Polynomial Runtime in Simulatability Definitions

Dennis Hofheinz[1], Jörn Müller-Quade[2], and Dominique Unruh[3]

[1] CWI, Amsterdam, Dennis.Hofheinz@cwi.nl

[2] IAKS, Universität Karlsruhe, muellerq@ira.uka.de

[3] IS&C, Saarland University, unruh@cs.uni-sb.de

Abstract. We elaborate on the problem of polynomial runtime in simulatability definitions for multi-party computation. First, the need for a new definition is demonstrated by showing which problems occur with common definitions of polynomial runtime. Then, we give a definition which captures in an intuitive manner what it means for a protocol or an adversary to have polynomial runtime.

We show that this notion is suitable for simulatability definitions for multi-party computation. In particular, a composition theorem is shown for this notion.

Keywords: multi-party computation, reactive simulatability, universal composability.

1 Introduction

1.1 Overview

It is a non-trivial task to define the security of a multi-party protocol. Even when one restricts to protocols for a very specific application, say, tossing a coin over a telephone line, it may not be obvious what technically the protocol goals are. The situation gets worse as soon as one tries to design a larger protocol, possibly composed of different smaller protocols with possibly colliding or incompatible security requirements.

[*] This is the full version of [19].

To cope with this problem, so-called simulation-based security definitions have been introduced. These security definitions define the security of a protocol relative to an ideal specification of the protocol task. This gives a simple and unified method for defining the security requirements of a multitude of different protocol tasks. Modern simulation-based frameworks (e.g., the Reactive Simulatability framework [24, 7] and the Universal Composability framework [8, 12]) are further equipped with a powerful composition theorem. These composition theorems guarantee that different protocols can be *composed* (i.e., run together) *without losing their security*.

Since in most cases we are interested in *computational* security, i.e., security that holds only with respect to computationally feasible attacks, a security definition should come with a notion of what a feasible attack (and a feasible protocol, for that matter) is. Usually, cryptographers consider an attack or protocol to be feasible if it runs in polynomial time. However, the notion of running in polynomial time (and therefore of feasible protocols and attacks) in such a simulatability-based setting turned out to be nontrivial.

Straightforward approaches lead to technical artifacts, limitations on the classes of protocols that may be investigated, or even to contradictory security models. (We will exemplify this below when the idea of simulation-based security has been covered in more detail.)

This problem has been recognized, but, as we will argue, not solved in a satisfying way. The goal of this work is thus to develop a notion of polynomial runtime that is appropriate for simulatability frameworks. We will propose a suitable notion, and demonstrate its usefulness by discussing a simple example that cannot be modeled with other approaches in a meaningful way. We also give a general composition theorem that shows that the security notion induced by our notion of polynomial runtime still enjoys the nice composability properties of a modern simulation-based framework.

Our argumentation will take place in the simulatability model of *Reactive Simulatability* (RSIM, cf. [24, 7]) of Backes, Pfitzmann, and Waidner. However, our ideas do not rely on specific properties of this framework. Hence, we also sketch how to apply our ideas to the *Universal Composability* simulatability model (UC, cf. [8]) of Canetti.

We now give more details on the setting and our approach.

1.2 The general idea of simulatability

In a nutshell, a simulation-based notion of security compares a real protocol to an ideal specification. That ideal specification is modelled by an ideal protocol or a so-called ideal functionality. This ideal protocol is secure by assumption. Then the simulation-based notion guarantees that if any attack that is possible on the real protocol is also possible on the ideal protocol. Since by assumption no harmful attack is possible on the ideal protocol, it follows that all attacks on the real protocol are harmless, too.

More concretely, in the Reactive Simulatability framework and the Universal Composability framework, this idea is modelled as follows: We require that for each adversary A_1 that attacks the real protocol, there is a corresponding ideal adversary or simulator A_2 that attacks the ideal protocol. We then introduce a third entity, the honest user[4] that communicates both with the protocol and with the adversary. If then no honest user can distinguish between running with the real protocol and the real adversary and running with the ideal protocol and the ideal adversary, we call the real protocol as secure as the ideal protocol. (More details on this modelling in the case of Reactive Simulatability are given in Section 2.)

However, when filling in the details of this definition, one question that arises is how to model feasible, i.e., polynomial-time adversaries/honest users and protocols. Modelling feasible adversaries/honest users is indispensable, since we want to quantify only over these when modelling computational security. And a corresponding definition of feasible protocols turns out to be necessary since for unlimited protocols but computationally limited adversaries and honest users we lose, e.g., the compositionality property of the security definition. We will discuss this issue further in the next section.

1.3 The traditional notion of feasibility and its problems

In this section, we will explain the various difficulties arising when trying to model polynomial-time protocols and adversaries. To exemplify these difficulties, we will con-

[4] In the framework of [8], this entity is called the (protocol) environment $\mathcal{Z}$.

sider the ideal functionality $\mathcal{F}_{\mathrm{PKE}}$ which is often used to model the security of public-key encryption. This functionality is defined as follows (we simplify somewhat):

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

Key generation: Upon receiving a message $(\texttt{KeyGen})$ from a party P_i, request a value e from the adversary and hand e to P_i.

Encryption: Upon receiving a message $(\texttt{Encrypt}, e', m)$ from a party P_j, hand $|m|$ to the adversary. When the adversary answers with a ciphertext c, pass c to P_j. Further, if $e = e'$, store the pair (c, m).

Decryption: Upon receiving a message $(\texttt{Decrypt}, c)$ from P_i (and P_i only), check whether (c, m) has been stored. If so, return m to P_i. Otherwise, ask the adversary for a value of m and return that value.

A detailed explanation of the workings of this functionality is out of the scope of this section and given in [8]. Note however that the intended implementation (i.e., real protocol) of this functionality is as follows: We assume a public-key encryption scheme to be given. When party P_i of the real protocol gets a message $(\texttt{KeyGen})$ from the honest user, it chooses a public/secret key pair (e, d) using the key generation algorithm and returns the public key e. When P_j gets a message $(\texttt{Encrypt}, e', m)$ it encrypts m with public key e' and returns the resulting ciphertext c. When P_i gets a message $(\texttt{Decrypt}, c)$, it decrypts c using the previously stored secret key d and returns the plaintext m.

We are now ready to examine the various approaches to the problem of modelling polynomial-time in simulation-based security model that are found in the literature.

Most models (e.g., [24] and [8]) bound the computational complexity of all machines[5] that participate in a protocol run (i.e., parties, adversary, and honest user) to strict polynomial-time in the security parameter k. That is, every machine M (this could be a protocol party, an adversary, or the honest user H) halts after running $p_{\mathsf{M}}(k)$ steps for a polynomial p_{M} which depends only on the machine M.

Although this approach may seem very natural, it is problematic for several reasons. First, it does not allow for modelling a protocol task that has no a-priori fixed bound on its running time. For example, consider the case of a public-key encryption scheme. Such a scheme should support to encrypt an arbitrary (though polynomial) number of messages, i.e., the number of messages depends on the application using that encryption scheme. In particular, the concrete number of messages that can be encrypted is not part of the specification of the encryption scheme. This fact is captured by the ideal functionality $\mathcal{F}_{\mathrm{PKE}}$ above.

However, with the above sketched notion of polynomial runtime, the machine $\mathcal{F}_{\mathrm{PKE}}$ must obey a strict polynomial bound on its runtime and hence terminate after a polynomial number of encryptions. This polynomial has to be fixed at specification time of

[5] Depending on the chosen framework, this might be an IO state automaton [24], or an interactive Turing machine [8].

$\mathcal{F}_{\mathrm{PKE}}$. Thus the formulation of $\mathcal{F}_{\mathrm{PKE}}$ as given at the beginning of this section is not a valid machine in this formulation of polynomial runtime.

Instead, it is necessary to use a *family* of functionalities $\mathcal{F}_{\mathrm{PKE,p}}$ parametrised by a polynomial p bounding the number of activations and the length of the messages.[6] However, using such a family does not capture the fact that the encryption scheme used for implementing the functionality does not impose any such bounds. Further, this abstraction makes modular protocol design more complicated, since there is not a single functionality $\mathcal{F}_{\mathrm{PKE}}$ that can be used once and for all, instead one has to choose for each application which functionality $\mathcal{F}_{\mathrm{PKE,p}}$ to take.

However, there is a second, more technical issue with a strict polynomial bound on the runtime of each machine. Namely, recall that in the simulatability definition, and consider the following order of quantifiers: "For every real adversary there is an ideal adversary such that no honest user H can distinguish the real and ideal protocol." (Different orders of quantifiers can be used, see Section 2.) In other words, the complexity of H may depend on the complexity of the real and ideal adversary. This means that H can first of all "outrun" the adversary it runs with by sending the adversary (and only the adversary) useless messages. This forces the adversary to waste its runtime by processing these useless messages from H. Eventually, the adversary will have to terminate, and

[6] The (unparametrised) functionality $\mathcal{F}_{\mathrm{PKE}}$ has actually been used in the literature together with the notion of strict polynomial time, despite the fact that it formally is not a valid machine. This exemplifies the need for a definition that allows to use this functionality.

the only running machines will be H and the protocol (which, up to that point, has not been activated at all). Hence, protocol π and idealization τ must look indistinguishable from the perspective of H *even when the respective adversary has halted*.

This is problematic when the real protocol π requires no adversary to provide mere functionality, but the idealization τ does require the "help" of the ideal adversary to look indistinguishable to π. While this may sound artificial, it is common practice to use this technique to make simulatability at all possible: e.g., this is the case when implementing $\mathcal{F}_{\mathrm{PKE}}$ using a public key encryption scheme. Observe that, e.g., to perform an encryption, the functionality $\mathcal{F}_{\mathrm{PKE}}$ invokes the adversary to get a ciphertext (this represents the fact that we impose no guarantees as how a ciphertext looks like). So in the ideal model running $\mathcal{F}_{\mathrm{PKE}}$, an encryption can only be performed when the adversary has not terminated yet. However, a party in the real protocol will usually encrypt by internally invoking the encryption algorithm. This does not require interaction with the adversary. So we have the surprising situation that formally a real protocol that does perform communication when encrypting messages cannot be a secure implementation of $\mathcal{F}_{\mathrm{PKE}}$, while a slight modification of that real protocol that sends a message to itself before returning the encryption would be secure.

In the special case of the functionality $\mathcal{F}_{\mathrm{PKE}}$ for public-key encryption, a solution was proposed in [18]: keep all machines polynomial *per activation*, and quantify only over honest users H that guarantee a polynomial total running time of the complete protocol run (with both π and τ). However, their definition was especially adapted to the special case of $\mathcal{F}_{\mathrm{PKE}}$ and did not specify a general solution to the problem of modelling computational security of arbitrary protocols.

A similar approach was later used in [11]: here, all machines are polynomial per activation in the maximum of the security parameter and the total length of all messages from the environment; however, environment and adversary are strictly polynomially bounded. In this situation, an environment is no longer able to flood (and thereby disable) the dummy parties with wrong inputs; yet, $\mathcal{Z}$ may still "exhaust" the ideal-model adversary. Furthermore, their model does not allow a party to run in polynomial time in the length of the incoming messages from other parties, in particular, a protocol party that receives a ciphertext, decrypts it, and then outputs it would not be polynomial in the sense of [11].

On December 13, 2005, the paper [12] was updated to contain an alternative approach to solve the problem of polynomial runtime. However, for technical reasons their approach requires that the total length of the messages sent by a party to the functionality is less than the total length of the inputs from the environment to that party. This implies that in protocols have to require special padding of their inputs to be allowed to access the functionality. Furthermore, as with [11], protocol parties are not allowed to run in polynomial time in the length of the message coming from other parties, resulting in the problems sketched in the preceding paragraph.

The original formulation of the RSIM framework [24] uses the approach of requiring all machines to run in strict polynomial time, i.e., to run in $p(k)$ steps where k is the security parameter. At the beginning of this section, we saw that this introduces the problem that a given machine (e.g., the honest user H) may cause other machines (e.g., the adversary) to terminate by sending many messages, resulting in unexpected effects.

This technical problem was addressed in [1, 2, 7] by allowing every machine to "block" selected connections. (To do so, a machine could set its so-called "length function" for that connection to zero.) So for example, the ideal-model adversary A_2 may—from a certain point in time on—block all connections from the user H, when the corresponding real-model adversary would have halted or blocked this connection. Thus, H is not able to "kill" A_2 anymore, while A_2 may still service requests from the functionality.

However, this approach does not solve the issue that the functionality has to be parametrised over the maximum number or length of messages it can handle (i.e., we still have to use a family of functionalities $\mathcal{F}_{\mathrm{PKE,p}}$).

Furthermore, notions like "polynomial fairness" of an adversary (which means that this adversary schedules messages between parties after a polynomial number of activations, cf. [4]) are not compatible with an a priori polynomially bounded adversary. This is due to the fact that the adversary is not able to schedule messages after it has halted, and thus situations may arise where the adversary has to run longer than the a-priori fixed runtime polynomial to fulfil its scheduling guarantees. Examples for this issue are given in [4].

In [16] the problems with polynomial runtime have also been noticed. Their solution consists of introducing so-called guards, a generalisation of length functions. These guards may reject or modify incoming messages without wasting any of the total runtime of the concerned machine. This solves the problem of “killing” a machine by sending nonsensical inputs (these may be removed by the guard), but still requires that the amount of actual work a machine does is a priori bounded. In particular, we would still need a family of functionalities $\mathcal{F}_{\mathrm{PKE,p}}$, since $\mathcal{F}_{\mathrm{PKE}}$ does not have an a-priori bound on the number of messages it encryption and thus on the amount of work invested by $\mathcal{F}_{\mathrm{PKE}}$.

1.4 Our Contributions

Motivated by the discussion above we give a new definition of polynomial runtime for simulatability and prove several desirable properties of our definition. The definition is stated in the model of reactive simulatability, but the concept is model-independent and should carry over to the UC framework.

The honest user H will be chosen to be *weakly polynomial* (cf. [7]), i.e., it will in each activation be polynomially limited in the security parameter and the overall length of all its incoming messages. The adversary (both real and ideal) will be limited in the runtime of H. To guarantee this, two specific connections between the adversary and the user will be used to limit the adversary in the message volume communicated over these lines. Honest users and adversaries limited in this sense will be called *continuously polynomial*.

We stress that this definition allows users and adversaries that do not terminate at all. Specifically, they may run long enough to break every complexity-based cryptographic system. However, the definition guarantees that they may not do so in polynomial prefixes of H's view. In fact, the definition guarantees that in polynomial prefixes of H's view, both A and H take only a polynomial number of steps, *and* both of them send only messages of at most polynomial size to the protocol. This captures a very intuitive notion of polynomial runtime for honest users and adversaries, the intuition being that a computationally secure protocol may well be broken given an unlimited amount of time, but not within a polynomial amount of time. This security notion is presented in Section 3.

Polynomial limitations of a protocol will be captured by the notion of *polynomially shaped* protocols. Roughly, a set of machines is polynomially shaped if the total length of all messages sent by these machines is polynomial in the security parameter k plus the overall length of inputs which machines from this set got from machines outside of this set. If additionally all machines in the set are weakly polynomial (see above) we call this set *polynomially shaped weakly polynomial* (*ps-wp* for short). The notion of ps-wp is a natural definition of a protocol being "polynomially bounded in input length and security parameter” without having to give explicit a priori bounds for the lifetime of machines.

In Section 4, we prove a generalised composition theorem for ps-wp protocols. Specifically, in any ps-wp collection of machines, a functionality may be replaced by a secure implementation if the resulting collection of machines remains ps-wp.

We note that the set of ps-wp protocols is *not* closed under composition (i.e., there are ps-wp protocols which yield a non-ps-wp protocol if composed). We argue that this is not a flaw of our notion, but a “necessary evil” if one wants to catch the intuitive notion of a polynomially bounded protocol. Therefore, we construct an example of two protocols which are “intuitively polynomial” (and ps-wp), but which compose to a protocol that is non-polynomial in every intuitive way.

Additionally, we give a sub-notion of ps-wp protocols that is closed under composition. As a simple consequence, the mentioned ps-wp composition theorem shows that this notion allows for a *secure* composition of protocols (without any additional conditions on the complexity of the composed protocols).

In Section 5, we relate our new notion of security to the existing notion of polynomial security from [7]. More specifically, we prove that our notion is at least as strict as the one from [7].

In Section 7, we sketch how to apply our ideas to the UC framework.

Finally, in Appendix B we show that the generalisation of simulatable security to machines which are intuitively polynomial as defined in this work, but not strictly polynomial, will allow us to omit the formal concept of length functions, which was introduced in [1] to solve problems arising with strictly polynomial functionalities. More specifically, we show that removing length functions from protocol machines does not change the notion of security.

1.5 Interaction with previous results

An example of a published claim which is not formally correct in a setting with strictly polynomial machines regards the ideal functionality $\mathcal{F}_{\mathrm{PKE}}$ for public key cryptography. Claim 15 in [9] states that the protocol $\pi_{\mathcal{S}}$ given there realises the ideal functionality $\mathcal{F}_{\mathrm{PKE}}$ using an IND-CCA secure public key cryptosystem.

The ideal functionality $\mathcal{F}_{\mathrm{PKE}}$ as described above can be invoked an unlimited number of times and it is hence not allowed as a strictly polynomial ideal protocol and the proof of Claim 15 is not formally correct. A bigger problem has already been stated as a motivating example in Section 1.3: An environment machine can exhaust the ideal adversary and distinguish the real and the ideal model as encryption and decryption will still work in the real model, but cannot work in the ideal model as the help of the ideal adverssary is needed. This invalidates Claim 15. However, the proof of Claim 15 seems to carry over to the setting of continuously polynomial security presented here.

The impossibility results for composable protocols remain unchanged by the new notion of continuously polynomial security. The impossibility of bit commitment [13] and other secure computations [14] seem to directly carry over to the notion of continous polynomial security, as the corresponding proofs and attack methodology does not seem to rely on specific complexity bounds.

Similarly, the proofs for the results in [20, 21] (these results investigate the importance of the order of quantification of A_1, A_2 and H) *seem* to carry over to our setting, although they use the notion of runtime in an inherent way. On the other hand, the results from [22] do not obviously carry over to our notion of polynomial runtime.

The notion of continuously polynomial runtime may not only affect proofs for previous results, but the presentation of protocols and functionalites could in future be substantially simplified. The cryptographic library in [5] uses explicit bounds on the length of messages, the number of signatures per key, and the number of inputs at each port. All these bounds are needed to model a strictly polynomial functionality which can be realised by a protocol with strictly polynomial machines. The notion of continuously polynomial security could help to remove these bounds which inhibit a clear and abstract presentation.

2 Review of Reactive Simulatability

In this section, we present the notion of Reactive Simulatability. This introduction only very roughly sketches the definitions, and the reader is encouraged to read [7] for more detailed information and formal definitions. A reader familiar with the model may skip this section and proceed to Section 3. Additionally, a glossary of important terms in the reactive simulatability framework can be found in Appendix A.

2.1 Outline

A protocol is simply a set of machines[7]. Intuitively, this includes only honest-acting protocol parties, but not an explicit adversary or corrupted parties. Assume a protocol $\hat{M}_1$. To define what we mean by saying that $\hat{M}_1$ is secure, we also define an idealized protocol $\hat{M}_2$. Most of the time, $\hat{M}_2$ consists only of a single machine that provides the same interface that all protocol machines of $\hat{M}_1$ (taken together as a whole) provide. Most importantly, the definition of $\hat{M}_2$ should be very simple and should guarantee "security by construction."

For example, $\hat{M}_1$ could be a concrete two-party protocol, where party P_1 inputs a message M and transmits it to party P_2 using a specific public-key encryption scheme. Party P_2 then outputs M upon receipt and decryption. (For simplicity, we do not consider a key distribution here.) Suppose we are interested in the question: "Does this protocol achieve a secure message transmission?" Then the corresponding idealisation $\hat{M}_2$

[7] We stick to the machine model of [24, 7], that is, to IO state automata.

is one single machine that inputs a message M on the same channel on which P_1 would get its input in $\hat{M}_1$. Then, $\hat{M}_2$ outputs M again on the output channel on which P_2 would output the decrypted message in $\hat{M}_2$.

Note that the definition of $\hat{M}_2$ does not involve any encryption scheme or key distribution. $\hat{M}_2$ captures the essence of a secure message transmission, and is independent of the way in which this goal is eventually achieved. In fact, $\hat{M}_2$ does not even involve transmitting a message over an insecure channel; it merely collects its input and returns its output. In this sense, $\hat{M}_2$ is "secure by construction."

The notion of Reactive Simulatability now seeks to compare $\hat{M}_1$ and $\hat{M}_2$ to express statements of the form "$\hat{M}_1$ achieves the security of $\hat{M}_2$." This is captured by the requirement that any weakness of $\hat{M}_1$ should be already present in the idealization $\hat{M}_2$. (The mentioning of "weakness" here is nothing to be worried about; $\hat{M}_2$ does not contain any weakness by construction, so this essentially only means that $\hat{M}_1$ does not contain any weakness as well. The only reason for this seemingly strange requirement is that it is formally not easy to express the statement "$\hat{M}_1$ does not contain any weakness" directly. Furthermore, there may be certain explicitly tolerated weaknesses in $\hat{M}_2$.)

More formally, we require that

- for every adversary A_1 that runs with $\hat{M}_1$ (and exhibits a potential weakness in $\hat{M}_1$),
- there exists an adversary A_2 that runs with $\hat{M}_2$ (and exhibits a corresponding weakness in $\hat{M}_2$),

- such that runs of A_1 with $\hat{M}_1$ look like runs of A_2 with $\hat{M}_2$.

This should hold in any protocol context in which $\hat{M}_1$, resp. $\hat{M}_2$ is used.

We give a more formal definition with more details now.

2.2 Detailed definitions

We say that protocol $\hat{M}_1$ (the *real protocol*) to be *as secure as* another protocol $\hat{M}_2$ (the *ideal protocol*, the *trusted host*), if for any adversary A_1 (also called the *real adversary*), and any *honest user* H, there is a *simulator* A_2 (also called the *ideal adversary*), s.t. the view of H is indistinguishable in the following two scenarios:

- The honest user H runs together with the real adversary A_1 and the real protocol $\hat{M}_1$
- The honest user H runs together with the simulator A_2 and the ideal protocol $\hat{M}_2$.

Note that there is a security parameter k common to all machines, so that the notion of indistinguishability makes sense.

This definition allows us to specify some trusted host—which is defined to be a secure implementation of some cryptographic task—as the ideal protocol, and then to consider the question, whether a real protocol is as secure as the trusted host (and thus also a secure implementation of that task).

2.3 Scheduling and runs

In order to understand the above definitions in more detail, we have to specify what is meant by machines “running together”. Consider a set of machines (called a *collection*).

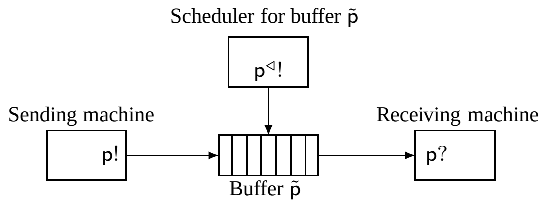

Fig. 1. A connection

Each machine has several *ports* with which it is connected to its outside world (i.e., to other machines, possibly in this collection). The machine receives input on its *in-ports*, and gives output on its *out-ports*. One distinguishes *simple* ports (i.e., simple in- and out-ports) and *clock* ports. While simple ports transport protocol data (e.g., messages, encryptions, etc.), clock ports are used for scheduling purposes only. To illustrate how this works, consider Figure 1. Each used *port name* (in this case p) represents a unidirectional connection between two machines. The names p in Figure 1 are all identical, and this is no accident: which ports connect where is solely determined by the port name p. Precisely the ports with identical port name are connected according to Figure 1.

The unique sending machine has the port $p!$, whereas the receiving machine has port $p?$. But once a message is sent over $p!$, it is not immediately delivered to $p?$. First, it is stored in a *buffer* $\tilde{p}$ (which is a very special type of machine used only for the purpose of message scheduling). The buffer itself delivers the message (to port $p?$) it has stored only upon explicit request. This request has to come from the clock out-port $p^{\triangleleft}!$, and the machine that has this port $\mathsf{p}^{\triangleleft}!$ is called the *scheduler* for buffer $\tilde{\mathsf{p}}$. Note that the scheduler for $\tilde{\mathsf{p}}$ may be the receiving machine itself, the sending machine, or a different machine.

As an example, consider our real protocol from above that transmits an encrypted message from P_1 to P_2. The connection from P_1 to P_2 could then be modeled as in Figure 1, where the sending machine is P_1, the receiving machine is P_2, and the scheduler for the buffer in between is the adversary A_1. That A_1 controls the buffer represents the fact that generally we assume the adversary to control the network *between* protocol parties.

Strictly speaking, this models a *secure* connection between P_1 and P_2, since the adversary controls *when* a message is delivered to P_2, but has no access to the contents. To model an authenticated channel, we would additionally force P_1 to send a copy of each sent message to A_1. (More detailed descriptions can be found in [7].)

Now, when a collection runs, the following happens: At every point in time, exactly one machine is activated. It may now read its simple in-ports (representing incoming network connections), do some work, and then write output to its simple out-ports. After such an activation the contents of the simple out-ports $\mathsf{p}!$ are appended to the queue of messages stored in the associated buffer $\tilde{\mathsf{p}}$. However, since now all messages are stored in buffers and will not be delivered by themselves, machines additionally have after each activation the possibility to write a number $n \geq 1$ to at most one clock out-port $\mathsf{p}^{\triangleleft}!$. Then the n-th undelivered message of buffer $\tilde{\mathsf{p}}$ will be written to the simple in-port $\mathsf{p}?$ and deleted from the buffer's queue. The machine that has the simple in-port $\mathsf{p}?$ will be activated next. (We stress again that in this description, p denotes always the same arbitrary, but fixed port name. Furthermore, for a given port name p, at most one machine with a port $\mathsf{p}?$ is allowed in a given system, so the machine with port $\mathsf{p}?$ is uniquely determined.)

So the clock out-ports control the scheduling. Usually, a connection is clocked by (i.e., the corresponding clock out-port is part of) the sender, or by the adversary. Since the most important use of a clock out-port is to write a 1 onto it (deliver the oldest message in the buffer), we say a machine clocks a connection or a message when a machine writes a 1 onto the clock port of that connection.

At the start of a run, or when no machine is activated at some point, a designated machine called the *master scheduler* is activated For this, the master scheduler has a special port, called the *master clock port* $\mathsf{clk}^{\triangleleft}?$.

Note that not all collections can be executed, only so-called *closed* collections, where all connections have their simple in-, simple out-, and clock out-port. If a collection is not closed, we call the ports having no counterpart *free ports*.

Discussion of the scheduling. This definition of scheduling is quite general and, in fact, somewhat complicated compared, e.g., to the (simple and indeterministic) scheduling in the π-calculus [23]. There are several ideas behind this. First, scheduling should generally be performed by the adversary. If, with a specific representation of attacks through an adversary as in our case, scheduling is performed *indeterministically*, then the adversary could effectively act indeterministically if he made his actions depend on his observations of the scheduling. This, however, conflicts with the fact that the RSIM framework considers probabilism in their security definitions. (Being indeterministic would, e.g., allow the adversary to “guess” the secret key of the employed public-key encryption scheme. However, this does not correspond to an “interesting” cryptographic attack on the scheme.)

2.4 Protocols

In order to understand how this idea of networks relates to the above sketch of reactive simulatability, one has to get an idea of what is meant by a protocol. A protocol is represented by a so-called *structure* $(\hat{M}, S)$, consisting of a collection $\hat{M}$ of the protocol participants (parties, trusted hosts, etc.), and a subset of the free ports of $\hat{M}$, the so-called *service ports* S. The service ports represent the protocol’s interface (the connections to the protocol’s users). The honest user can then only connect to the service ports (and to the adversary), all other free ports of the protocol are intended for the communication with the adversary (they may e.g. represent side channels, possibilities of attack, etc.).

Such free non-service ports are more commonly found with trusted hosts (i.e., the abstract idealizations of protocols that are used to define security), explicitly modelling their imperfections.

But also, free non-service ports can be found in protocols as a modeling of the adversary's power to interfere with the protocol scheduling and the interactions among honest protocol machines.

With this information we can review the above "definition" of security. Namely, the honest user H, the adversary, and the simulator are nothing else but machines, and the protocols are structures. The view of H is then the restriction of the run (the transcripts of all states and in-/output of all machines during the protocols execution, also called trace) to the ports and state of H.

A little more formally, we consider *configurations* of the real and ideal protocol. Concretely, a configuration of a structure $(\hat{M}, S)$ is a tuple $(\hat{M}, S, \mathsf{H}, \mathsf{A})$, where H is a valid user and A is a valid adversary for $(\hat{M}, S)$. Validity simply means that certain natural port requirements are. In particular, A only connects to non-service ports of $\hat{M}$ and to H, and H only connects to service ports of $\hat{M}$ and to A. The set of (in this sense) valid configurations for $(\hat{M}, S)$ is denoted by $\mathsf{Conf}(\hat{M}, S)$. For the purpose of comparing two structures $(\hat{M}_1, S)$ and $(\hat{M}_2, S)$ (with identical sets of service ports), it is also useful to define the set $\mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ of valid configurations $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1)$ for $(\hat{M}_1, S)$ such that H is also valid for $(\hat{M}_2, S)$.

Formally, we then define $(\hat{M}_1, S)$ as secure as $(\hat{M}_2, S)$ iff for every valid configuration $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$, there is a valid configuration $(\hat{M}_1, S, \mathsf{H}, \mathsf{A}_2) \in$

$\mathsf{Conf}(\hat{M}_2, S)$ (with the same honest user H), such that the views of H in both configurations is computationally indistinguishable.

What we have just defined is *standard security*, the default notion of security in the Reactive Simulatability framwork. If we demand that in the above definition, the ideal adversary A_2 does not depend on H, but only on A_1, we obtain a stricter definition that is called *universal security*. (The term *universal* refers here to the universality of A_2 with respect to the honest user H.) These two definitions are not equivalent [20].

We stress that here, we do *not* consider the concept of *black box security*, which intuitively demands that A_2 only depends on A_1 in a "black box" manner. This is an involved concept, and there are several definitions that try to capture black box simulatability (e.g., the definitions in [24, 6, 12] are all different).

2.5 Systems

The definition, as presented so far, still has one drawback. We have not introduced the concept of a corruption. This can be accommodated by defining so-called systems. A *system* is a set of structures, where to each "corruption situation" (set of machines, which are corrupted) one structure corresponds. That is, when a machine is corrupted, it is not present anymore in the corresponding structure, and the adversary takes its place. For a trusted host, the corresponding system usually consists of structures for each corruption situation, too, where those connections of the trusted host, that are associated with a corrupted party, are under the control of the adversary.

We can now refine the definition of security as follows: A *real system* Sys_1 is as secure as an *ideal system* Sys_2, if every structure in Sys_1 is as secure as the corresponding structure in Sys_2.

2.6 Combination

A very useful technical tool (that we will use later on) is the *combination* $\mathsf{comb}(\hat{M})$ of a set $\hat{M}$ of machines. Informally, $\mathsf{comb}(\hat{M})$ is the single machine that internally runs all machines in $\hat{M}$ as submachines. All ports of machines in $\hat{M}$ are preserved. In [24], the following statement is proven: If we substitute the combination $\mathsf{comb}(\hat{M})$ for a set $\hat{M}$ of machines in any closed collection, then the common view of all machines (including the submachines in $\mathsf{comb}(\hat{M})$ stays unchanged.

2.7 Composition

A major advantage of a security definition by simulatability is the possibility of *composition*. The notion of composition can be sketched as follows: If we have on structure or system A (usually a protocol) implementing some other structure or system B (usually some primitive), and we have some protocol X^B (having B as a sub-protocol, i.e. using the primitive), then by replacing B by A in X^B, we get a protocol X^A which is as secure as X^B. This allows to modularly design protocols: first we design a protocol X^B, and then we find an implementation for B.

We stress that these composability guarantees are bought at a high price. Namely, several protocol tasks are simply not realizable in a simulatability-based framework. This includes cryptographic building blocks such as bit commitment [13]. There have been suggestions how to avoid these impossibilities (e.g., [13, 15]), but they rely on additional assumptions or imply weakened composability guarantees.

3 Continuously Polynomial Security

In this section, a new notion of polynomial runtime for the adversary and the protocol user H, *continuously polynomial*, is defined. For users and adversaries subject to our definition, terms like "guaranteed delivery after polynomial time" can be defined in a meaningful way. The definition of protocols which are *polynomially shaped* of Section 4 together with the restriction to weakly polynomial machines (ps-wp protocols) will ensure without explicit lifetime bounds that only polynomial-time computations are performed within polynomial time as seen by the protocol user H.

First, we demand from the protocol user H that it is *weakly polynomial*, as defined in [7]. There it is required that there is a polynomial p, such that in each activation, H runs at most $p(k + |I|)$ steps, where k is the security parameter, and $|I|$ is the length of all inputs H has received so far.[8] We explicitly stress that this allows Hs that do not halt, i.e., run infinitely long. It also does not forbid H to send messages to itself (possibly

[8] This is similar to [17], where this approach is taken for the special case of secure function evaluations.

doubling the size of this “loopback” message every time to get twice the computational power for the next activation), or to receive large messages.

To make sure that the induced security notion stays sensible, we will restrict our attention to polynomial prefixes of H’s view. That is, we consider only things that happen during polynomially-sized prefixes of H’s view.[9] Here, the size of a view-prefix is the concatenated size of all inputs and outputs on H’s ports.

Second, an adversary A is required to be polynomial in H’s view. There are two obvious ways to do this: keeping A polynomial in the messages it *receives* from H, or keeping A polynomial in the messages it *sends* to H. We decided for a combination of both: in our definition, A must be polynomial in the size of the A-H-communication in *both* directions. We did so to give A more freedom: with the first notion, it would not be possible, in some cases, for A to simply forward protocol messages to H. Conversely, the second notion may forbid A to forward messages from H to the protocol. Thus, only our combined notion allows for a “dummy adversary” (an adversary that only acts as a relay between internal protocol lines and H). The concept of such a dummy adversary is useful, e.g., for proving concurrent composition properties.

[9] Alternatively, one could fix such a prefix with H and “hardwire” that bound into H to make it strictly polynomial in the traditional sense. However, in the case of standard security, the simulator is then chosen after H and thus knows the runtime bound of H. When trying to define notions like fairness (i.e., the property that the adversary eventually delivers messages), the simulator could then simply deliver all messages *after* the termination of the honest user H. This would circumvent the idea of a fair delivery.

However, this preliminary definition gives rise to a subtle problem with the proof of the composition theorem. In this proof, surrounding protocol machines are, for certain steps of the proof, simulated by the protocol user H. So an adversary considered in the proof of composability may have communication lines which are sometimes connected to protocol machines and sometimes connected to a protocol user mimicking these machines. Hence an adversary which is polynomial as described above could lose this property by the "regrouping" of machines during the composition proof, and the proof would fail.

Therefore, we introduce two specific communication lines which are *guaranteed* to connect the adversary and H. The ports for these two lines will have names of the form $\mathsf{cpoly\_}\ldots$, and such ports will not be allowed in any protocol. Now the total length of messages exchanged over these two specific lines is used as a lower bound for the "time" which has passed for H, and the adversary must be polynomial in this "volume" plus k. This volume includes the messages which is sent from the adversary to H in the same activation.

Counting a message, that is sent to H in the same activation, to the volume in which the adversary must be polynomial allows the adversary to receive (and, e.g., forward) arbitrarily long messages from the protocol. However, an adversary computing for a long time *must* send a long message to H to ensure that a long "elapse in time" is observed in the view of H. There is one important detail here: every prefix of the view of H is a sequence of results from whole activations. That is, if an adversary took a superpolynomial "debt" (e.g., by factoring a large integer), then the superpolynomial message which he is forced to send to H in the same activation will not be contained in any polynomial prefix of H's view. So whenever the adversary is performing a superpolynomial number of computation steps, it is ensured that the result will not, not even in parts, be considered in the definition of security.

A further condition we impose on the adversary is the following: The adversary is required to read all incoming messages completely. This seemingly unnecessary condition has important consequences: Assume a protocol (e.g., for secure message transmission) in which a ciphertext is transmitted. Assume further that for generating a realistic first bit of the ciphertext, a runtime linear in the length of the message is required.[10] Then a real adversary A could do the following: It intercepts the ciphertext, but reads only the first bit and forwards that bit to the honest user H. Since A only reads one bit, its running time is independent of the length of the transmitted message and it does not need to output anything on the $\mathsf{cpoly}\_\ldots$ connection. However, the simulator now has the task to generate a realistic first bit, which takes a runtime linear in the length of the message. In the case of universal security, since the simulator is chosen before the honest user, this length may be larger than the number of steps the simulator may run without output on

[10] An example would be if the protocol prepended the bit $H^l(0)$ to the ciphertext, where l is the length of the message, and H a suitable function so that computing $H^l(0)$ cannot be done faster than in $\Omega(l)$. Clearly, an IND-CCA2 secure cryptosystem would not lose its security by such an addition.

the $\mathsf{cpoly\_...}$ connection. So the simulator must output something there and the honest user can distinguish. By introducing the condition that the adversary reads all its inputs, this problem is fixed, since A now has to read the whole message, too, and hence also outputs on the $\mathsf{cpoly\_...}$ connection.

As a technicality, messages sent from the adversary A to H over the specific line which influences A's runtime must be delivered immediately to ensure the direct correspondence between runtime and messages received by H.[11]

Although we cannot guarantee that these requirements cover all technical artifacts that can happen, at least the sketched problems with existing definitions of polynomial runtime do not occur. Furthermore, Section 6 demonstrates that at least in natural examples, no odd effects happen.

We turn to the actual definition:

Definition 1 (Continuously polynomial honest users and adversaries). *We call an honest user* H continuously polynomial, *if it is weakly polynomial, has ports* $\mathsf{cpoly\_ha}!$, $\mathsf{cpoly\_ah}? \in \mathsf{ports}(\mathsf{H})$, *and the length function for* $\mathsf{cpoly\_ah}?$ *is* ∞ *in every non-final state (i.e., all inputs on* $\mathsf{cpoly\_ah}?$ *are written in full length to* H*'s view).*

We call an adversary A continuously polynomial, *if*

– *it has ports* $\mathsf{cpoly\_ha}?, \mathsf{cpoly\_ah}!, \mathsf{cpoly\_ah}^{\triangleleft}!$*, and*

[11] To facilitate the presentation, we say that a message m form a machine M is delivered immediately over a port $\mathsf{p}!$ if the receiving machine is activated with this message directly after M has entered a waiting state or a final state. In the model of [24, 7], this happens if the buffer $\tilde{\mathsf{p}}$ is empty and M performs the commands $\mathsf{p}! := m$; $\mathsf{p}^{\triangleleft}! := 1$.

– *there is a polynomial p, s.t. for any closed collection $\hat{C}$ of machines with $\mathsf{A} \in \hat{C}$, and any possible view of* A *in $\hat{C}$ (on security parameter 1^k), the following holds:*

- *Let t_μ be the total number of Turing steps of* A *up to its μ-th activation (inclusive). Let c_μ be the total length of outputs on* cpoly_ah! *and inputs on* cpoly_ha? *up to* A*'s μ-th activation (inclusive). Then for all $\mu \in \mathbb{N}$ it is*

$$t_\mu \leq p(c_\mu + k).$$

- *Whenever* A *sends a message on* cpoly_ah!*, it is delivered immediately.*
- A *never sets its length functions to anything other than ∞, and A always completely reads all incoming messages.*[12]

We can now define continuously polynomial security by simply restricting honest user and adversary to continuously polynomial ones:

Definition 2 (Continuously polynomial security). *Let $(\hat{M}_1, S)$ and $(\hat{M}_2, S)$ be structures (i.e., protocols), s.t. $\hat{M}_1$ and $\hat{M}_2$ have no port named* cpoly_ah *or* cpoly_ha. *Define*[13]

$$\mathsf{Conf}_{\mathsf{cpoly}}(\hat{M}_2, S) := \{(\hat{M}_2, S, \mathsf{H}, \mathsf{A}) \in \mathsf{Conf}(\hat{M}_2, S) \ : \ \mathsf{A} \textit{ and } \mathsf{H} \textit{ are continuously polynomial}\},$$

$$\mathsf{Conf}^{\hat{M}_2}_{\mathsf{cpoly}}(\hat{M}_1, S) := \mathsf{Conf}_{\mathsf{cpoly}}(\hat{M}_1, S) \cap \mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S).$$

[12] That is, in each activation, A takes at least $|I|$ steps, where $|I|$ is the length of A's input in that activation.

[13] Recall that $\mathsf{Conf}^{\hat{M}_2}(\hat{M}_1, S)$ and $\mathsf{Conf}(\hat{M}_2, S)$ are the sets of configurations $(\hat{M}, S, \mathsf{H}, \mathsf{A})$ so that H, A are valid honest user and adversary for the given protocol in the real and ideal model, respectively, and S is the set of service ports of the protocol $\hat{M} = \hat{M}_1, \hat{M}_2$, resp. Essentially, H and A are called valid if there are no open connections, and H only connects to service ports.

Less formally, the class of admissible honest users, adversaries and simulators is restricted to continuously polynomial ones.

If $view$ is a view of some machine, then by $\mathsf{pfx}_t(view)$ we denote the longest prefix, s.t. the total length of all inputs and outputs in that prefix is bounded by $t \in \mathbb{N}$ (we will call such a prefix a t-prefix).

We call $(\hat{M}_1, S)$ continuously polynomially as secure as *$(\hat{M}_2, S)$ (written: $\geq_{\mathsf{sec}}^{\mathsf{cpoly}}$), if for every configuration $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}_{\mathsf{cpoly}}^{\hat{M}_2}(\hat{M}_1, S)$, there exists a configuration $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}_{\mathsf{cpoly}}(\hat{M}_2, S)$ (essentially, this means that for continuously polynomial H, A_1 there is a continuously polynomial simulator A_2) s.t. for all polynomials l*

$$\mathsf{pfx}_{l(k)}\big(view_{conf_1,k}(\mathsf{H})\big) \approx_{\mathsf{poly}} \mathsf{pfx}_{l(k)}\big(view_{conf_2,k}(\mathsf{H})\big).$$

That is, for every adversary A_1 and user H that run with $\hat{M}_1$, we require the existence of an adversary A_2 that runs with H and $\hat{M}_2$, such that all polynomial prefixes of H's view are indistinguishable in both protocols.

For universal security, *(written: $\geq_{\mathsf{sec}}^{\mathsf{cpoly,uni}}$) we additionally require that A_2 does not depend on H.*

4 A Generalised Composition Theorem

This section gives a generalised composition theorem for not necessarily terminating protocols. To this end, a new notion of polynomial runtime for *protocols* is introduced.

For describing polynomial complexity, it is not only necessary to limit the computation time of a machine in each activation. It should also hold that superpolynomial “events” within the protocol yield a view for the user H having a superpolynomial representation. It should not pass unnoticed by H if a protocol machine gains superpolynomial computing power through a superpolynomial number of activations (which intuitively means that superpolynomial time must have passed) or by playing ping-pong with messages of growing size.

The definition of a *polynomially shaped* protocol ensures that each protocol machine can produce only messages of a total length which is polynomial in the length of the messages coming from outside the protocol, e.g. from the protocol user H or the adversary. The outside of the protocol is represented by a machine T in the definition below. If additionally, each protocol machine is weakly polynomial, then the number of Turing steps a protocol runs between two activations of H or the adversary is polynomially limited in the security parameter and the length of the overall protocol input.

Definition 3 (Polynomially shaped). *A collection $\hat{C}$ of machines containing no master scheduler is called p-shaped for a function $p : \mathbb{N} \to \mathbb{N}$, if for all machines T s.t. $\hat{C} \cup \mathsf{T}$ is closed (i.e., there are no open connections) the following property holds with overwhelming probability in the security parameter k:*

Let o_μ denote the total length of the output of all machines in $\hat{C}$ at position μ in the run of $\hat{C} \cup \mathsf{T}$. Similarly, i_μ denotes the total length of the input of machines in $\hat{C}$ on ports *coming from* T *(i.e., ports* $\mathsf{p}?$ *s.t.* $\mathsf{p}! \in \mathsf{ports}(\mathsf{T})$*). Further* a_μ *denotes the total number of activations of machines in* $\hat{C}$ *at that point. Then*

$$o_\mu + a_\mu \le p(i_\mu + k).$$

The following observation shows, in form of a lemma, that by restricting the honest user's view to polynomial prefixes, we also restrict whole runs to polynomial size. While this lemma is never explicitly used in the upcoming proofs, it already gives a good intuition of what effect it has to restrict the user's view. Also, it formalizes the intuition that protocol and adversary do not "run at a superpolynomially faster speed" than the protocol user.

Lemma 1. *Let* $(\hat{M}, S)$ *be a structure with* $\hat{M}$ *that is polynomially shaped and weakly polynomial. Assume that* $(\hat{M}, S, \mathsf{H}, \mathsf{A})$ *is a valid configuration of* $(\hat{M}, S)$ *with continuously polynomial* H *and* A*.*

Then, when the view honest user is restricted to a (fixed) polynomial size, the whole protocol (including adversary and honest user) can be simulated inside a single strictly polynomial-time machine.

Proof. Fix a polynomial $q = q(k)$. Because A is continuously polynomial, there is a polynomial $p_1 = p_1(k)$ such that A must output (in total) at least $q(k)$ bits to H if it runs at least $p_1(k)$ steps itself.

On the other hand, the runtime of $\hat{M}$ is polynomial in the size of its own input, since $\hat{M}$ is weakly polynomial. By assumption on A, the inputs that A gives to $\hat{M}$ are in turn polynomial in the size of H's view. Also, the input that H hands to $\hat{M}$ appears in H's own view. In summary, there is a polynomial $p_2 = p_2(k)$ such that H's view is at least of size $q(k)$ if $\hat{M}$ runs at least $p_2(k)$ steps.

Concluding, since H itself is weakly polynomial, there is an overall polynomial $p = p(k)$ such that the whole collection $\hat{M} \cup \{\mathsf{H}, \mathsf{A}\}$ runs at least $p(k)$ steps only if the view of $\hat{M}$ is of size more than $q(k)$. The lemma follows, if we consider a single machine M that simulates the whole collection, but halts as soon as the total number of steps taken by all machines exceeds $p(k)$.

In principle, the adversary or the user could try to gain superpolynomial computing power by playing "ping-pong" with a protocol which has no lifetime bound. However, this does not affect the security definition and computational assumptions can still be used, because security is defined by comparing only polynomial prefixes of the view of the user H. It is easy to see that results of a superpolynomial ping-pong cannot be contained in such a polynomial prefix if all machines are weakly polynomial, the protocol is polynomially shaped, and the user and the adversary are continously polynomial. A superpolynomial number of invocations of the protocol either directly implies a superpolynomial view of the using machine H or it implies a superpolynomial view of the adversary. A result of such a superpolynomial computation can only appear in a superpolynomial view of the adversary. For a continously polynomial adversary A and user H an event not visible in any polynomially view of the adversary cannot be visible in a polynomial prefix of the view of H. Even though the weakly polynomial machines could, in the long run, break any cryptosystem this does not imply distinguishability and computational assumptions can be used.

Next we generalise the composition theorem to continuously polynomial users and adversaries interacting with polynomially shaped protocols.

Note that the notion of polynomially shaped protocols is itself *not* closed under composition. A simple counterexample can be obtained from the two machines M_1, M_2 as follows. The machine M_1 has two input lines and one output line. It forwards each input to the output line and clocks the output line. The machine M_2 has one input line and one output line and acts as a repeater. It forwards each input to the output line and clocks the output line in the same actiovation. Both machines are polynomially shaped (as collections), but if we connect the two machines leaving one input line of M_1 open we obtain a collection which can generate infinite internal communication on one single input. This is a very bad effect as such a machine could run until it has solved some "hard" problem thereby invalidating computationmal assumptions.

So the generalised composition theorem states that a composed protocol is secure if it *remains* polynomially shaped. It is in the responsibility of the protocol designer to avoid "loops" when designing a protocol.

However, one can restrict the security definition to a subclass of polynomially shaped protocols which is closed under composition. Then the composition theorem still holds and e.g. loops cannot arise from composition.

A subclass of polynomially shaped protocols which is closed under composition can be obtained by restricting to protocols which give a shorter output then the total length of inputs given so far. This subclass contains a lot of natural protocols. It seems very difficult to find a subclass which is closed under composition and contains all natural protocols: for instance, a broadcast protocol has a larger output than the length of the input.

Intuitively, the generalised composition theorem says: Let a weakly polynomial protocol $\hat{M}_1$ use a sub-protocol $\hat{M}_0'$ such that the composition of $\hat{M}_1$ and $\hat{M}_0'$ is polynomially shaped. Let further $\hat{M}_0$ be a protocol which can connect to the protocol $\hat{M}_1$ in the same way as $\hat{M}_0'$ and for which the composition of $\hat{M}_1$ and $\hat{M}_0$ is polynomially shaped, too. Then the following holds: If $\hat{M}_0$ is at least as secure as $\hat{M}_0'$ according to Definition 2, then $\hat{M}_0'$ can be replaced by $\hat{M}_0$ without loss of security.

Theorem 1. *Let $(\hat{M}_0, S_0)$, $(\hat{M}_0', S_0)$, $(\hat{M}_1, S_1)$ be structures (i.e., protocols), s.t. no port in $\hat{M}_1$, $\hat{M}_0$, or $\hat{M}_0'$ is named* cpoly_ah *or* cpoly_ha. *Let then $(\hat{M}^{\#}, S) := (\hat{M}_1, S_1)\|(\hat{M}_0, S_0)$, $(\hat{M}^*, S) := (\hat{M}_1, S_1)\|(\hat{M}_0', S_0)$ (i.e., $\hat{M}^{\#}$ is the composition of $\hat{M}_1$ and $\hat{M}_0$, while $\hat{M}^*$ is the composition of $\hat{M}_1$ and $\hat{M}_0'$). Assume that*

– *The collections of machines $\hat{M}^{\#}$ and $\hat{M}^*$ are polynomially shaped.*

- *The collection of machines $\hat{M}_1$ is weakly polynomial.*
- *It is $(\hat{M}_0, S_0) \geq_{\mathsf{sec}}^{\mathsf{cpoly}} (\hat{M}_0', S_0)$.*
- *It is $\mathsf{ports}(\hat{M}_0') \cap S_1^c = \mathsf{ports}(\hat{M}_0) \cap S_1^c$.*[14]

Then we have

$$(\hat{M}^\#, S) \geq_{\mathsf{sec}}^{\mathsf{cpoly}} (\hat{M}^*, S),$$

i.e., $\hat{M}^\#$ is continously polynomially as secure as $\hat{M}^*$.

The same holds for universal security.

Proof. In the following proof, we assume all polynomials to be monotone. Furthermore, k always denotes the security parameter.

Let $conf_1 := (\hat{M}^\#, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}_{\mathsf{cpoly}}^{\hat{M}^*}(\hat{M}^\#, S)$ be given (i.e., let some suitable continuously polynomial honest user H and adversary A_1 be given). To prove the theorem, we have to find a continuously polynomial simulator A_2, s.t. $conf_2 := (\hat{M}^*, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}_{\mathsf{cpoly}}(\hat{M}^*, S)$ and

$$\mathsf{pfx}_l\big(view_{conf_1}(\mathsf{H})\big) \approx_{\mathsf{poly}} \mathsf{pfx}_l\big(view_{conf_2}(\mathsf{H})\big) \tag{1}$$

for all polynomials l.

To prove universal security, we additionally need, that A_2 does not depend on H.

W.l.o.g. we can restrict our attention to honest users which do not terminate. Other honest users can be transformed into an honest user H' which 1. does not terminate, 2. is

[14] This is a formally necessary structural condition on the available ports, which also appear in the original version of the composition theorem, cf. [7] for details.

continuously polynomial, and for which 3. the view of the original H is a prefix of the new H'.

Consider the combination $\mathsf{H}' := \mathsf{comb}(\{\mathsf{H}\} \cup \hat{M}_1)$ of H and $\hat{M}_1$. Since H and $\hat{M}_1$ are weakly polynomial, so is their combination H'. Since H does not terminate, the length function for $\mathsf{cpoly\_ah}$ of H' is always ∞, therefore H' is continuously polynomial.

Since $(\hat{M}_0, S_0) \geq_{\mathsf{sec}}^{\mathsf{cpoly}} (\hat{M}_0', S_0)$ there is a continuously polynomial simulator A_2, s.t.

$$\mathsf{pfx}_L\big(view_{\hat{M}_0 \cup \mathsf{H}' \cup \mathsf{A}_1}(\mathsf{H}')\big) \approx_{\mathsf{poly}} \mathsf{pfx}_L\big(view_{\hat{M}_0' \cup \mathsf{H}' \cup \mathsf{A}_2}(\mathsf{H}')\big)$$

for all polynomials L.

To show (1) from this, it is sufficient to show that for any polynomial l there is a polynomial L, s.t. the l-prefix of H is (with overwhelming probability) contained[15] in the L-prefix in H' (intuitively, this means that the view of H does not grow superpolynomially by inclusion of $\hat{M}_1$).

First, consider the view of H in the real model (i.e. in the collection $\mathsf{H} \cup \mathsf{A}_1 \cup \hat{M}^{\#}$). Fix a polynomial l. Let then the random variable μ_k be the index in the run of the last element of the l-prefix of H's view (more formally, the minimal μ_k, s.t. $\mathsf{pfx}_{l(k)}(view(\mathsf{H}))$ is contained in the first μ_k elements of the run).

Since A_1 is continuously polynomial, there exists a polynomial r (dependent on l) s.t. up to the μ_k-th step in the run the total length of A_1's output is bounded by $r(k)$.

[15] Here, containment is to be understood in the sense that the view of H' can be restricted to the subview of the submachine H of H'.

Since the total length of the output of H up to the μ_k-th step is bounded by $l(k)$ (by definition of l), we conclude that the total input of $\hat{M}^{\#}$ coming from H and A_1 is bounded by $l(k) + r(k)$. Since $\hat{M}^{\#}$ is polynomially shaped, it follows (by Definition 3) that the total output of $\hat{M}^{\#}$ is bounded by some polynomial $p(k)$ (dependent on l, r) with overwhelming probability.

So the length of the inputs and outputs of H' (being the combination of H and $\hat{M}_1 \subseteq \hat{M}^{\#}$) is bounded by $L_1(k) := l(k) + r(k) + l(k) + p(k) + p(k)$ (the summands being upper bounds for: in-/output of H; output of A_1; output of H; output of $\hat{M}_1$; output of $\mathsf{M}^{\#}$ (the latter appearing as input to H')). Therefore the l-prefix of H's view appear with overwhelming probability in an L-prefix of the view of H' (in the real model).

Using the fact that $\hat{M}^*$ is polynomially shaped, too, we get by analogous discussion that the l-prefix of H's view appear with overwhelming probability in an L_2-prefix of the view of H'. By choosing L as a polynomial bounding both L_1, L_2, the remaining goal is shown, so (1) follows.

5 Relations to Polynomial Security

Continuously polynomial security allows for users and adversaries which are not strictly polynomial. On the other hand, every strictly polynomial pair of user and adversary can be interpreted as continuously polynomial ones—only the formally necessary $\mathsf{cpoly\_ah}$ and $\mathsf{cpoly\_ha}$ connections have to be added (but they need not be used).

However, this inclusion does not immediately imply that continuously polynomial security can be related in any way to the well-known concept of strictly polynomial security (for which only strictly polynomially bounded users and adversaries are considered). Namely, in case of continuously polynomial security, not only real adversaries, but also simulators may be drawn from a larger pool of possible adversaries. So in principle, continuously polynomial security of a system could mean that even for strictly polynomially bounded real attacks, a simulator might be necessary which is *not* polynomially bounded; strictly polynomial security might not follow from continuously polynomial one.

Fortunately, we can still show the following, not immediately obvious relation between continuously polynomial and strictly polynomial security:

Theorem 2. *Let $(\hat{M}_1, S)$ and $(\hat{M}_2, S)$ be polynomially shaped structures (i.e., protocols) satisfying $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{cpoly}} (\hat{M}_2, S)$. Then $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$, i.e. continuously polynomial security implies strictly polynomial security for polynomially shaped protocols.*

Proof. Assume $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{cpoly}} (\hat{M}_2, S)$. To prove that $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (\hat{M}_2, S)$ we have to show that for every $conf_1 := (\mathsf{H}, \mathsf{A}_1, \hat{M}_1, S) \in \mathsf{Conf}_{\mathsf{poly}}^{\hat{M}_2}(\hat{M}_1, S)$ (i.e., for any strictly polynomial honest user H and real adversary A_1), there is a simulator A_2 with $conf_2 := (\mathsf{H}, \mathsf{A}_2, \hat{M}_2, S) \in \mathsf{Conf}_{\mathsf{poly}}(\hat{M}_2, S)$ (i.e., a strictly polynomial adversary), s.t.

$$view_{conf_1}(\mathsf{H}) \approx_{\mathsf{poly}} view_{conf_2}(\mathsf{H}). \qquad (2)$$

Without loss of generality we can assume that no port of H and A_1 is named $\mathsf{cpoly\_ah}$ or $\mathsf{cpoly\_ha}$.

First, since H and A_1 are strictly polynomial, and $\hat{M}_1$ is polynomially shaped, there is a polynomial p, s.t. $p(k)$ is with overwhelming probability an upper bound for the total length of all messages sent in a run of $\{\mathsf{H}, \mathsf{A}_1\} \cup \hat{M}_1$.

Therefore, we can construct a new real adversary A_1^p from A_1 as follows: We add new ports $\mathsf{cpoly\_ha}?$, $\mathsf{cpoly\_ah}!$, and $\mathsf{cpoly\_ah}^{\triangleleft}!$. A_1^p completely reads all its inputs and behaves as A_1 would (and ignores $\mathsf{cpoly\_ha}$-messages). Only if the total length of the incoming messages received throughout the run exceeds $p(k)$, all messages are forwarded to H through $\mathsf{cpoly\_ah}$ instead of simulating A_1. Clearly, since A_1 was strictly polynomial, A_1^p is continuously polynomial.

Similarly, we construct a new honest user H' from H: We add new ports $\mathsf{cpoly\_ah}?$, $\mathsf{cpoly\_ha}!$, $\mathsf{cpoly\_ha}^{\triangleleft}!$. The length function on $\mathsf{cpoly\_ah}?$ is set to ∞, but any input on this port is ignored. No output is ever sent on the new ports. Clearly, since H was strictly polynomial, H' is continuously polynomial.

Intuitively, we have added a new connection between H and A_1 which is not used at all, but needed to fulfil the formal requirements of continuously polynomial honest users and adversaries. Since the new connection is not used, and A_1^p's communication limit $p(k)$ is reached only with negligible probability, it immediately follows that

$$view_{conf_1}(\mathsf{H}) \approx view_{\mathsf{H}' \cup \mathsf{A}_1^p \cup \hat{M}_1}(\mathsf{H}'). \tag{3}$$

Since the machines H' and A_1^p are continuously polynomial, by $(\hat{M}_1, S) \geq_{\mathsf{sec}}^{\mathsf{cpoly}} (\hat{M}_2, S)$ there is a continuously polynomial simulator A_2^p s.t. for all polynomials l

$$\mathsf{pfx}_l\big(view_{\mathsf{H}'\cup\mathsf{A}_1^p\cup\hat{M}_1}(\mathsf{H}')\big) \;\approx_{\mathsf{poly}}\; \mathsf{pfx}_l\big(view_{\mathsf{H}'\cup\mathsf{A}_2^p\cup\hat{M}_2}(\mathsf{H}')\big) \tag{4}$$

Since in runs of $\mathsf{H}' \cup \mathsf{A}_1^p \cup \hat{M}_1$ the adversary A_1^p sends anything on $\mathsf{cpoly\_ah}$ only with negligible probability, A_2^p only sends with negligible probability on that port, too.

Therefore it is possible to construct a new simulator A_2 from A_2^p by removing the ports $\mathsf{cpoly\_ha}?$, $\mathsf{cpoly\_ah}!$, $\mathsf{cpoly\_ah}^{\triangleleft}!$ (here A_2 simply terminates when A_2^p would have sent on $\mathsf{cpoly\_ah}$). Since only with negligible probability data is ever transmitted over these ports, it is immediate that

$$view_{\mathsf{H}'\cup\mathsf{A}_2^p\cup\hat{M}_2}(\mathsf{H}') \approx view_{\mathsf{H}\cup\mathsf{A}_2\cup\hat{M}_2}(\mathsf{H}) \tag{5}$$

using the same identification of views as in (3).

Further, since A_2^p is continuously polynomial, and thus can only make a polynomial number of Turing steps while not receiving on $\mathsf{cpoly\_ha}$ or sending on $\mathsf{cpoly\_ah}$, it follows that A_2 is strictly polynomial.

Setting $conf_2 := (\mathsf{H}, \mathsf{A}_2, \hat{M}_2, S)$, and combining (3), (4) and (5), we get

$$\mathsf{pfx}_l\big(view_{conf_1}(\mathsf{H})\big) \approx_{\mathsf{poly}} \mathsf{pfx}_l\big(view_{conf_2}(\mathsf{H})\big) \tag{6}$$

for all polynomials l.

And since H and A_1 are strictly polynomial, and $\hat{M}_1$ is polynomially shaped, it follows from Definition 3 that there is a polynomial l s.t.

$$\mathsf{pfx}_l\big(view_{conf_1}(\mathsf{H})\big) = view_{conf_1}(\mathsf{H})$$

with overwhelming probability (i.e., that the view is almost always of length at most $l(k)$).

The analogue holds for H, A_2 and $\hat{M}_2$, so from (6) follows (2), which concludes the proof.

Note that the above proof does not work for universal security, since A_2 depends on p which again depends on H.

This theorem has several applications: first, it shows that continuously polynomial security is not “too weak” a security notion. In fact, anyone who would accept strictly polynomial security as a sufficiently strong security assumption should also find continuously polynomial security sufficiently strong.

Second, established results which need strictly polynomial security of a given system as a prerequisite can also be used with continuously polynomially secure systems. Consider the following example: You have proven continuously polynomial security for each of the many components of a large e-commerce protocol. The protocol and each of its components are—to avoid fixing a priori runtimes—formulated as a ps-wp protocol. Of course you use Theorem 1 to derive the security of the composed protocol. (Note that already this step would not have been possible with the strictly polynomial version of the composition theorem from [24], since for its application, the large protocol must be strictly polynomial-time.) Using [1, Theorem 5.1][16] and Theorem 2, you can now show that, e.g., integrity properties—as defined in [1]—the ideal version of the large protocol has are inherited by the composed (completely real) protocol. Since these steps involve composition of ps-wp systems, showing the same integrity properties of the composed real system is non-trivial when using only results which deal with strictly polynomial security.

6 A Simple Example

We will show the applicability of our definition using the very simple example of secure message transmission (SMT) over an authenticated channel using a one-time-pad. Note that despite its simplicity, such a functionality could not have been modelled in earlier approaches without bounding number and length of the messages (e.g., the SMT-functionality in [24] is parametrised by explicit bounds s and L for number and length of the messages).

To keep the presentation of this example simple, we assume a key exchange functionality KE that is has the following specification: When receiving a message of the form $\mathtt{1}^L$ from party $\mathsf{P}_{\mathsf{Alice}}$, a random $K \in \{\mathtt{0},\mathtt{1}\}^L$ is sent to the parties $\mathsf{P}_{\mathsf{Alice}}$ and $\mathsf{P}_{\mathsf{Bob}}$

[16] This theorem states the preservation of integrity properties and is applicable even to protocols which are not polynomial-time.

and a message $\mathsf{1}^L$ is sent (with immediate scheduling) to the adversary (informing him that a key exchange took place).[17]

We now want to implement the following functionality SMT: Whenever a message m is received from $\mathsf{P}_{\mathsf{Alice}}$, a message $\mathsf{1}^{|m|}$ is sent (and immediately scheduled) to the adversary, and the message m is sent to $\mathsf{P}_{\mathsf{Bob}}$. (Note that here the adversary can reorder the messages, since he may choose when to schedule the delivery of m from SMT to $\mathsf{P}_{\mathsf{Bob}}$.)

The protocol we propose for SMT is fairly straightforward. When receiving a message m, $\mathsf{P}_{\mathsf{Alice}}$ first requests a key of length $L := |m| + k$ from the functionality KE where k is the security parameter. Upon receipt of the key K it sends $c := (m\mathsf{0}^k) \oplus K$ to $\mathsf{P}_{\mathsf{Bob}}$ over an authenticated channel. $\mathsf{P}_{\mathsf{Alice}}$ repeats this protocol for each new message.

Then, upon reception of a key K from KE and a ciphertext c from $\mathsf{P}_{\mathsf{Alice}}$, $\mathsf{P}_{\mathsf{Bob}}$ calculates $\tilde{m} := c \oplus K$. If $\tilde{m}$ has the form $m\mathsf{0}^k$, $\mathsf{P}_{\mathsf{Bob}}$ outputs m.

Obviously this protocol is ps-wp, for each input of length L it generates a communication volume of $5L + 4k$.

We now give a proof sketch that this protocol indeed realises SMT: First, consider the case that no party is corrupted. Then, for each adversary A_1 we construct a simulator A_2 as follows: A_2 simulates the adversary, as well as $\mathsf{P}_{\mathsf{Alice}}$ and $\mathsf{P}_{\mathsf{Bob}}$. When the simulator A_2 receives a message $\mathsf{1}^L$ from SMT (informing it that a message of length L

[17] This key exchange functionality could then easily be implemented by doing an L-bit Diffie-Hellman-style key exchange.

is being sent), a random message $\tilde{m} \in \{0,1\}^L$ is given to $\mathsf{P}_{\mathsf{Alice}}$ as input, thus creating as fake view for the adversary. When $\mathsf{P}_{\mathsf{Bob}}$ finally outputs the message $\tilde{m} \in \{0,1\}^L$ (and the adversary schedules that output), the simulator schedules the delivery of the corresponding message m from SMT to the environment.

Since the adversary (and the honest user) does not learn the key K generated by KE, they may not distinguish whether the cipertexts intercepted by the adversary correspond to the messages generated by the honest user, or to random messages of the same length generated by the simulator. However, one subtle point must be taken care of: If several messages are in the process of being sent, the adversary may reorder the keys from KE differently on $\mathsf{P}_{\mathsf{Alice}}$'s and $\mathsf{P}_{\mathsf{Bob}}$'s side. Then it is possible that wrong messages get decoded. However, in order for this to happen, two generated keys have to match on the last k bits. Since the honest user H is continuously polynomial, for each prefix of length p of H's view at most $O(p(k))$ messages are sent, thus at most $O(p(k))$ keys generated, so the probability of such a collision of keys is bounded by $O(p(k)^2 2^{-k})$.

We add a short remark here: If instead of the one-time-pad an only computationally secure cipher had been used, we would additionally have to note that since the protocol is polynomially shaped, and the honest user and adversary are continously polynomial, the adversary and honest user together can run at most a polynomial number of steps. Hence, they cannot break the cipher with more than a negligible probability.

The last thing left to check for the uncorrupted case is that our simulator is indeed continuously polynomial. Whenever the simulator gets a message 1^L from SMT, a simulation of $\mathsf{P}_{\mathsf{Alice}}$ and $\mathsf{P}_{\mathsf{Bob}}$ runs. The runtime needed for this simulation is polynomial in L. However, in the simulation $\mathsf{P}_{\mathsf{Alice}}$ immediately sends a message of length $L+k$ which is passed to the simulated adversary. So the runtime needed for the simulation is polynomial in the length of the messages the simulated adversary gets. And since the simulated adversary is continuously polynomial, its runtime (which is also an upper bound for its incoming communication) is polynomial in its communication on the $\mathsf{cpoly}\ldots$ ports. So the total runtime of the simulator is polynomial in its communication on the $\mathsf{cpoly}\ldots$ ports (since all the communication of the simulated adversary on these ports is passed to H), and thus the simulator is continuously polynomial.

So at least in the uncorrupted case, our protocol is a continuously polynomially secure implementation of SMT.

The cases where $\mathsf{P}_{\mathsf{Alice}}$ or $\mathsf{P}_{\mathsf{Bob}}$ are corrupted are even easier, since here the simulator can learn the transmitted message. Checking that the simulator in these cases is also continuously polynomial is done very similarly to the uncorrupted case. We omit the details of these cases.

7 Applying our idea to the UC framework

We have shown how to allow for a more general class of polynomial-time protocols in the framework of reactive simulatability. Since we have not used any specific properties of the reactive simulatability framework, we believe that our approach can be adapted to the UC framework [8]. Several differences between the UC and the reactive simulatability framework that induce minor changes in our definitions are worth mentioning here:

– In the UC model, there is no concept of ports, the recipient of a message is dynamically specified by the sending machine. Therefore in Definition 1 we cannot consider the messages sent only over the $\mathsf{cpoly}\_\ldots$ ports. Instead, the messages intended to be sent over this connection must be marked in a special way, e.g., by a special prefix which is not allowed in messages sent to the protocol.

– In the UC model, indistinguishability of real and ideal protocols is not formulated in terms of the view, but in terms of the final output of the environment. Instead of quantifying over polynomial prefixes of the views in Def. 2 we would simply quantify only over environments that must terminate after a polynomial length of input and output. We stress that for "specialized simulator UC" (the UC equivalent of standard simulatability), this results in an order of quantifiers that differs from the one presented here for reactive simulatability. We do not know what side effects this may have for specialized simulator UC.

– In the UC model, it is possible that additional machines appear during the execution of the protocol (these can model e.g., new participants, newly invoked subroutine threads, multiple instances of a functionality). The definition of a polynomially shaped protocol (Def. 3) should therefore require, that the outputs of *all* machines (including submachines that are created only during the execution of the protocol) are bounded polynomially in the external input of *all* machines. Only considering the machines present at the beginning of the protocol execution would not be sufficient, of course.

8 Conclusions

We have motivated and introduced a novel formulation of the intuitive requirement of simulatable security with respect to polynomially bounded attacks and protocol runs. We have shown that the induced security notion allows for composition and is at least as strong as the established notion of strictly polynomial security.

We have presented our approach in the modelling of reactive simulatability [7]. The ideas presented here should be applicable to the UC model [8], too.

Many of the oddities that arise with a combination of simulatable security and a strict polynomial bounding (as with strictly polynomial security) of all entities in a protocol are settled by our approach. Nonetheless, more radical techniques are possible: e.g., message scheduling and scheduling of activations could be separately managed by distinguished entities. In such a setting, machines can send messages which are scheduled *while* the sending machine remains activated. Then, a very intuitive formulation of "polynomial runtime," which can even more closely model realistic protocol situations, would seem possible.

Acknowledgements

We thank Michael Backes and Ran Canetti for helpful discussions and comments.

References

1. M. Backes. *Cryptographically Sound Analysis of Security Protocols*. PhD thesis, Universität des Saarlandes, 2002. Online available at `http://www.infsec.cs.uni-sb.de/˜backes/papers/PhDthesis.ps.gz`.
2. M. Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. In R. Amadio and D. Lugiez, editors, *Concurrency Theory, Proceedings of CONCUR '03*, volume 2761 of *Lecture Notes in Computer Science*, pages 350–365. Springer-Verlag, 2003. Full version online available at `http://eprint.iacr.org/2003/114.ps`.
3. M. Backes. E-mail communication with the authors, June 2004.
4. M. Backes, D. Hofheinz, J. Müller-Quade, and D. Unruh. On fairness in simulatability-based cryptographic systems. In R. Küsters and J. Mitchell, editors, *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, pages 13–22. ACM Press, 2005. Full version online available at `http://eprint.iacr.org/2005/294`.
5. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM Conference on Computer and Communications Security, Proceedings of CCS 2003*, pages 220–230. ACM Press, 2003. Extended abstract, extended version online available at `http://eprint.iacr.org/2003/015.ps`.

6. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In M. Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, number 2951 in Lecture Notes in Computer Science, pages 336–354. Springer-Verlag, 2004. Online available at `http://www.zurich.ibm.com/security/publications/2004/BaPfWa2004MoreGeneralComposition.pdf`.

7. M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, Mar. 2004. Online available at `http://eprint.iacr.org/2004/082.ps`.

8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity, Oct. 2001. Full version of [10], online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

10. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

11. R. Canetti. On universally composable signature, certification and authentication. IACR ePrint Archive, Aug. 2004.

12. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, Dec. 2005. Full and revised version of [8], online available at `http://eprint.iacr.org/2000/067.ps`.

13. R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 2001. Full version online available at `http://eprint.iacr.org/2001/055.ps`.

14. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In E. Biham, editor, *Advances in Cryptology, Proceedings of EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer-Verlag, 2003. Full version online

available at `http://eprint.iacr.org/2004/116.ps`.

15. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

16. A. Datta, R. Küsters, J. C. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. In J. Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, pages 476–494. Springer-Verlag, 2005. Online available at `http://www.ti.informatik.uni-kiel.de/~kuesters/publications_html/DattaKuestersMitchellRamanathan-TCC-2005.ps.gz`.

17. O. Goldreich. Secure multi-party computation. Online available at `http://www.wisdom.weizmann.ac.il/~oded/PS/prot.ps`, Oct. 2002.

18. D. Hofheinz, J. Müller-Quade, and R. Steinwandt. On modeling IND-CCA security in cryptographic protocols. IACR ePrint Archive, Feb. 2003. Online available at `http://eprint.iacr.org/2003/024.ps`.

19. D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial runtime in simulatability definitions. In *18th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2005*, pages 156–169. IEEE Computer Society, 2005. Online available at `http://iaks-www.ira.uka.de/home/unruh/publications/hofheinz05polynomial.html`.

20. D. Hofheinz and D. Unruh. Comparing two notions of simulatability. In J. Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, pages 86–103. Springer-Verlag, 2005. Online available at `http://iaks-www.ira.uka.de/home/unruh/publications/hofheinz05comparing.html`.

21. D. Hofheinz and D. Unruh. Simulatable security and polynomially bounded concurrent composition. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2006*, pages 169–182. IEEE Computer Society, 2006. Full version online available at `http://eprint.iacr.org/2006/130.ps`.

22. Y. Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 394–403. IEEE Computer

Society, 2003. Online available at `http://eprint.iacr.org/2003/141`.

23. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–40, 1992.

24. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '01*, pages 184–200. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/066.ps`.

A Glossary

In this section we explain the technical terms of the reactive simulatability framework used in this paper. Longer and formal definitions can be found in [7].

$[\hat{C}]$: The completion of the collection $\hat{C}$. Results from adding all missing buffers to $\hat{C}$. $\mathsf{Conf}_x(\hat{M}_2, S)$: Set of ideal configurations that are possible for structure $(\hat{M}_2, S)$. $\mathsf{Conf}_x^{\hat{M}_2}(\hat{M}_1, S)$: Set of real configurations possible for structure $(\hat{M}_1, S)$. $\mathsf{ports}(M)$: The set of all ports, a machine or collection M has. **to clock**: To write 1 onto a clock out-port. ***EXPSMALL***: The set of exponentially small functions. ***NEGL***: The set of negligible functions (asymptotically smaller than the inverse of any polynomial). **buffer**: Stores message sent from a simple out- to a simple in-port. Needs an input from a clock port to deliver. **clock out-port $\mathsf{p}^{\triangleleft}!$**: A port used to schedule connection. **closed collection**: A collection is closed, if all ports have all their necessary counterparts. **collection**: A set of machines. **combination**: The combination of a set of machines is a new machine simulating the other machines. A set of machines can be replaced by its combination without changing the view of any machine. **composition**:

Replacing sub-protocols by other sub-protocols. **computational security**: When in the security definition, honest user and adversary are restricted to machines running in polynomial time, and the views are computationally indistinguishable. **configuration**: A structure together with an honest user and an adversary. **free ports**: The free ports of a collection are those missing their counterpart. **honest user**: Represents the setting in which the protocol runs. Also called environment. **intended structure**: A structure from which a system is derived making a structure for every corruption situation. **master clock port $\mathsf{clk}^{\triangleleft}?$**: A special port by which the master scheduler is activated. **master scheduler**: The machine that gets activated when no machine would get activated. **perfect security**: When in the security definition, the real and ideal run have to be identical, not only indistinguishable. Further the machines are completely unrestricted.[18] **run**: The transcript of everything that happens while a collection is run. Formally a random variable over sequences. $run_{conf,k,l}$ is the random variable of the run when running the configuration $conf$ upon security parameter k, restricted to its first l elements. If k is omitted, a family of random variables is meant. If l is omitted, we mean the full run. **service ports**: The ports of a structure to which the honest user may connect. They represent the interface of the protocol. As service ports

[18] In [7] a machine can in every activation for a given input and current state only reach one of a finite number of states (this convention has been chosen for simplicity [3]). However, this cannot even model the simple Turing machine that tosses (within one activation) coins until a 1 appears, and then stores the number of coin tosses. Therefore we will here adopt the convention that each state can have a countable number of potential successor states, from which one is chosen following some distribution depending on the input and the current state.

are most often ports of a buffer, they are sometimes specified through the set S^c of their complementary ports; S^c consists of all ports which directly connect to a service port. **simple in-port $\mathsf{p}?$**: A port of a machine, where it can receive messages from other machines. **simple out-port $\mathsf{p}!$**: As simple in-port, but for sending. **statistical security**: When in the security definition the statistical distance of polynomial prefixes of the views have a statistical distance which lies in a set of small functions $SMALL$ (in the security parameter k). Usually $SMALL = NEGL$. Further the machines are completely unrestricted.[18] **structure**: A collection together with a set of service ports, represents a protocol. **view**: A subsequence of the run. The $view(M)$ of some collection or machine M consists of the run restricted to the ports and states of M. Possible indices are as with runs.

B Length Functions

In Section 1, we mentioned that in security definitions which handle only strictly polynomial protocols it is often necessary to restrict the amount of data (lengths of inputs, number of invocations) a protocol can handle by some polynomial in the security parameter. We saw in Section 4 that the notion of continuously polynomial security allows to consider a much larger class of protocols, namely protocols which are ps-wp. This frees protocols from the necessity of terminating after some amount of input; rather protocols are only required to be polynomial in the "input from outside".

In earlier versions of the reactive simulatability definitions and the modelling of universal composability (e.g., in [8]), the following problem arose: consider e.g. the seemingly trivial functionality/trusted host, that has two in-ports and two out-ports (representing two parties) and on each pair of in-/out-port would just echo every input. In order to make this functionality strictly polynomial, it is now necessary to restrict the amount of echoed data to some polynomial p. Then the functionality has to terminate after receiving p messages on the first port, otherwise it might have to spend super-polynomial time by ignoring the incoming messages on that port. Then of course the functionality would not echo anything on the second port, even if no message has been echoed there yet. This introduces a flow of information between the two echo ports which certainly was not the intention of the original functionality.

To handle this artefact and allow functionalities to “switch off” selected ports, [1] introduces so-called *length functions*. These allow a machine to set the maximal length of messages it can receive through a given port at a given time. In particular, by setting the length function on a port to 0, the port is blocked and will not be activated by messages on that port, so that ignored messages do not consume runtime.

Since with continuously polynomial security, we do not need strictly polynomial protocols, one might wonder whether it is still necessary to consider and use length functions in this modelling, since these are an answer to a problem which is actually solved by our modelling in another manner. This question will be addressed in the present section, where we will show that we can in fact assume all protocol machines to have no length functions.[19]

The question is therefore whether a ps-wp protocol/functionality with length functions can be modified into another ps-wp protocol/functionality without length functions so that the security is not affected. Fortunately the following straightforward modification already has the desired property: we say a machine M' results from another machine M by removing length functions if M' has no length functions, but otherwise behaves as M does. That is, when receiving a message, the content of the message after the prefix the length function of M indicates is ignored, and only that prefix is used for the simulation of M (or the message is ignored, if the length function is 0). In other words, M' simulates the length functions of M without actually having them. When $\hat{M}$ is a collection, removing length functions means removing them from every machine in $\hat{M}$.[20]

Since obviously the difference between M and M' is only a formal property, not a difference in behaviour, we would expect M' to be a suitable replacement for M. This is confirmed by the following

[19] Formally, by a machine *without length functions* we mean a machine, whose length functions are ∞ in every non-final state.

[20] A careful study of the definition of machines in [7] shows, that formally we can define the machine resulting from removing length functions from a machine $\mathsf{M} = (name, Ports, States, \delta, l, Ini, Fin)$ simply as $\mathsf{M}' = (name, Ports, States, \delta, \infty, Ini, Fin)$, where ∞ denotes the length function yielding ∞ for all ports and non-final states.

Lemma 2. *Let $\hat{M}_i$ ($i = 1, 2$) be collections without master schedulers, and let $\hat{M}_i'$ result from $\hat{M}_i$ by removing length functions. Then it holds that*

– *$\hat{M}_i$ is polynomially shaped iff $\hat{M}_i'$ is.*

– *If $\hat{M}_i$ is weakly polynomial, so is $\hat{M}_i'$.*

– *The following are equivalent:*

$$(\hat{M}_1, S) \geq_{\mathsf{sec}} (\hat{M}_2, S), \qquad (\hat{M}_1, S) \geq_{\mathsf{sec}} (\hat{M}_2', S),$$

$$(\hat{M}_1', S) \geq_{\mathsf{sec}} (\hat{M}_2, S), \qquad (\hat{M}_1', S) \geq_{\mathsf{sec}} (\hat{M}_2', S)$$

Here $\geq_{\mathsf{sec}}$ denotes one of the following security notions: perfect / statistical / strictly polynomial / continuously polynomial in the flavours of standard or universal security.

The main idea of the proof is that the removal of length functions does not change the behaviour of the protocol, therefore the equivalences of the three security relations. Then it remains to be seen that the machines do not need superpolynomial runtime in the input to ignore the inputs (this shows the modified machines to be weakly polynomial), and that the amount of output does not change (this shows the resulting structures to be polynomially shaped). Note that such a property would not hold for strictly polynomial structures, since by removing a length function from a blocked port the resulting machine would have to ignore but accept an unbounded number of messages on that port, which is not allowed for strictly polynomial machines. The full proof goes as follows:

Proof. Let $\hat{C}_i$ be some collection, s.t. $\hat{M}_i \cup \hat{C}_i$ is a closed collection (i.e., no port is unconnected). Then removing the length functions from $\hat{M}_i$ yields a collection $\hat{M}_i' \cup \hat{C}_i$, so that the run of $\hat{M}_i' \cup \hat{C}_i$ differs from that of $\hat{M}_i' \cup \hat{C}_i$ only in the following points: 1. the inputs of machines in $\hat{M}_i'$ are changed (i.e., they are longer since with unmodified $\hat{M}_i$ they were added to the run in truncated form), 2. there are additional activations of machines in $\hat{M}_1'$ with empty outputs.

Now let any machine T without length functions be given, s.t. $\hat{M}_i \cup \mathsf{T}$ is closed. Consider then a run *run* of $\mathsf{T} \cup \hat{M}_i$ with security parameter k and the corresponding run *run*′ of $\mathsf{T} \cup \hat{M}_i'$ (i.e, the runs result from the same random choices). Let $\mu \in \mathbb{N}$. Then let t_μ denote the total length of the output of T, a_μ the number of activations of machines in $\hat{M}_i$, and o_μ the total length of the output of machines in $\hat{M}_i$, all up to the μ-th activation of T in *run* (cf. Definition 3). Let t_μ', a_μ', and o_μ' be defined analogously for *run*′. By setting $\hat{C}_i := \{\mathsf{T}\}$ the considerations at the beginning of the proof tell us that

$$a_\mu \le a_\mu', \qquad t_\mu = t_\mu', \qquad o_\mu = o_\mu'.$$

Note further that whenever a simple machine (no master scheduler) is activated, some other machine necessarily sent a nonempty message to that effect. This allows to conclude $a_\mu' \le t_\mu' + o_\mu'$.

If then $\hat{M}_i$ is p-shaped then from these inequalities we get with overwhelming probability for all μ

$$a_\mu' + o_\mu' \;\le\; t_\mu' + 2o_\mu' \;=\; t_\mu' + 2o_\mu \;\le\; t_\mu' + 2p(t_\mu + k) \;\le\; (2p + \mathrm{id})(t_\mu' + k),$$

so $\hat{M}_i'$ is $(2p + \mathrm{id})$-shaped.

If on the other hand $\hat{M}_i'$ is p-shaped, it is

$$a_\mu + o_\mu \leq a_\mu' + o_\mu' \leq p(t_\mu' + k) = p(t_\mu + k),$$

so $\hat{M}_i$ is p-shaped. So the claim follows that $\hat{M}_i$ is polynomially shaped iff $\hat{M}_i'$ is.

Now assume some weakly polynomial machine M is given, and M' results by removing length functions. Let some input sequence for M resp. M' be given. Then for activation μ we distinguish two cases: First, the length function of M is not zero on the port containing input. Then M' only has to ignore any trailing input, which can be done with an overhead polynomial in the running time of M in that activation. Second, if the length function of M is zero, the overhead of M' is constant, i.e., in particular polynomially bounded in the size of the non-empty input. So summarising we see that the overhead of M' is polynomial in the running time of M and the length of the input, so M' is weakly polynomial, too. Therefore this shows the claim $\hat{M}_1'$ is weakly polynomial if $\hat{M}_1$ is.

Considering again the results from the beginning of the proof, and letting $\hat{C}_1$ be the honest user together with the real adversary, we see that the view of the honest user is not changed by removing the length functions from the machines in $\hat{M}_1$, so $(\hat{M}_1, S) \geq_{\mathsf{sec}} (\hat{M}_2, S)$ is equivalent to $(\hat{M}_1', S) \geq_{\mathsf{sec}} (\hat{M}_2, S)$ and $(\hat{M}_1, S) \geq_{\mathsf{sec}} (\hat{M}_2', S)$ is equivalent with $(\hat{M}_1', S) \geq_{\mathsf{sec}} (\hat{M}_2', S)$. Similarly with $\hat{C}_2$ being the honest user together with the simulator, we see that $(\hat{M}_1, S) \geq_{\mathsf{sec}} (\hat{M}_2, S)$ is equivalent with $(\hat{M}_1, S) \geq_{\mathsf{sec}} (\hat{M}_2', S)$.

This shows the third claim.