES-realizes monitored functionality $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$ against static adversaries, under assumptions A1, A2 and A3.*

5.3 Extensions to Adaptive Adversaries

Above we analyzed security in the presence of static adversaries, for the sake of simplicity. Here we mention how the tools developed here can be extended to the case of adaptive adversaries. Firstly, if we expand the adversary class to allow the adaptive corruption of only the clients, it is easy to see that the analyses still hold. The only modification required is that (in the case of the "SECRET" versions of the functionalities), the encryption protocols used will need to be secure against adaptive adversaries as well.

However extending to full-fledged adaptive corruption (i.e., adaptive corruption of the server as well) requires more modifications. Note that the Imaginary Angel Γ functions as a null-angel when the server $\mathcal{S}$ is corrupted, but otherwise gives access to the distribution $\mathcal{D}_r$. If $\mathcal{S}$ is initially uncorrupted and corrupted later on, removing access to $\mathcal{D}_r$ is not enough; having had access to $\mathcal{D}_r$ in the past gives the adversary an advantage. To fix this, we can use the hash function used in [19] for commitment, which takes one more parameter, namely, the ID of the receiving party. The assumptions used and the Imaginary Angel will then be the same as in [19]. The difference with [19] is that the "basic commitment" and "basic ZK proof" protocols there cannot be directly used to satisfy the security requirements there, where as the final protocols developed are secure only against static adversaries. In our case these basic protocols can be directly used to securely realize monitored functionalities. Note however that there is no significant advantage in using the client-server model anymore if we use the same assumptions as in [19]. Indeed, the resulting protocols securely realize the monitored functionalities in the unrestricted environmental setting (without the restrictions of the client-server model), against adaptive adversaries.

6 Limitations and Challenges

6.1 Problem with $\langle \mathcal{F}_{\widetilde{\text{CSC}}} \rangle$

Though we have successfully applied our tools in the new framework to obtain a 2-party computation protocol, there are some serious limitations to this functionality. Clearly, the set of functions that are computed are limited (namely, only client-server computations). But more seriously, the guarantee given by the monitor is not satisfactory.

In particular, there is no guarantee of *"independence" of inputs*. Though the monitor records a value for the server's input prior to the client sending out its input, the value recorded is allowed to be *dependent on the entire system*, and in particular on the input of the client![5]

6.2 The Solution: Restricting the Monitors

In ongoing work, we suggest ways to address this problem. There we show that if the clients keep their private inputs totally unused until the point of commitment (but may use them immediately afterwards), then the monitor can be required to record a value independent of their private inputs. As it turns out, the protocols are not altered, but some restrictions are imposed on the monitor, and some parts of the proof become significantly more involved.

We allow that some part of the state of the system can be kept "locked." This part, which we shall call the *locked state*, cannot be used in the system (until it is unlocked). The requirement on the monitor is that it does not have access to the part of the system state if it is locked at the point the monitor is required to record a value; it will have to record a value based on the rest of the system, which we shall call the *open state*.

Technically, the locked state corresponding to a protocol execution is defined at the beginning of that execution: it is the maximal part of the system state, not including any of the adversary's state, such that the *distribution* of the rest of the system state at the recording point is *independent* of it. Note that the independence requirement implies in particular that the probability of unlocking the state before the monitor finishes recording, is zero (unless the locked state is completely predictable *a priori* from the open state).

We do allow the locked state to evolve, as long as the independence is maintained (in particular, no information should pass between the locked state and the open state). Further, for full generality, we allow the locked state to be randomized: i.e., its value is a random variable. However, we shall require that this random variable is efficiently sampleable (which is implied by the assumption that the non-adverserial part of the system is PPT). In particular all the "future" randomness, i.e., randomness which is sampled after the monitor finishes recording, can be considered part of the locked state.[6]

As indicated earlier, the reason we allow the notion of a locked state in our framework has to do with the meaningfulness of the two-party computation scenario. With the modification sketched above in place, we can allow the client to keep its input locked, and then even the monitor does not get to see it, before recording the other party's input.

[5] The monitor's recorded value is independent of as yet unsampled randomness in the system. So if the client's input is only a freshly sampled random value, as is the case in a ZK proof or coin-tossing protocol, this issue does not arise.

[6] Incidentally, in the use of semi-functionalities in [19], the only locked state is future randomness. However this is an especially simple special case, taken care of by the original proof there. [19] does not introduce or require a generalization. As it turns out generalizing to other locked states complicates our arguments considerably.

However, note that to keep an input locked, it can never be used in the system at all (until it is unlocked).[7] This is because the monitor is computationally unbounded. Note that this is related to the problem of malleability: if it was used in the system previously, somehow that can be mauled and used to make a commitment related to it. (It is an interesting problem to relax this information theoretic locking constraint to a computational equivalent.) However, interestingly we do avoid the problem of malleability while *opening* a commitment: the locked state is allowed to be unlocked *before the commitment is opened.* Indeed, if the locked states are to be kept locked until after the protocol terminates completely, restricting the monitor to the rest of the system state is automatic. But to be useful, we need to allow locked states which can be opened after the monitor records its value, but before the protocol terminates.

In work under progress we show how to prove that in all the monitored functionalities we use, the monitors can be required not to inspect the locked state of the system. Surprisingly, this complicates the construction of the monitor and the proofs considerably. Below we sketch the changes in the proof in the case of $\mathcal{F}_{\widetilde{\mathrm{COM}}}$.

Lemma 8. *For any polynomials (in the security parameter k) τ and Π, under assumption A1, there is a monitor satisfying the requirements specified by $\langle \mathcal{F}_{\widetilde{\mathrm{COM}}} \rangle$ which does not inspect the locked state of the system, such that the probability of the monitor raising an alarm within time τ is less than $1/\Pi$.*

Proof (sketch): When $\mathcal{F}_{\widetilde{\mathrm{COM}}}$ sends the `commit` message the monitor $\mathfrak{M}_{\tau,\Pi}$ must record a bit b^* internally. First, we sketch how $\mathfrak{M}_{\tau,\Pi}$ does this. As before, the basic idea is for the monitor to look ahead in the system, and record the more likely bit that the sender will ever reveal; if the sender can reveal to both bits with significant probability, a reduction can be used to obtain a circuit for finding collisions in the hash function. But note that here $\mathfrak{M}_{\tau,\Pi}$ does not know the value of the locked state, and so it cannot calculate the bit as above. However, we can show that for no two values for the locked state, can the sender feasibly reveal the commitment in different ways. Intuitively then, the monitor can use an arbitrary value for the locked state and use it to carry out the calculation. However, there are a couple of problems with this. Firstly, revealing can depend not only on the open state of the system at the end of commitment, but also on the locked state, as it might be unlocked after the commit phase is over. In particular, for certain values of the locked state (and open state), the sender might never complete the reveal phase. So using a single value of the locked state will not suffice. The second problem is that while $\mathfrak{M}_{\tau,\Pi}$ is computationally unbounded, the reduction to finding collision should use a polynomial sized circuit. This circuit will need to be given the value(s) of the locked state with which it will emulate the system. Further, the circuit will obtain as input the random challenge in the commitment. Thus, the value(s) of the locked state that it obtains should be defined prior to seeing the random challenge.

[7] In other words, the inputs are for *one time* use only. After that if it is used as a client input in a server-client computation protocol, there is no guarantee that the server's input will be independent of that input. This is a significant limitation. However note that a client's input for a "server-client" computation, with a corrupt server is the *last time* it can be used secretly, as the computation gives the client's input to the server.

Nevertheless, we show how to define *polynomially many values for the locked state of the system, based only on the open state of the system*, and obtain a bit b^* using just these values. To show that the probability of $\mathfrak{M}_{\tau,\Pi}$ raising an alarm within time τ is less than $1/\Pi$, we show that otherwise we can give a polynomial sized circuit (with the above mentioned values of the locked states builtin) which can find a collision in our hash function for a random challenge with significant possibility. ■

The construction of the monitor for $\mathcal{F}_{\widetilde{\text{CAP}}}$ is also changed in a similar fashion. However, since the monitor in this case is defined based on an extractor, and the extractor itself will need to be modified to take polynomially many values of the locked state, the proof is much more involved. The monitors for the $\mathcal{F}_{\widetilde{\text{ZK}}}$ and $\mathcal{F}_{\widetilde{\text{CSC}}}$ need to be modified too. However since their description and proof is based on those of $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{CAP}}}$ respectively they do not involve much change.

7 Conclusion

We introduced a framework of Monitored Functionalities, which provides a way to define and prove relaxed (but ES) security guarantees for (relatively simple) protocols. We also introduced a restricted model called the Client-Server model, which allows simpler protocols to be secure, and potentially under simpler computational assumptions. Both these relaxations, we believe, would help in further exploring Environmental Security (Network-Aware Security).

However, the applicability of the security guarantees from this work are somewhat limited. It is an open problem to work around these limitations, while still maintaining the relaxed nature of the security requirement so that simple protocols are possible. We suggest restricting the computational powers of the monitors (but still giving them more power than the players) as a useful direction.

There are many other ways in which this line of research can be furthered. It is a challenge to try and base these results on more conventional computational assumptions, without setups. On the other hand it should be relatively simpler to allow setups and replace the use of gES model here, by the ES/UC model. A general direction to pursue is to use Monitored Functionalities or other similar notions to give some security guarantee to many simple, efficient and intuitively secure protocols currently used in practice.

References

1. B. Barak. How to Go Beyond the Black-Box Simulation Barrier. FOCS 2001: 106-115.
2. Boaz Barak. Constant-Round Coin-Tossing with a Man in the Middle or Realizing the Shared Random String Model. FOCS 2002: 345-355.
3. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. FOCS 2001: 136-145.
4. R. Canetti and M. Fischlin. Universally composable commitments. Crypto 2001: 19-40.
5. Ran Canetti and Hugo Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. EuroCrypt 2002: 337-351.
6. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. EuroCrypt 2003: 68-86.

7. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. STOC 2002: 494-503.
8. Danny Dolev, Cynthia Dwork, Moni Naor. Nonmalleable Cryptography. SIAM J. Comput. 30(2) 2000: 391-437.
9. Cynthia Dwork, Moni Naor, Amit Sahai. Concurrent Zero-Knowledge. STOC 1998: 409-418.
10. S. Goldwasser, Y. Lindell. Secure Computation without Agreement. DISC 2002: 17-32.
11. Joe Kilian, Erez Petrank. Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. STOC 2001: 560-569.
12. Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. STOC 2003: 683-692.
13. Tatsuaki Okamoto. An Extension of Zero-Knowledge Proofs and Its Applications. AsiaCrypt 1991: 368-381.
14. Rafael Pass. Simulation in Quasi-Polynomial Time, and Its Application to Protocol Composition. EuroCrypt 2003: 160-176.
15. Rafael Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. STOC 2004: 232-241.
16. Rafael Pass, Alon Rosen. Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. FOCS 2003: 404-413.
17. Birgit Pfitzmann, Michael Waidner. Composition and integrity preservation of secure reactive systems. ACM Conference on Computer and Communications Security 2000: 245-254.
18. Manoj Prabhakaran, Alon Rosen, Amit Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. FOCS 2002: 366-375.
19. Manoj Prabhakaran, Amit Sahai. New Notions of Security: Achieving Universal Composability without Trusted Setup. STOC 2004: 242-251.
20. Ransom Richardson, Joe Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. EuroCrypt 1999: 415-431.
21. Amit Sahai. Non-malleable Non-interactive Zero Knowledge and Adaptive Chosen Ciphertext Security. FOCS 1999: 543-553.