ACM SIGACT News Distributed Computing Column 24

Sergio Rajsbaum*

Abstract

The Distributed Computing Column covers the theory of systems that are composed of a number of interacting computing elements. These include problems of communication and networking, databases, distributed shared memory, multiprocessor architectures, operating systems, verification, Internet, and the Web. This issue consists of:

- "Security and Composition of Cryptographic Protocols: A Tutorial (Part II)" by Ran Canetti.

The first part appeared in the previous SIGACT News, the September 2006 issue. Many thanks to Ran for his contributions to this column.

Request for Collaborations: Please send me any suggestions for material I should be including in this column, including news and communications, open problems, and authors willing to write a guest column or to review an event related to theory of distributed computing.

Security and Composition of Cryptographic Protocols: A Tutorial (Part II)

Ran Canetti [1]

Abstract

What does it mean for a cryptographic protocol to be "secure"? Capturing the security requirements of cryptographic tasks in a meaningful way is a slippery business: On the one hand, we want security criteria that prevent "all potential attacks" against a protocol; on the other hand, we want our criteria not to be overly restrictive and accept those protocols that do not succumb to attacks.

The first part of this two-part tutorial presented a general methodology for defining security of cryptographic protocols. The methodology, often dubbed the "trusted party paradigm", allows for defining the security requirements of a large variety of cryptographic tasks in a unified and natural way. We also reviewed a basic formulation of this paradigm. The second part, presented here, concentrates on the often subtle security concerns that arise under *protocol composition*, namely when a protocol runs alongside other protocols in a larger system. We first assert the limitations of the basic formulation from Part I in this setting; Next we present a stronger formulation and study its security-preserving composability properties.

*Instituto de Matemáticas, UNAM. Ciudad Universitaria, Mexico City, D.F. 04510. rajsbaum@math.unam.mx.
[1]IBM T.J. Watson Research Center, canetti@watson.ibm.com. Supported by NSF CyberTrust Grant #0430450.

1 Introduction

We briefly summarize the first part of the tutorial [C06] and review the contents of the second part.

Summary of Part I. The first part has motivated the need for a general framework for specifying the security requirements of cryptographic tasks, and for representing and analyzing cryptographic protocols. It also presented a definitional methodology, the *trusted party paradigm*, that enables specifying the security requirements of practically any cryptographic task in an intuitively satisfying way. This paradigm (which originates in [GMW87]) proceeds essentially as follows:

In order to determine whether a given protocol is secure for some cryptographic task, first envision an ideal process for carrying out the task in a secure way. In the ideal process all parties secretly hand their inputs to an external trusted party who locally computes the outputs according to the specification, and secretly hands each party its prescribed outputs. This ideal process can be regarded as a "formal specification" of the security requirements of the task. The protocol is said to securely realize a task if running the protocol amounts to "emulating" the ideal process for the task, in the sense that any damage that can be caused by an adversary interacting with the protocol can also be caused by an adversary in the ideal process for the task.

Finally, a basic formalization of the trusted party paradigm was given, along with a number of examples for how to use the general paradigm to specify the security requirements of common tasks. The basic feasibility results were also reviewed. These assert that practically any cryptographic task can be in principle realized according to this notion,

Part II. The second part concentrates on the goal of guaranteeing security of protocols in settings where different protocol instances run alongside each other in the same system. As demonstrated within, this setting turns out to be vastly different than the basic setting, discussed in Part I, where the entire system consists of a single execution of a single protocol.

Section 2 gives a general introduction to the concept of security-preserving protocol composition, with as little as possible discussion of specific notions of security. After exemplifying the security pitfalls in protocol composition, it describes the common ways in which protocols are often composed together in systems. It then addresses the question of what it means for a protocol to be "securely composable" with respect to a certain composition method.

Section 3 investigates the composability properties of the notion of security presented in Part I of this tutorial. (This is essentially the notion of [C00], extended to deal with reactive tasks.) It turns out that this notion guarantees that security is preserved under composition, as long as no two protocol instances run concurrently. As long as even *two* protocol instances run concurrently, security is no longer guaranteed.

Section 4 then presents and discusses a stronger notion of security, called universally composable (UC) security. The main advantage of this notion is that it guarantees security under essentially any type of composition. The main feasibility results for this notion are then reviewed, along with some directions for relaxing this notion. (In contrast with the basic notion, here some far-reaching impossibility results hold.) The tutorial is then concluded with some high-level remarks.

Finally, the Appendix contains a brief survey of the main works that contributed to our understanding of the trusted party definitional paradigm an its composability properties.

2 Security preserving protocol composition

In Part I of this tutorial, we have only considered security in a setting where the protocol in question is executed once, in isolation. This setting is indeed appropriate as a first one to consider when the goal is to understand the basic security properties of a protocol. However, analyzing security of a protocol in this stand-alone setting does not allow discovering potential weaknesses that come to play when the protocol runs alongside other protocols, or even alongside other executions of the same protocol. Consequently, so far the only method we have for analyzing security of some system is to model the entire system as a single protocol and analyze it as an atomic unit.

Analyzing security of systems in this way is challenging even for modest-size systems. When considering security of modern, multi-party, complex systems, the above one-shot approach becomes completely impractical. Furthermore, in open systems (such as today's Internet) whose makeup may change dynamically, and arbitrary new protocols might be added after the time of analysis, the above notion does not provide an adequate security guarantee to begin with.

Instead, we would like to be able to carve out pieces of a large system, analyze the security of each piece as if it were stand-alone, and then use the security of the individual pieces to deduce security properties of the overall system. Furthermore, this should be doable even when the overall system is not fully known at the time of analysis. To do that, we need to be able to argue about the behavior and security of protocols when running alongside, or composed with, other protocols. It turns out that this is a non-trivial task.

This section provides an introduction to the security issues associated with protocol composition. We start (in Section 2.1) with some examples that demonstrate various ways in which security might be compromised when composing together protocols that would be secure when run in isolation. We then proceed (in Section 2.2) to provide a brief taxonomy of the main types of composition operations considered in the literature. Finally, we motivate and present the concept of security-preserving composition (Section 2.3).

2.1 What might go wrong

To get some feel for the potential security pitfalls in protocol composition, we sketch three examples that demonstrate different ways in which protocols that are arguably secure in a stand-alone setting become insecure when run in conjunction with other protocols. In all the examples the problem is the same: The attacker uses information learned in one execution to "break" the security of another execution. In each example, this attack takes on a different form. The presentation is very informal throughout this section; indeed, the problems discussed are basic ones, and do not depend on the details of a specific definition of security.

Key Exchange and Secure Communication. This example demonstrates how two protocols can interact badly in settings where the parties uses *secret local outputs* obtained from one protocol as input for the other. It highlights the subtleties involved in maintaining overall security of a system that is designed in a modular way and consists of different interacting protocols.

Consider the task of Key Exchange, discussed in Section 4.3 of Part I. Recall that here two parties, an initiator I and a responder R wish to jointly generate a key that remains unknown to an external adversary. This key is typically used in order to encrypt and authenticate messages between I and R. Let π be key-exchange protocol that's proven to be secure in a stand-alone setting (say, with unauthenticated communication), and consider the protocol π' that's identical to π except that the following instruction is added to the code of I and R: "If the key has already been generated, and the incoming message includes the correct value of the key, then send a message `yes`. Else send `no`."

We first claim that, in a stand-alone setting, π' is just as secure as π. Indeed, since π is a secure protocol, then certainly it does not instruct any party to send the generated key in the clear. Furthermore, the adversary will be unable to figure out the value of the key just by interacting with the protocol. Thus, the added instruction will never be activated (except perhaps with negligible probability), and π' is effectively identical to π.

On the other hand, consider a setting where π runs in conjunction with a protocol that uses the key to encrypt messages. Furthermore, assume that the message takes one out of two possible values (say, either "sell" or "buy"), and furthermore that the encryption scheme in use is one-time-pad. That is, the encryption protocol obtains the key k from π', and has one party (say, I) send a ciphertext c which is either $k \oplus$ "$sell$" or $k \oplus$ "buy". (Here $\oplus$ stands for bitwise exclusive or.) We claim that now an adversary can use c in order to find out both k and the plaintext. In fact, all the adversary has to do is to compute $c' = c \oplus$ "buy" and send it to the other party as a message of π'. Now, if the encrypted message was "buy", then $c' = c \oplus$ "buy" $= k \oplus$ "buy" $\oplus$ "buy" $= k$ and R will respond with `yes`. If the encrypted message was "sell", then R will respond with `no`.

The point of this example (which is a variant of an observation of Rackoff from '95), is that π' allows the attacker to use the legitimate parties as "oracles" for testing guesses regarding the value of the key. As long as the system runs only π', and the key is never *used*, this "weakness" has no effect. However, as soon as the key is used and some values of the key become more plausible than others, the weakness becomes devastating. Finally, we remark that some prominent definitions of security for key-exchange in the literature (e.g., that of [BR93]) do not rule out this deceivingly simple weakness.

Parallel composition of Zero-Knowledge protocols. This example shows how certain protocols may be secure when run in a stand-alone setting, but lose their basic security properties as soon as even *two* instances of the *same* protocol are executed concurrently in the same system. This holds even if the system involves no other protocols. (Examples of a similar nature are given in [LLR02] for authenticated Byzantine Agreement protocols, and in [KLR06]. An interesting aspect of the [KLR06] example is that it remains valid even when all parties are computationally unbounded.)

Recall the task of Zero-Knowledge (ZK), discussed in Section 4.3 of Part I. Here we have a public binary relation R. The prover P transmits a value x to a verifier V, and in addition wants to convince V that it (P) has a secret "witness" w such that $R(x, w)$ holds. This should be done so that V learns nothing more than the fact that P has such a witness.

The example is essentially the one in [GK89, F91]. It uses a combinatorial gadget, which we describe here only very informally. Assume we have a "puzzle system" where both the prover and the verifier can generate puzzles p that have the following properties. First, the prover can solve any given puzzle. Second, the verifier cannot feasibly solve puzzles; in fact, the verifier cannot even verify the validity of a solution. That is, even for puzzles generated by the verifier, the verifier cannot distinguish between a valid solution or a random, invalid one. (Such a gadget can be shown to exist, either via allowing the prover to be computationally unbounded, as in [GK89], or based on some trapdoor information held by the prover, as in [F91].)

Now, let π be a ZK protocol (for some relation R). Construct the protocol π' where the parties first run π, and then continue with the following interaction. First, P sends a random puzzle p to V. Then, V responds with a purported solution s for p, plus a puzzle p'. If s is a correct solution, then P reveals the secret witness w. Otherwise, P sends to V a solution s' for the puzzle p' provided by V.

We first argue that if π is ZK in a stand-alone setting, then π' satisfies the ZK requirement. Intuitively, this holds since, by assumption, V cannot solve puzzles, thus in a stand-alone execution of π P never reveals w (except perhaps with negligible probability). Furthermore, the fact that P provides V with a solution s'

to the puzzle p' is not really a problem in a stand-alone setting, since V cannot distinguish s' from a random value (which V could have generated by itself).

However, when a prover P runs two concurrent executions of π' with V (say, on the same input (x, w)), then a cheating V can easily extract the witness: V first waits to receive the puzzles p_1 and p_2 from P in the two sessions. It then sends (s, p_2) to P in the first session, for some arbitrary s. In response, V gets from P a solution s_2 to p_2, which it returns to P in the second session. Since s_2 is a correct solution, P will now disclose w.

Malleability of commitment. The following example highlight two issues. First, it demonstrates that a multi-execution system brings forth entirely new *security concerns* that do not exist in a stand-alone setting. Second, it highlights the difficulty in arguing security of a protocol with respect to *arbitrary* other protocols, especially protocols that have been designed specifically so as to "interact badly" with the analyzed protocol.

Recall the task of commitment, discussed in Section 4.3 of Part I. This is a two-stage task, where in the commit stage a committer C provides a receiver R with a "commitment value" c to a secret value x. In the opening stage C discloses x. (For simplicity, we assume here that both stages consist of a single message from C to R.) There are essentially two security requirements: A secrecy requirement, that x remains completely secret throughout the commit stage, and a binding requirement, that there is at most one value x that R will accept as a valid opening for a commitment value c.

Consider the following natural sealed-bid auction protocol: Each party commits to its bid (say, over a broadcast channel). Once the bidding stage is over, all parties open their commitments and the winner is decided. It is tempting to deduce that any secure commitment protocol would suffice here. It turns out, however, that there exist natural commitment protocols that satisfy both secrecy and binding (and in fact satisfy the definition from Section 4.3 of Part I), but which are susceptible to the following attack: An attacker might use a commitment c, that was generated by an honest committer C that commits to a value x, to generate a commitment c'; later, when C opens c to value x, the attacker is able to "open" c' to a value x' that is related to x (say, $x' = x + 1$).[2] Of course, this attack is devastating for the auction protocol, in spite of the fact that neither secrecy nor binding of the commitment protocol were violated here. Rather, a new concern arises, namely the need to maintain "independence" between the committed values in different executions of the protocol. This concern (which is called non-malleability in the literature, following [DDN00] who pointed out this concern and showed how to address it) does not come to play in a stand-alone execution.

Several non-malleable commitment schemes have been constructed, using different set-up and network assumptions. Indeed, these schemes are not susceptible to the above attack. However, notice that this attack captures only a limited aspect of the "independence" problem, where there are only *two* executions, and more importantly the executions are of the *same protocol.* What about independence between an execution of a commitment protocol π and an execution of another protocol, π'? This seems like a hopeless goal, especially when π' is designed specifically to interact with π. To see this, consider the following example. Let π be any (even non-malleable) commitment protocol, and let π' be the protocol where in order to commit to a value x, one runs π on committed value $x - 1$. Assume that C commits using protocol π, and that a malicious C' announces that it commits using protocol π'. Now, when C sends its commitment string c, all

[2] For instance, consider Pedersen's commitment scheme [P91]: Let G be an algebraic group of large prime order, and assume that two random generators g, h of G are publicly known (say, they are announced by the auctioneer). In the commit stage, C sends $c = g^x \cdot h^r$, where $x \in G$ is the committed value, and $r \stackrel{R}{\leftarrow} G$. To open, C sends x and r and R accepts if $c = g^x \cdot h^r$. Here secrecy is perfect (and unconditional). Binding holds under the assumption that computing discrete logarithms in G is infeasible. In fact, a somewhat augmented variant also realizes $\mathcal{F}_{\text{COM}}$ as in Definition 3 in Part I. Still, consider a malicious committer C' that wishes to commit to the value committed by C, plus one. Then all C' has to do is to generate $c' = c \cdot g$. When C' sees a valid opening (x, r) of c, it can generate the valid opening $(x + 1, r)$ of c'.

C' has to do is to copy c as its own commitment. When C opens c to a value x, C' can use the same opening to open c to the value $x + 1$. Note that C' can use π' in a completely different context, say with a set of parties that do not know about C or π. This will make the attack hard to detect.

Indeed, guaranteeing security against these "chosen protocol attacks" seems intuitively impossible. However, contrary to this intuition, Section 4 demonstrates that such attacks *can* be protected against in most cases, via appropriate use of some set-up assumptions.

2.2 How can protocols be composed

This section provides a brief taxonomy of the different types of protocol composition operations considered in the literature, namely the various ways of combining together protocols in a single system. Taking another point of view, these operations naturally correspond to different ways of de-composing a complex system into separate pieces, which we would like to view as individual "protocols."

We first list some salient parameters for composition operations. Next we discuss some well-studied settings in terms of these parameters. Finally, we show how all these settings can be cast as special cases of a single, general composition operation.

Timing coordination: This parameter refers to the possible ways in which the messages of the individual executions can interleave with each other. Salient options include:

Sequential composition: Here no two messages of different protocol executions may interleave. That is, when ordering the events of sending and receiving of messages in the system along a common time axis, then all the events related to each protocol execution must form an uninterrupted sequence.

Enforcing global sequentiality requires each party to locally coordinate the different executions in terms of the timing of message sending. It also requires some level of global coordination among the parties, to guarantee that no party "gets ahead of the pack" and starts sending messages of a new execution before other parties completed prior executions.

Non-concurrent composition: This is a somewhat more general variant that allows "nesting" of protocol executions, as long as there is no "interleaving" of messages. That is, assume some message of execution e_1 was delivered, and at a later point a message of execution e_2 was delivered. Then, once another message of execution e_1 is delivered, messages of execution e_2 can no longer be delivered. Also here, guaranteeing global non-interleaving requires global coordination.

Parallel composition: Here it is assumed that the messages in each protocol execution are naturally associated with "rounds", where a "round i message" is sent only in response to receiving a "round $i - 1$ message". The composed execution of a given set of protocol executions allows any interleaving of protocol messages, as long as all the "round i messages" of all the executions are delivered before any "round $i + 1$ message" is delivered.

While this composition method is also quite restrictive and requires global timing coordination among the executions, it is natural in synchronous systems where messages are naturally associated with rounds.

Concurrent composition: Here any interleaving of messages from different protocol executions is allowed. Clearly, concurrent composition allows both sequential and parallel composition as special cases. It also allows many other special types of interleaving, such as the common case where various executions wait for an external global event to proceed. Concurrent composition

is very powerful in that it requires no timing coordination among the various executions. Indeed, the timing of events may of course be adversarially coordinated.

We note that the level of timing coordination between executions is in principle unrelated to the synchrony guarantees of the underlying communication network. For instance, different executions can be composed concurrently and "asynchronously" even when each execution is synchronous within itself. Also, sequential or non-concurrent composition can be sometimes guaranteed even in a completely asynchronous communication network.

Input coordination: This parameter refers to the possible relations between the input values to the various protocol executions. We distinguish three variants:

Same input: Here each party has the same input value for all the executions. Taking the role of a party in a protocol as part of its input, this means that each party has the same role in all the executions it participates in. Still, different executions may include different parties. (A somewhat more restrictive case is where the same set of parties participate in all executions.)

Fixed inputs: Here the inputs to different executions can be arbitrarily different from each other. In particular, a party may have different roles in different executions. (For instance a party may be a receiver in one execution of a commitment protocol, and a committer in a different execution.) Still, all inputs, including the set of participants in each execution, are fixed in advance before the execution of the composed system starts.

Adaptively chosen inputs: Here each input to each party in each execution can be determined adaptively based on the current state of the composed system. This is of course the most general setting of this parameter, and includes the above two settings as special cases. Variants of this setting depend on the amount of information available to the entities that choose the inputs; for instance, the inputs of a given party may be determined only based on the information available to that party, or alternatively based on the current global state of the system.

Protocol coordination: This parameter refers to the possible relations between the *programs*, or *codes*, executed in different executions. We distinguish two main cases:

Self composition: Here all executions run the same program. A closely related case is where different executions may run different programs, but the set of programs is fixed and known in advance. (Indeed, running a fixed number of programs is equivalent to running a single program that multiplexes between the many programs depending on the input.)

General composition: Here a given execution of a protocol may be running alongside arbitrary other protocols (i.e., programs) that may not be known in advance. Furthermore, these programs may be determined adaptively, depending on the protocol in question and potentially even on the current state of the composed system. This is indeed a highly adversarial setting. Still, it seems to adequately model the situation in open and unregulated networks such as the global Internet.

State coordination: This parameter refers to the amount and type of information that is shared among different executions. We distinguish the following cases:

Independent states: This is the "classic" case of protocol composition where different executions have no shared state. That is, The local variables of each execution within each participant are seen only by that execution. Also, the random choices made within each execution are independent from those in other executions. (Of course, different executions can still have related inputs.)

Joint state: Here some variables or random choices may be visible to multiple protocol executions. One salient example of such a setting is a protocol where the same secret signing key for a signature scheme is used in multiple protocol executions (say, for generating multiple session keys). Another example is a "common random string", namely a public string that is drawn from some distribution and is assumed to be globally available in the system. Here the "joint part" is typically modeled as a "subroutine protocol" that takes input from and provides output to multiple protocol executions. We note that, although this type of composition is somewhat non-traditional, without it it would not be possible to de-compose such systems into smaller components — such as a single exchange of a key in a key-exchange protocol.

Number of executions: This parameter determines the number of protocol executions that run together in the composed system. It is crucial, in the sense that, for most settings of the rest of the parameters and for each i, it is possible to construct protocols that "compose securely" as long as at most i executions run together, but break as soon as the system involves $i + 1$ executions. Three salient settings are:

Fixed number of executions: Here the number of executions is fixed in advance. In particular, it is does not depend on the input, nor on a security parameter.

Bounded number of executions: The maximum number of executions may depend on public information, such as the security parameter or some global input, but is known when designing the protocol. In particular, the complexity of the protocol may depend on this bound.

Unbounded number of executions: The number of executions is chosen adversarially in an adaptive way, and is limited only by the runtime of the adversary. In particular, it may depend on the execution, and remain unknown to any or some of the parties.

Some studied settings. Almost any combination of the above parameters yields a meaningful setting for the study of security-preserving protocol composition. Yet, some settings have been the focus of much dedicated study, both in the context of specific primitives such as key-exchange, zero-knowledge or commitment, and in more general contexts. We briefly mention some of these settings. (For sake of conciseness and brevity, we do not expand here on the specific contributions of the works mentioned below, nor on the notions of security that are obtained in each of these settings.)

Perhaps the simplest setting to consider is that of sequential self-composition with same input. This setting is studied in the context of zero-knowledge in [GO94] and general function evaluation in [B91]. In the case of parallel and concurrent composition, it was demonstrated in Section 2.1 that zero-knowledge is not preserved under same-input self-composition of even *two* executions [GK89, F91]. Still, protocols that remain zero-knowledge in this setting exist [GO94]. This primitive case of concurrent composition is generalized in a number of directions. One direction is that of multiple concurrent instances, while keeping the restriction to same input. Obtaining zero-knowledge in this case, especially when the number of executions is *unbounded* and not known a priori, turns out to be a non-trivial problem that requires new protocol techniques [F91, DNS98, RK99, PRS02].

Another extension is to the case of concurrent self-composition when parties can have different inputs in different executions. The case of *two* copies and *fixed* inputs, studied in [DDN00] and its many follow-up papers, brings about the concern of *malleability*, or *input independence.* Generalizing to adaptively chosen inputs and a bounded number of concurrent instances, or else to fixed inputs and an unbounded number of sessions, requires yet another set of techniques (e.g. [P04]), while a general solution for the case of adaptively chosen inputs and an unbounded number of instances requires either some initial set-up [L04] or some relaxation of the notion of security [BS05].

So far, we discussed the case of self-composition. General composition was first studied in the non-concurrent case, where it was shown to preserve some general ideal-model based notions of security for function evaluation [MR91, C00]. Notions of ideal-model based security that are preserved under concurrent general composition were subsequently developed, e.g. [DM00, PW00, C01, MRST06]. Methods for arguing about composition with joint state were developed in the context of general composition, e.g. [CR03, CDPW07].

Universal composition. Next we describe a single composition operation (namely, a way of combining several protocols into a single protocol) that can be used to express all the settings discussed above. Having such a generic composition operation is convenient in that composability properties proven with respect to this operation apply to all settings. Furthermore, this specific operation seems to closely correspond to the structure of actual protocols. It also meshes nicely with the trusted party paradigm (we'll see this in the next section).

The composition operation, which we call universal composition, is a natural extension of the "subroutine composition" operation on sequential algorithms to distributed protocols. That is, let ρ be a protocol (i.e., a set of instructions for the participants), where the instructions of each party include an instruction to provide input to some "subroutine program," denoted ϕ, as well as instructions on what to do when the subroutine program ϕ generates output. (Using the formalism of Section 3.1 in Part I, the system contains ITIs running the code ρ, alongside ITIs running the code ϕ; the ITIs running ρ write in the input tapes of ITIs running ϕ, and the ITIs running ϕ write on the subroutine output tapes of ITIs running ρ.)

Let π be another protocol. Then the composed protocol, denoted $\rho^{\pi/\phi}$, is the protocol where the code of each party is the same as that of ρ, with the exception that the instance of ϕ is replaced by an instance of π. That is, each instruction to provide input to ϕ is replaced by an instruction to provide the same input to π, and the instructions to be carried out upon receipt of an output from ϕ are now carried out upon receipt of an output from π. It is stressed that the replacement is done separately within each party running ρ. In particular, an execution of $\rho^{\pi/\phi}$ involves an entire distributed instance of protocol π, where the different parties of this instance exchange messages among themselves.

The case where ρ uses multiple (potentially unboundedly many) instances of ϕ is defined analogously. That is, each instance of ϕ is replaced by an instance of π. It is assumed that protocol ρ has some mechanism to distinguish among the various instances of ϕ; this mechanism remains the same with respect to distinguishing among the instances of π. While in principle there is no need to specify a particular mechanism, for sake of concreteness we assume that ρ associates a unique session identifier (SID) with each instance of ϕ, where the SID is included in all inputs to and outputs from this instance. Then the composed protocol $\rho^{\pi/\phi}$ keeps the same SIDs as in ρ.

Now, the various settings described above for protocol composition can be captured via different codes for the "high-level protocol", ρ. For instance, concurrent self composition with same input is captured by the protocol ρ that simply runs multiple instances of its subroutine ϕ on the same input, and outputs whatever these subroutines output. To capture fixed or adaptively chosen inputs modify ρ accordingly, to obtain the inputs for the various instances in advance or during the course of the execution. General composition is captured by allowing ρ to be arbitrary.

Sequential self composition in a synchronous execution setting is captured by the protocol ρ that runs multiple instances of its subroutine ϕ, one after the other in a sequential way, either with the same input or with different inputs, as may be the case, and outputs whatever these subroutines output. To capture parallel composition, ρ runs all instances of ϕ together and in each round delivers all the current messages of all instances. in lockstep. Non-concurrent general composition allows ρ to be arbitrary, as long as all parties start and end each instance of ϕ at the same global round, and only messages of this instance of ϕ are sent

while this instance is active.

Finally, we note that the above description of universal composition treats the protocol ϕ merely as a formal "placeholder" in the description of protocol ρ. Yet, as seen in the next section, protocol ϕ can have a central role in specifying the security properties required from protocol composition.

2.3 Security preserving composition

So far, we have treated the security requirements from cryptographic protocols under composition in an informal way. That is, we have expressed the desire to have protocols that "maintain their security properties" when run alongside other protocols. We have also observed, in Section 2.1, that some desirable security properties may no longer hold in such settings. How can we formalize the security requirements from protocols under composition?

One way, of course, is to list a set of specific properties that we would like to guarantee, and demonstrate that these properties hold. For instance, for protocols that evaluate some function of the inputs of the parties, we can require that *correctness* is preserved, in the sense that in all instances the outputs of the parties agrees with the value of the function at their inputs. If the evaluated function is probabilistic then we can also require that the randomness used in each execution is in some sense "independent" of the randomness used in other executions. We can also require that secrecy of certain values is preserved even in the composed system. (An example of a setting where such a specific requirement is made is that of concurrent zero-knowledge, mentioned above.) An additional specific requirement is that of *input independence*, or *non-malleability*, namely that the outputs of a protocol execution will not depend in "illegitimate ways" on secret inputs to another execution.

However, in the spirit of Section 2 in Part I, we prefer to make a single, unified security requirement that would imply all of the specific requirements mentioned above, as well as other potential requirements. And, again, in the spirit of Section 2 in Part I, we use the ideal-model paradigm to do so.

Recall that, by this paradigm, a protocol π is considered a secure implementation for a given task if it behaves in essentially the same way as an ideal protocol ϕ for that task, where the ideal protocol instructs all parties to privately hand their inputs to a trusted party which computes the desired outputs and hands them back to the parties. Furthermore, the requirement "π behaves in essentially the same way as ϕ" is formalized to mean "π emulates ϕ" as in Definition 2 in Part I. The compositionality requirement we make is analogous: Consider a task that is represented via an ideal protocol ϕ, and let π be a protocol that uses (potentially multiple instances of) ϕ. We say that π implements the task in a composable way with respect to π, if π continues to behave essentially the same when the instances of ϕ are replaced by instances of π. In the language of universal composition and emulation, we want that the protocol $\rho^{\pi/\phi}$ will emulate the original protocol ρ.

Definition 1 *Protocol π emulates an ideal protocol ϕ with ρ-composable security if it holds that $\rho^{\pi/\phi}$ emulates ρ.*

We observe that the notion of composable security indeed guarantees all the compositionality requirements listed above. Indeed, when ρ makes subroutine calls to the various instances of the ideal protocol ϕ, it is guaranteed that each instance of ϕ returns a correct function value, regardless of the activity in the rest of the system. The definition of emulation guarantees that ρ continues to exhibit essentially the same behavior when the instances of ϕ are replaced with instances of π. Similarly, since the trusted parties operate independently of each other, their outputs are computed using independent random choices. Also, the secrecy of data in each individual execution is guaranteed regardless of the rest of the system. Input independence is guaranteed since each party has to explicitly provide its inputs to each instance of ϕ, based only on its

legitimate outputs from the various instances of ϕ. Again, the definition of emulation guarantees that ρ continues to exhibit essentially the same behavior when the instances of ϕ are replaced with instances of π.

The above line of reasoning considers a single "calling protocol", ρ. Secure composability with respect to different types of composition operations are captured by considering the corresponding classes of the calling protocol, as described in Section 2.2.

One potential shortcoming of Definition 1 is that the notion of emulation, as defined so far, does not necessarily imply composable security. This means that Definition 1 does not necessarily guarantee that security is preserved under "iterated composition". That is, the fact that π emulates ϕ with ρ-composable security does not necessarily imply that $\rho^{\pi/\phi}$ emulates ρ with ρ'-composable security for an arbitrary ρ' (or even for $\rho' = \rho$). See more discussion on this point in Section 4.

3 The composability properties of basic security

Intuitively, the trusted-party definitional paradigm as formalized in Section 4 in Part I appears to be "inherently compositional". In particular, the notion of protocol emulation seems to almost immediately guarantee — at least in spirit — that no external process will be able to distinguish between the emulating protocol and the emulated one. Thus it seems natural to expect that basic security will imply ρ-composable security with respect to *any* polytime protocol ρ. That is, it is natural to expect that if protocol π realizes an ideal functionality $\mathcal{F}$ with basic security (as in Definition 3 in Part I), then $\rho^{\pi/\phi}$ would emulate ρ for any polytime protocol ρ.

It turns out that this this intuition can indeed be formalized for some types of composition, namely *non-concurrent* general composition. However, as soon as the non-concurrency condition is violated this intuition is incorrect. Details follow.

Recall that in *non-concurrent* composition it is guaranteed that no two protocol instances run concurrently with each other, except for simple nesting (see Section 2.2). More precisely, say that a protocol ρ is non-concurrent if any execution of ρ^{π}, with any subroutine protocol π, has the following property: Order all messages sent in the system along a single time axis, and Let e_1 and e_2 be two protocol executions where the first message of e_1 was sent before the first message of e_2. Then, once the first e_2-message is sent, no e_1-messages are sent until the last e_2 message is delivered. Then we have:

Theorem 2 ([C00]) *Let π and ϕ be protocols such that π emulates ϕ as in Definition 2 in Part I. Then, π emulates ϕ with ρ-composable security for any* non-concurrent *protocol ρ.*

Proof idea. We very briefly sketch the main idea behind the proof. For simplicity we concentrate on the case where ρ uses only a single instance of ϕ. Since no two instances of ϕ run concurrently, it is straightforward to extend the proof to the case where ρ uses multiple instances of ϕ.

Let $\mathcal{A}$ be an adversary that interacts with parties running ρ^{π}. We need to construct an adversary $\mathcal{A}_{\rho}$, such that no environment $\mathcal{E}$ will be able to tell whether it is interacting with $\rho^{\pi/phi}$ and $\mathcal{A}$ or with ρ and $\mathcal{A}_{\rho}$. The idea is to construct $\mathcal{A}_{\rho}$ in two steps: First "cut out" of $\mathcal{A}$ a real-life adversary, denoted $\mathcal{A}_{\pi}$, that operates against protocol π as a stand-alone protocol. The fact that π emulates ϕ guarantees that there exist an adversary ("simulator") $\mathcal{A}_{\phi}$, such that no environment can tell whether it is interacting with π and $\mathcal{A}_{\pi}$ or with ϕ and $\mathcal{A}_{\phi}$. Next, construct $\mathcal{A}_{\rho}$ out of $\mathcal{A}$ and $\mathcal{A}_{\phi}$.

We sketch the above steps. Essentially, $\mathcal{A}_{\pi}$ represents the "segment" of $\mathcal{A}$ that interacts with protocol π. That is, $\mathcal{A}_{\pi}$ expects to receive in its input (coming from the environment $\mathcal{E}$) a configuration of $\mathcal{A}$, and simulates a run of $\mathcal{A}$ starting from this configuration. Once the execution of this instance of π has completed, $\mathcal{A}_{\pi}$ outputs the current configuration of the simulated $\mathcal{A}$.

Adversary $\mathcal{A}_\rho$ is essentially the adversary $\mathcal{A}$, where the segment that interacts with π is replaced by the simulator $\mathcal{A}_\phi$. That is, $\mathcal{A}_\rho$ starts by invoking a copy of $\mathcal{A}$ and following $\mathcal{A}$'s instructions, up to the point where the first message of π is sent. At this point, $\mathcal{A}$ expects to interact with π, whereas $\mathcal{A}_\rho$ interacts with ϕ. To continue running $\mathcal{A}$, adversary $\mathcal{A}_\rho$ runs $\mathcal{A}_\phi$, with input that describes the *current* state of $\mathcal{A}$. The interaction between $\mathcal{A}_\phi$ and ϕ is emulated by $\mathcal{A}_\rho$, using $\mathcal{A}_\rho$'s own access to ϕ. Recall that the output of $\mathcal{A}_\phi$ is a (simulated) internal state of $\mathcal{A}$ at the completion of protocol π. Once protocol π completes its execution and the parties return to running ρ, adversary $\mathcal{A}_\rho$ returns to running $\mathcal{A}$ (starting from the state in $\mathcal{A}_\phi$'s output) and follows the instructions of $\mathcal{A}$.

The validity of the construction is demonstrated by reduction: Assume that there is an environment $\mathcal{E}$ that distinguishes between an interaction with ρ and $\mathcal{A}_\rho$, and an interaction with $\rho^{\pi/\phi}$ and $\mathcal{A}$. Then one constructs an environment, $\mathcal{E}_\pi$, that distinguishes between an interaction with ϕ and $\mathcal{A}_\phi$, and an interaction with π and $\mathcal{A}_\pi$. Essentially, $\mathcal{E}_\pi$ runs $\mathcal{E}$, where the interaction between $\mathcal{E}$, ρ, and the segment of $\mathcal{A}$ that does not interact with the subroutine, is simulated internally. The interaction with the subroutine (either π or ϕ) and its adversary (either $\mathcal{A}_\pi$ or $\mathcal{A}_\phi$) is taken to be the interaction with the actual external protocol and adversary.

Finally, it is shown that the view of $\mathcal{E}$, when simulated by environment $\mathcal{E}_\pi$ that interacts with adversary $\mathcal{A}_\pi$ and parties running π, is distributed identically to the view of $\mathcal{E}$ that interacts with adversary $\mathcal{A}$ and parties running $\rho^{\pi/\phi}$. Similarly, the view of $\mathcal{E}$, when simulated by environment $\mathcal{E}_\pi$ that interacts with adversary $\mathcal{A}_\phi$ and parties running ϕ, is distributed *identically* to the view of $\mathcal{E}$ that interacts with adversary $\mathcal{A}$ and parties running ρ. (These two equivalences are essentially standard bisimulation arguments from the distributed systems community.) It is stressed that the bisimulation is exact and the distributions over the views are identical. Consequently, the "loss in security" incurred by the theorem is zero.

Basic security under concurrent composition. Can these composability results be extended to concurrent protocol composition? It turns out that the answer is strongly negative. In fact, we have already seen a counter-example: As argued in Section 4.3 of Part I, the set of protocols that realize f_{ZK}^R, the zero-knowledge function with relation R, roughly corresponds to a class of zero-knowledge protocols for the language L_R. Furthermore, as seen in section 2.1, it is possible to construct zero-knowledge protocols (for any given language) where running even two instances of the protocol in parallel allows the verifier to extract the entire witness. Indeed, this example can be easily extended to come up with a protocol π and a relation R such that π realizes f_{ZK}^R, but $\rho^{\pi/\phi}$ does not emulate $\rho^{f_{\text{ZK}}^R}$ where ρ is the protocol that runs two instances of its subroutine concurrently, on the same input. Similarly, it can be demonstrated that basic security does not guarantee non-malleability. Further discussion on why this is the case appears in the next section.

4 Universally Composable Security

In spite of the intuitive appeal and expressive power of the basic notion of security developed in Sections 3 and 4 in Part I, we have seen in Section 3 that this notion provides only limited compositionality guarantees: As soon as protocols are allowed to run concurrently — as they often do in actual composed systems — no security guarantees are given. Furthermore, we have seen examples where security breaks down completely.

Universally Composable (UC) security is a strengthening of the basic notion of security, that comes to address the issue of preserving security under concurrent composition. The goal is to have a notion of security that guarantees security under all commonplace types of protocol composition, and in particular the ones described in Section 2.2. This should be done without losing on the intuitive appeal and expressive power, and with as mild as possible additional requirements from protocols.[3] This section is organized as

[3]The term *universally composable security* might be somewhat confusing, given that the term *universal composition* was used

follows. Section 4.1 presents and motivates the notion of UC security and its relation to the basic notion from previous sections. Section 4.3 very briefly presents the known results regarding the realizability of this notion. Finally, Section 4.3.2 touches upon directions for relaxing UC security while retaining some of its security and composability guarantees.

4.1 The definition

Why does the basic definition of security from Sections 3 and 4 in Part I fail to guarantee security under concurrent composition? When reviewing the definition in an attempt to answer this question, one notices that the model of protocol execution as defined there allows the environment, which models the "external world", to exchange information with the adversary, which models a coordinated attack against a single protocol execution, only once at the beginning of the execution, where the environment provides information to the adversary, and once at the end, where the adversary provides output to the environment. In a way, this modeling treats an execution of a protocol as an "atomic step," where there is no "information flow" between the protocol execution and the external environment during the protocol execution. (Some protocols may indeed allow the adversary and environment to exchange additional information via the inputs and outputs to the parties, but such exchanges are protocol-dependent and cannot be used in general arguments on the model.)

This modeling is indeed appropriate in a system where only a single protocol execution is active at any given point in time. However, it seems insufficient for capturing the often "circular" information flow among protocol executions that run *concurrently.* In particular, it fails to capture situations such as the ones described in Section 2.1, where an attacker uses information gathered in one execution in order to extract information in another execution, and then uses the extracted information back in the first execution.

UC security is aimed at correcting this shortcoming of the basic definition. The idea is to modify the model of protocol execution so as to allow the environment and the adversary to interact freely throughout the course of the computation. That is, whenever the environment is activated, it is allowed to provide input not only to the parties running the protocol, but also to the adversary. Similarly, whenever the adversary is activated, it can provide output to the environment. This means that the environment and the adversary can communicate before and after each activation of a party running the protocol; in other words, the "atomic unit" of uninterrupted execution is now a single activation of a party, rather than an entire execution of a protocol. As seen below, this change to the model turns out to suffice for proving general composability. It also changes the set of acceptable protocols in a radical way.

Another, more technical modification of the model from Section 4 in Part I is to add more structure to the communication model in order to facilitate the distinction between protocol instances in a composite system. A more detailed description follows.

The system model. We use the system model from Section 4.1 in Part I, with one change. To facilitate the distinction among different protocol executions in a system, we assume that the identity of each party (i.e., the contents of the identity tape) consists of two fields: a session ID (SID) and a party ID (PID). The SID is used to specify the "session", or "protocol instance" to which the ITI "belongs", and is joint to all the ITIs in a session. The PID distinguishes the ITI from other ITIs in that protocol instance. It can also be used to associate an ITI with a "cluster" of ITIs, such as the cluster of procedures running on a single physical

to denote a specific composition operation. In particular, several different definitions of security are known to be "universally composable", in the sense that they support a universal composition theorem such as Theorem 4 below. We thus use the acronym "UC security" to refer to the specific notion discussed here. (The duplicate terminology can be somewhat justified by Proposition 5 below, which implies that UC security is in a sense a *minimal* extension of basic security that is preserved under universal composition.)

computer. An instance of a protocol π with SID s in a certain configuration of a system is now defined to be the set of ITIs that have code π and SID s.

Remark: The above modeling of the SID is only one out of many possible ways for representing and distinguishing among protocol instances in a composite system. Still, the fact that all ITIs in a protocol instance have the same SID, which is determined by the invoking ITI, seems like a natural choice. In particular, it is easy to realize (say, by letting the party which initiates an instance to determine the SID and communicate it to all other participants). It also often facilitates the design and analysis of protocols, by providing to the participants a common value that is unique to the instance.

The protocol execution experiment. The protocol execution experiment is the same as the one in Section 4.2 in Part I, with the following two modifications. First, as mentioned above, we allow the environment to provide inputs to the adversary at any time. Similarly, we allow the adversary to provide outputs to the environment at any time.

Second, recall that in the model of Section 4.2 in Part I all parties (ITIs) invoked by the environment must run the same protocol (ITM). Furthermore, all the parties were treated as participating in a single protocol instance. In the present model, unless explicitly restricted, the environment can in principle invoke multiple protocol instances, by giving different SIDs to different parties. To keep in the spirit of a single instance, we require that all the parties invoked by the environment participate in the same protocol instance, namely they all have the same SID. (The value of the SID is of course chosen by the environment.)

Analogously to the notation $\text{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ from Section 4.2 in Part I, let $\text{UC-EXEC}_{\pi,\mathcal{A},\mathcal{E}}(x)$ denote the random variable describing the output of environment $\mathcal{E}$ when interacting with adversary $\mathcal{A}$ and protocol π on input x (for $\mathcal{E}$) in the present model. $\text{UC-EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ denotes the ensemble $\{\text{UC-EXEC}_{\pi,\mathcal{A},\mathcal{E}}(x)\}_{x\in\{0,1\}^*}$.

Restricting the environment to run only a single protocol instance significantly simplifies the model and the analysis of protocols. On the down side, it comes at the price of some restrictions on the class of protocols which can be composed in a secure way. See more discussion in Section 4.2.

The ideal process. The ideal process remains the same as the one in Section 4.2 in Part I, with the following exception: We restrict attention to ideal functionalities $\mathcal{F}$ where an instance ignores inputs that do not specify its SID. (Recall that the SID of an instance is determined by the ITI that called this instance for the first time.) Similarly, we assume that $\mathcal{F}$ includes its SID in all of its outputs. We note that this restriction is not essential; its purpose is to simplify the modeling and analysis of protocols.

Protocol emulation. The notion of protocol emulation and realizing functionalities is the same as in Section 4.2 in Part I, except that it relates to the present execution experiments:

Definition 3 *UC protocol emulation and realization A protocol π* UC-emulates *protocol ϕ if for any PT adversary $\mathcal{A}$ there exists a PT adversary $\mathcal{S}$ such that for all PT environments $\mathcal{E}$ that output only one bit:*

$$\text{UC-EXEC}_{\phi,\mathcal{S},\mathcal{E}} \approx \text{UC-EXEC}_{\pi,\mathcal{A},\mathcal{E}}$$

A protocol π UC-realizes *an ideal functionality $\mathcal{F}$ if π UC-emulates the ideal protocol for $\mathcal{F}$.*

4.2 Composability

The main attraction in UC security is that it guarantees composable security with respect to almost any PT calling protocol. That is, we restrict the way a protocol receives inputs from and provides output to the surrounding system in the following natural way: We assume that the only component of the "subroutine

protocol" that receives inputs from the outside and provides outputs to the outside is the "top-level program". More precisely, recall that an ITI P is called a subroutine of an ITI P' if P takes input from P' or provides output to P'; P is a subsidiary of P' if it a subroutine of P' or of a subsidiary of P'. Say that an ITM π is subroutine respecting if any ITI P running the code π has the property that all subsidiaries of P are subroutines only of P or of subsidiaries of P. A protocol is subroutine respecting if it is subroutine respecting as an ITM. We have:

Theorem 4 *Let π and ϕ be subroutine-respecting PT protocols such that π UC-emulates ϕ. Then $\rho^{\pi/\phi}$ UC-emulates ρ for any PT protocol ρ.*

Historical note. Theorem 4 was first proven in [PW00, PW01] for the case where ρ invokes a single instance of the subroutine protocol ϕ. (These proofs are set in their formalism, which has several technical differences from the one presented here.) The case where ρ may invoke an unbounded number of instances of ϕ was first proven in [C01] in a model similar to the one presented here, and subsequently re-proven in a number of different models, e.g. [BPW04, DKMR05, K06, CKLP06].

Proof idea. At high level, the proof of Theorem 4 follows the same steps as the proof of Theorem 2, with the exception that here protocol ρ may call multiple instances of ϕ, where these instances run concurrently. Consequently, the adversary $\mathcal{A}_\rho$ that interacts with protocol ρ concurrently invokes multiple instances of the simulator $\mathcal{A}_\phi$, where each instance of $\mathcal{A}_\phi$ interacts with a single instance of ϕ. In order to be able to carry out the overall interaction with ρ and the environment in a globally consistent manner, $\mathcal{A}_\rho$ uses the fact that each instance of $\mathcal{A}_\phi$ outputs the necessary information *after each activation.* This allows $\mathcal{A}_\rho$ to use information generated in one instance of $\mathcal{A}_\phi$ as input to another instance of $\mathcal{A}_\phi$. (Recall that in the case of basic security $\mathcal{A}_\phi$ is required to generate output only at the end of the execution; such a guarantee would not suffice for the present case.)

As in the proof of Theorem 2, the proof of validity of $\mathcal{A}_\rho$ proceeds by reduction to the validity of $\mathcal{A}_\phi$. The main difference from Theorem 2 is that here there are multiple instances of a subroutine protocol (either ϕ or π), running concurrently. Thus, we need to demonstrate that no environment can tell the difference between the case where all instances of ϕ are replaced by π and the case where none of the instances of ϕ are replaced by π. This is done via a standard hybrid argument, namely by considering multiple hybrid executions where in each execution one more instance of ϕ is replaced by π. An environment that distinguishes between two consecutive instances is now translated into an environment that contradicts the validity of $\mathcal{A}_\phi$. We omit further details.

4.2.1 Discussion

To interpret Theorem 4 recall that, for any given calling protocol ρ, the fact that $\rho^{\pi/\phi}$ UC-emulates ρ implies that replacing the instances of ϕ by instances of π does not change the behavior of ρ with respect to PT adversaries in a noticeable way; in particular, it does not introduce any new vulnerabilities to $\rho^{\pi/\phi}$. Furthermore, recall that any of the composition scenarios mentioned in Section 2.2 (with the exception of joint-state composition, discussed below) can be captured as universal composition with some set of calling protocols. Thus, Theorem 4 guarantees security-preserving composition in any of these scenarios. Some additional aspects of the theorem are discussed next.

Modular protocol analysis. The fact that Theorem 4 puts very few restrictions on the calling protocol ρ makes it conducive to carrying out the plan from the preamble of Section 4.3 of Part I in a way that meshes

naturally with the structure of common protocols. That is, the theorem allows de-composing protocols to many simple subroutines, analyzing each subroutine separately, and then deducing the security of the overall protocol from the security of the subroutines. In particular, the partitioning to subroutines can be nested in an arbitrary way. This is a powerful methodology, especially given the fact that rigorous analysis of even simple cryptographic protocols tends to be dauntingly complex.

Enabling sound symbolic and automated analysis. Another advantage of Theorem 4 is that it allows to "abstract away" cryptographic imperfections such as computational bounds and error probabilities, while maintaining soundness of the abstractions. This enables applying automated proof tools that require symbolic representations of protocols (as in, say, [DY83]) and cannot directly handle asymptotic modeling and cryptographic imperfections. To do that, first devise functionalities that capture in an ideal way the security properties of the cryptographic primitives (say, encryption in the case of [DY83]) used in the analyzed system. Next, re-write the protocols to be analyzed in a symbolic, non-asymptotic model that corresponds to having access to the devised ideal functionalities. Now, one can apply an automated tool to the symbolic representation of the protocol. Finally, use the UC theorem to deduce that, if the cryptographic protocols in use UC-realize the devised ideal functionalities, then the overall system enjoys the same properties proven for the abstract version. Some works that take this approach include [BPW03, CH04, SB+06].

Two things should be kept in mind, however, when taking this approach. First, for the analysis to be of value, one has to make sure that the asserted abstract security properties have meaningful translations to concrete security properties of concrete protocols.

Second, the complexity of automated analysis tools grows very rapidly as a function of the number of messages and sessions in the analyzed system (see e.g. the undecidability and NP-completeness results in [EG82, DLMS99]). Consequently, a viable instantiation of the above approach would need to break down protocols to simple subroutines and analyze each subroutine separately as a single session. Here the UC theorem is once again a crucial enabler.

Representing communication models. Another use of Theorem 4 is for modular representation of various communication models within the basic model of computation described above. That is, to capture a given communication model, simply devise an ideal functionality $\mathcal{F}$ that guarantees the abstractions provided by that model. Now, designing protocols in that model is translated to designing protocols that run in the basic model and make calls to $\mathcal{F}$. In order to further simplify the code of $\mathcal{F}$, one can allow for multiple instances of $\mathcal{F}$ to run concurrently, where each instance deals with a single use of the underlying model (say, a single sending of a message in the case of an authenticated communication abstraction). Here we do not necessarily intend to *realize* $\mathcal{F}$ in an algorithmic way; rather, $\mathcal{F}$ merely serves as a functional description of the desired abstraction. Still, in some cases the same ideal functionality can be used both as the basis for a communication model and as a target to be realized by cryptographic protocols. Some communication models that have been captured this way include authenticated communication, secure communication, and synchronous communication (see e.g. [C01]).

Composition with joint state. The restriction to subroutine-respecting protocols, made in Theorem 4, excludes the case of composition with joint state, namely in the case where parties in two or more protocol instances have access to the same instance of some subroutine program. We currently have two alternative methods to deal with this situation. A first method is to explicitly model the subroutine as an entity that interacts with multiple protocol instances (even arbitrary ones). This in turn requires working with a strong variant of UC security, called *generalized UC*, which allows capturing such subroutines and the protocols that use them. See details in [CDPW07].

A second option is to demonstrate that *all* the protocols that use the joint subroutine do so via an interface that satisfies a certain condition. Essentially, this condition requires that the interface looks like the interface of multiple independent instances of a simpler procedure. In this case, one can again demonstrate a security-preserving composition result similar to Theorem 4. See details in [CR03].

Some equivalent variants. Finally, we note that several variants of Definition 3 turn out to be equivalent to the present formulation. First, allowing the environment to output an arbitrarily long string, or alternatively restricting the environment to deterministic computation do not change the definition, nor does letting the simulator depend on the environment results in an equivalent definition.[4] Also, restricting the adversary $\mathcal{A}$ to only forward messages from the environment to the parties and back results in a definition that is equivalent to the present formulation. Similarly, restricting the adversary $\mathcal{S}$ to have only black-box access to $\mathcal{A}$ results in an equivalent definition.

4.3 Feasibility and relaxations

We very briefly survey the feasibility results regarding UC-realizing ideal functionalities. As we'll see, in spite of the apparent syntactic similarity with basic security (Section 4.4 in Part I), UC security is in general a considerably more restrictive notion. In particular, some far-reaching impossibility results exist. Consequently, several relaxations and work-arounds have been proposed. We will briefly survey these as well.

Encryption, signing, and secure communication. We start with some positive results. It turns out that for the basic tasks of encryption, digital signatures, and other tasks associated with secure communication, there are universally composable formulations that are realizable by known and natural protocols. In fact, in some cases the UC definitions are closely related, or even equivalent, to standard definitions (which use some special-purpose formulations).

Two salient examples are the ideal public-key encryption functionality, $\mathcal{F}_{\text{PKE}}$, and the ideal signature functionality, $\mathcal{F}_{\text{SIG}}$, which capture the basic requirements of encryption and signature in an abstract and unconditional way. UC-realizing $\mathcal{F}_{\text{PKE}}$ (for non-adaptive party corruptions) is essentially equivalent to the standard notion of security against chosen ciphertext attacks [DDN00, RS91]. UC-realizing $\mathcal{F}_{\text{SIG}}$ is essentially equivalent to the standard notion of existential unforgeability against chosen message attacks [GMRi88].

Another class of examples are functionalities related to the task of obtaining secure communication. These include the key-exchange functionality from Section 4.3 of Part I, as well as ideal functionalities capturing authenticated and secure communication sessions, entity authentication, and related tasks. All of these functionalities can be UC-realized by simple and known protocols. For instance, see the modeling of certified mail in [PSW00a] or secure channels in [PW01, CK02]. In addition, both the ISO 9798-3 key-exchange protocol and IKEv2 (the revised key exchange protocol of the IPSEC standard) UC-realize the ideal key-exchange functionality [CK02, CK02a].

General feasibility. Can the general feasibility results for basic security assuming authenticated communication (see Section 4.4 in Part I) be carried over to UC security? When the majority of the parties are honest (i.e., they are guaranteed to follow the protocol), the answer is positive. In fact, some known protocols for general secure function evaluation turn out to be universally composable. For instance, the

[4] We remark that in other formalisms, where entities are required to be polynomial in a global security parameter rather than in the length of local inputs, the last equivalence does not hold, see e.g. [HU05].

[BGW88] protocol (both with and without the simplification of [GRR98]), together with encrypting each message using non-committing encryption [CFGN96], is universally composable as long as less than a third of the parties are corrupted, and authenticated and synchronous communication is available. Using [RB89], any corrupted minority is tolerable. Asynchronous communication can be handled using the techniques of [BCG93, BKR94]. Note that here some of the participants may be "helpers" (e.g., dedicated servers) that have no local inputs or outputs; they only participate in order to let other parties obtain their outputs in a secure way.

However, things are different when honest majority of the parties is not guaranteed, and in particular in the case where only two parties participate in the protocol and either one of the parties may be corrupted. First, one of the most common proof-techniques for cryptographic protocols, namely black-box simulation with rewinding of the adversary, does not in general work in the present framework. The reason for that is that in the present framework the ideal adversary has to interact directly with the environment which cannot be "rewound". (Indeed, it can be argued that the meaningfulness of black-box simulation with rewinding in a concurrent execution setting is questionable.)

Furthermore, in the UC framework many interesting functionalities cannot be realized at all by plain protocols. (A *plain* protocol uses no ideal functionality other than the authenticated communication functionality.) For one, the ideal commitment functionality from Section 4.3 of Part I cannot be UC-realized by plain two-party protocols [CF01]. Similar impossibility results hold for the ideal coin tossing functionality, the ideal Zero-Knowledge functionality, and the ideal Oblivious Transfer functionality [C01]. These results extend to unrealizability by plain protocols of almost all "non-trivial" deterministic two-party functions and many probabilistic two-party functions [CKL03], and to impossibility of realizing *any* "ideal commitment functionality", namely any functionality that satisfies the basic correctness, binding and secrecy properties of commitment in a perfect way [DDMRS06]. These results apply also to multi-party extensions of these primitives, whenever the honest parties are not in majority.

Three main approaches for circumventing these impossibility results have been considered in the literature. The first approach is simply to try to formulate more *relaxed* ideal functionalities, that will be easier to realize, but will still capture the security requirements of the desired task. This is a task-specific and delicate endeavor. Some works that take this approach are [CK02, PS05]; a salient characteristic of these relaxations are that security is guaranteed only in a computational sense *even in the ideal process.*

A second approach is to assume that the parties have access to some trusted set-up. A third approach is to *relax* the UC-emulation requirement. These approaches are described in Sections 4.3.1 and 4.3.2, respectively.

4.3.1 Adding set-up assumptions

It turns out that general feasibility can be regained when some trusted set-up is assumed. One such trusted set-up assumption, called the key registration (KR) model, assumes that there exists a trusted "registration authority" where parties can register public keys associated with their identities, while demonstrating that they have access to the corresponding secret keys. (Alternatively, parties can let the authority choose public keys for them; here the corresponding secret keys need not be revealed, even to the "owners" of the public keys.) Then, parties can query the authority for a party identity and obtain the registered public key for that identity. Practically any ideal functionality can be UC-realized by interactive protocols in the key registration model, under standard computational hardness assumptions. Furthermore, the protocols remain secure even in the presence of arbitrary other protocols *that use the same public keys.*

Taking a short detour, it is interesting to compare this set-up assumption to the set-up assumptions needed for guaranteeing authenticated communication. To obtain authenticated communication (namely, to UC-realize an ideal functionality that provides an authenticated communication service), it is necessary

and sufficient to have access to an ideal functionality that allows parties to register public keys that will be associated with their identities, *without* having to disclose the secret keys to the registration authority. This set-up is structurally similar to the key registration set-up, except that the trust put in the registration authority is considerably milder.

An alternative set-up assumption, called the common random string (CRS) model, is that all parties have access to a string that is guaranteed to be taken from a predetermined distribution, typically the uniform distribution. Furthermore, it is assumed that the string was "ideally generated" in the sense that no set of participants have any "side information" on the common string (such as the preimage of the string according to some one-way function). This assumption is attractive in that it can be realized by physical processes that minimize the trust that participants need to put in external authorities. Also, it does not require parties to explicitly register before participating in the computation. However, here the general feasibility results are weaker, in the sense that the protocols are not (and, in fact, provably cannot be) shown secure in the presence of arbitrary other protocols that use the same common string. Instead, security is shown only when all protocols that use the common string do so using a very specific interface.

Yet another alternative set-up assumption, called the timing model, is of a somewhat different flavor: It assumes that there is a bound on the delay of messages delivered in the network, as well as on the mutual discrepancy in local time measurements, and that these bounds are known to all parties. Here too it is possible to realize any ideal functionality, under standard hardness assumptions [LPT04].

Historically, general feasibility results were first demonstrated in the CRS model [CLOS02]. The overall structure of that protocol is the same as in [GMW87], as sketched in Section 4.4 in Part I. The main difference is in the zero-knowledge and coin-tossing components, which are very different. In particular, the new components (based partly on the UC commitment protocol in [CF01]) allow for simulation "without rewinding", using the CRS set-up. Protocols in the KR model again use the same structure. For non-adaptive party corruptions, it was observed that the [CLOS02] protocols can be modified to work in the KR model [BCNP04]. For adaptive party corruptions some new protocols have been developed [CDPW07].

Can we characterize which functionalities are realizable without set-up? or only given authenticated communication? Alternatively, can we characterize the set-up functionalities that suffice for realizing a given task? Some limited answers to the former question, for the case of evaluating a pre-determined function of the parties' inputs, and for the case of functionalities aimed at guaranteeing secure commitment, are known [CKL03, DDMRS06]. Otherwise, these are interesting open questions.

4.3.2 Relaxing UC security

In light of the restrictiveness of UC-emulation, and in particular given the above impossibility results regarding realizing UC-realizing functionalities without initial set-up, it is natural to look for alternative notions of security, that will still provide some general security and composability guarantees while being easier to realize.

This question is highlighted by the fact that UC-emulation appears to be overly strong with respect to the notion of composable security (Definition 1). That is, Theorem 4 states that $\rho^{\pi/\phi}$ UC-emulates ρ, where Definition 1 only requires that $\rho^{\pi/\phi}$ emulates ρ according to the basic notion of emulation, namely Definition 2 in Part I. On the one hand, this extra strength is useful, in that it guarantees that security is preserved even after multiple applications of the universal composition operation. On the other hand, though, this extra strength raises the question of whether there is a less demanding variant of UC-emulation that would still satisfy Definition 1.

It turns out, however, that the answer to this question is negative. That is, it can be seen that *any* notion of emulation that satisfies Definition 1 with respect to any calling protocol ρ implies UC-emulation. That is:

Proposition 5 *Assume that protocol π emulates protocol ϕ with ρ-composable security for any subroutine-respecting protocol ρ. Then π UC-emulates ϕ.*

The idea here is that an arbitrary calling protocol ρ can essentially mimic any interactive environment $\mathcal{E}$, even in the basic setting where the external environment cannot interact with the adversary during the execution. This holds even though ρ^π is only required to emulate ρ^ϕ according to the basic notion, since the instructions of ρ can require the adversary to provide "on-line" information in the same way that $\mathcal{E}$ expects to have in the UC modeling. We omit further details.

Proposition 5 can be interpreted as stating that UC-security is in some sense a "minimal" requirement that guarantees both composability and basic security. It also means that the extra strength in the conclusion of Theorem 4 comes without any additional requirements from the protocol. (Some closely related results appear in [L03, L04].)

Still, some relaxed variants of UC-emulation have been proposed and shown to be preserved under universal composition with arbitrary protocols [PS04, BS05, MMY06]. By Proposition 5, these variants necessarily provide security guarantees that are weaker than basic security. Still, the provided guarantees are often meaningful. In addition, it was demonstrated that these notions allow realizing any ideal functionality given only authenticated communication, under general (but stronger than usual) hardness assumptions.

Essentially, the way in which these notions weaken the security requirement is by allowing the "simulator" to run in super-polynomial time T. This means that meaningful security is guaranteed only when the following two conditions are met. First, the ideal functionality should be such that security is guaranteed even against adversaries running in time T. This condition is met by most common formulations of ideal functionalities; in fact, most common formulations provide "perfect" security, even against computationally unbounded ideal-model adversaries.

The second condition is a bit more subtle: Recall that the definition only guarantees that the environment, or the calling protocol, cannot tell whether it is interacting with the emulating protocol π and adversary $\mathcal{A}$ (which may be PT), or with the emulated protocol and adversary $\mathcal{S}$, which may run in time T. Thus, security is meaningful only when the the *calling protocol itself* withstands adversaries that run in time T. To exemplify this point, we note that it is possible to construct protocols that are secure according to this notion, and yet completely "break down" under self-composition of only two instances.

5 Conclusion

This tutorial addressed the challenges associated with rigorously modeling cryptographic protocols and capturing their security properties. Particular stress was put on guaranteeing security in settings where protocols are *composed* with each other in a number of ways. We have reviewed a general definitional approach, the *trusted party paradigm.* We saw two formalizations of this approach: A basic formalization, that is easier to satisfy but provides only limited secure composability guarantees, and a more advanced formalization that is considerably more restrictive in general, but provides very strong secure composability guarantees.

When looking back at the covered material, one thing becomes very clear: It is far from obvious what is "the right" way to capture and formalize security properties of cryptographic protocols. In fact, there probably is no single good way to do so, and different formalisms have incomparable strengths. Furthermore, seemingly small differences in the formalisms result in drastic differences — both in the meaningfulness (e.g. in the behavior under protocol composition), and also in the restrictiveness, namely in the ability to assert security of natural protocols.

One consequence of this fact is that finding viable notions of security for cryptographic protocols re-

mains an intriguing and lively research area. Another consequence is that appropriately formulating the security requirements of a given cryptographic task can be a delicate challenge in itself. In fact, this is often the "hard part" of the security analysis, more so than actually asserting that a given protocol satisfies the formulated property in the devised model.

Acknowledgments. My thinking and understanding of cryptographic protocols has been shaped over the years by discussions with many insightful researchers, too numerous to mention here. I thank you all. Oded Goldreich and Hugo Krawczyk were particularly influential, with often conflicting (complementary?) views of the field. I'm also grateful to the editor, Sergio Rajsbaum, for his effective blend of flexibility and persistence and to Shai Halevi, Ralf Küsters and Birgit Pfitzmann for helpful remarks.

References

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM conference on computer and communications security (CCS)*, 2003. Extended version at the eprint archive, http://eprint.iacr.org/2003/015/.

[BPW04] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *1st Theory of Cryptography Conference (TCC)*, LNCS 2951 pp. 336–354, Feb. 2004.

[B+05] B. Barak, R. Canetti, Y. Lindell, R. Pass and T. Rabin. Secure Computation Without Authentication. In Crypto'05, 2005.

[BCNP04] B. Barak, R. Canetti, J. B. Nielsen, R. Pass. Universally Composable Protocols with Relaxed Set-Up Assumptions. 45th FOCS, pp. 186–195. 2004.

[BS05] B. Barak and A. Sahai, How To Play Almost Any Mental Game Over the Net - Concurrent Composition via Super-Polynomial Simulation. 46th FOCS, 2005.

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

[BR93] M. Bellare and P. Rogaway. Entity authentication and key distribution. *CRYPTO'93*, LNCS. 773, pp. 232-249, 1994.

[BCG93] M. Ben-Or, R. Canetti and O. Goldreich. Asynchronous Secure Computation. 25th Symposium on Theory of Computing (STOC), 1993, pp. 52-61. Longer version appears in TR #755, CS dept., Technion, 1992.

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

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

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

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

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

[C01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Extended abstract in *42nd FOCS*, 2001. A revised version (2005) is available at IACR Eprint Archive, eprint.iacr.org/2000/067/ and at the ECCC archive, http://eccc.uni-trier.de/eccc-reports/2001/TR01-016/.

[C06] R. Canetti. Security and Composition of Cryptographic Protocols: A Tutorial, (Part I). in Distributed Computing Column of ACM SIGACT News, Vol. 37, Num. 3, (Whole Number 140), Sept. 2006.

[C+06] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-Structured Probabilistic I/O Automata. In Workshop on discrete event systems (WODES), 2006.

[C+06a] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Time-Bounded Task-PIOAs: A Framework for Analyzing Security Protocols. In 20th symposium on distributed computing (DISC), 2006.

[CDPW07] R. Canetti, Y. Dodis, R. Pass and S. Walfish. Universally Composable Security with Pre-Existing Setup. *4th theory of Cryptology Conference (TCC)*, 2007.

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

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

[CH04] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Cryptographic Protocols (The case of encryption-based mutual authentication and key-exchange). Eprint archive, http://eprint.iacr.org/2004/334. Extended Abstract at 3rd TCC, 2006.

[CK02] R. Canetti and H. Krawczyk. Universally Composable Key Exchange and Secure Channels . Eurocrypt '02, pages 337–351, 2002. LNCS No. 2332. Extended version at http://eprint.iacr.org/2002/059.

[CK02a] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-based Key-Exchange Protocol. Crypto '02, 2002. Extended version at http://eprint.iacr.org/2002/120.

[CKL03] R. Canetti, E. Kushilevitz, Y. Lindell. On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions. EUROCRYPT 2003, pp. 68–86, 2003. Extended version at the eprint archive, eprint.iacr.org/2004/116.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai. Universally composable two-party and multi-party secure computation. 34th STOC, pp. 494–503, 2002.

[CR03] R. Canetti and T. Rabin. Universal Composition with Joint State. Crypto'03, 2003.

[CKLP06] L. Cheung, D. Kaynar, N. Lynch, O. Pereira. Compositional Security for Task-PIOAs. Manuscript, 2006.

[DDMRS06] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan and A. Scedrov. Games and the Impossibility of Realizable Ideal Functionality. 3rd theory of Cryptology Conference (TCC), 2006.

[DKMR05] A. Datta, R. Küsters, J. C. Mitchell and A. Ramanathan. On the Relationships between Notions of Simulation-based Security. 2nd theory of Cryptology Conference (TCC), 2005.

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

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

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

[DLMS99] N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov. Undecidability of bounded security protocols. Workshop on Formal Methods and Security Protocols (FMSP), 1999.:w

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

[EG82] S. Even and Oded Goldreich. On the Security of Multi-Party Ping-Pong Protocols. 24th FOCS, 1983.

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

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

[G01] O. Goldreich. Foundations of Cryptography. Cambridge Press, Vol 1 (2001) and Vol 2 (2004).

[GK89] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. SIAM. J. Computing, Vol. 25, No. 1, 1996.

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

[GO94] O. Goldreich and Y. Oren. Definitions and properties of Zero-Knowledge proof systems. J. Cryptology, Vol. 7, No. 1, 1994, pp. 1–32.

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

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

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

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

[H85] C. A. R. Hoare. Communicating Sequential Processes. International Series in Computer Science, Prentice Hall, 1985.

[HU05] D. Hofheinz and D. Unruh. Comparing Two Notions of Simulatability. *2nd theory of Cryptology Conference (TCC)*, pp. 86-103, 2005.

[IPSEC] The IPSec working group of the IETF. See http://www.ietf.org/html.charters/ipsec-charter.html

[KLR06] E. Kushilevitz, Y. Lindell and T. Rabin. Information-Theoretically Secure Protocols and Security Under Composition. 38th STOC, pages 109-118, 2006.

[K06] R. Küsters. Simulation based security with inexhaustible interactive Turing machines. 19th CSFW, 2006.

[L03] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. 43rd FOCS, pp. 394–403. 2003.

[L04] Y. Lindell. Lower Bounds for Concurrent Self Composition. 1st Theory of Cryptology Conference (TCC), pp. 203–222. 2004.

[LLR02] Y. Lindell, A. Lysyanskaya and T. Rabin. On the composition of authenticated Byzantine agreement. 34th STOC, 2002.

[LPT04] Y. Lindell, M. Prabhakaran, Y. Tauman. Concurrent General Composition of Secure Protocols in the Timing Model. Manuscript, 2004.

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

[LT89] N. Lynch and M. R. Tuttle. An introduction to input/output automata. CWIQuarterly, 2(3):219-246, September 1989.

[LSV03] N. Lynch, R. Segala and F. Vaandrager. Compositionality for Probabilistic Automata. 14th CONCUR, LNCS vol. 2761, pages 208-221, 2003. Fuller version appears in MIT Technical Report MIT-LCS-TR-907.

[MMY06] T. Malkin, R. Moriarty and N. Yakovenko. Generalized Environmental Security from Number Theoretic Assumptions. *3rd Theory of Cryptology Conference (TCC)*, 2006, pp. 343-359.

[MMS03] P. Mateus, J. C. Mitchell and A. Scedrov. Composition of Cryptographic Protocols in a Probabilistic Polynomial-Time Process Calculus. 14th CONCUR, pp. 323-345. 2003.

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

[M89] R. Milner. Communication and Concurrency. Prentice Hall, 1989.

[M99] R. Milner. Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, 1999.

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

[MRST06] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. Theor. Comput. Sci. 353(1-3): 118-164 (2006). Preliminary version in LICS'01.

[P04] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. 36th STOC, pp. 232–241. 2004.

[P91] T. P. Pedersen: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. CRYPTO 1991: 129-140

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

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

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

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

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

[PRS02] M. Prabhakaran, A. Rosen, A. Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. 43rd FOCS, 2002: 366-375

[PS04] M. Prabhakaran, A. Sahai. New notions of security: achieving universal composability without trusted setup. 36th STOC, pp. 242–251. 2004.

[PS05] M. Prabhakaran, A. Sahai. Relaxing Environmental Security: Monitored Functionalities and Client-Server Computation. *2nd Theory of Cryptology Conference (TCC)*, 2005.

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

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

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

[SB+06] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann and M. Waidner. Cryptographically Sound Theorem Proving. 19th Computer Security Foundations Workshop (CSFW), 2006.

[Y82A] A. Yao. Protocols for Secure Computation. In 23rd Annual Symp. on Foundations of Computer Science (FOCS), pages 160–164. 1982.

[Y86] A. Yao, How to generate and exchange secrets, In 27th Annual Symp. on Foundations of Computer Science (FOCS), pages 162–167. 1986.

A Trusted-party based security: A mini survey

This section briefly surveys some works that are directly relevant to the development of the trusted-party paradigm as a method for defining security of protocols. (Indeed, this is only a fraction of the body of work on modeling cryptographic protocols and asserting security properties.) More detailed surveys on this topic can be found in [C00, C01]. Also, some of these works have already been mentioned earlier and are not re-addressed here.

Two works that essentially "laid out the field" of general security definitions for cryptographic protocols are the work of Yao [Y82A], which expressed for the first time the need for a general "unified" framework for expressing the security requirements of cryptographic tasks and for analyzing cryptographic protocols; and the work of Goldreich, Micali and Wigderson [GMW87], which put forth the approach of defining security via comparison with an ideal process involving a trusted party (albeit in a very informal way).

The first rigorous definitional framework is due to Goldwasser and Levin [GL90], and was followed shortly by the frameworks of Micali and Rogaway [MR91] and Beaver [B91]. In particular, the notion of "reducibility" in [MR91] directly underlies the notion of protocol composition in many subsequent works, including the notion of universal composition as descried here. Beaver's framework was the first to directly formalize the idea of comparing a run of a protocol to an ideal process. Still, the [MR91, B91] formalisms only address security in restricted settings; in particular, they do not deal with computational issues.

Canetti [C95] provides the first ideal-process based definition of computational security against resource bounded adversaries. [C00] strengthens the framework of [C95] to handle secure composition. In particular, security of protocols in that framework is shown to be preserved under non-concurrent universal composition. This work also contains sketches on how to strengthen the definition to support concurrent composition. A closely related formulation appears in [G01].

The framework of Hirt and Maurer [HM00] is the first to rigorously address the case of reactive functionalities. Dodis and Micali [DM00] build on the definition of Micali and Rogaway [MR91] for unconditionally secure function evaluation, and prove that their notion of security is preserved under a general concurrent composition operation similar to universal composition. However, their definition involve notions that make sense only in settings where the communication is ideally private; thus this definition does not apply to the common setting where the adversary has access to the communication between honest parties.

All the work mentioned above assumes *synchronous* communication. Security for asynchronous communication networks was first considered in [BCG93, C95].

The framework of Pfitzmann, Schunter and Waidner [PSW00, PW00] is the first to rigorously address *concurrent* universal composition in a computational setting. (This work is based on [PW94], which describes a basic system model and sketches a notion of security.) They define security for reactive functionalities in a synchronous setting and prove that security is preserved when a *single* instance of a subroutine protocol is composed concurrently with the calling protocol. An extension of the [PSW00, PW00] framework and notion (called *reactive simulatability*) to asynchronous networks appears in [PW01].

Universal composability in its full generality was first considered in [C01], which addressed the case of unbounded number of concurrently composed protocols. This work also demonstrated how the security requirements of a number of commonplace and seemingly unrelated cryptographic tasks can be captured via the trusted-party paradigm in the devised model.

A process calculus for representing probabilistic polynomial time interacting processes is developed in [LMMS98, MRST06]. In [MMS03] the notion of protocol emulation and realizing an ideal functionality is formalized in this model, and shown to be preserved under universal composition with any calling protocol. Other models that define emulation-based security include [DKMR05, K06].

At very high level, the notions of security in [PW01, C01, MMS03, DKMR05, K06] are similar. However,

the underlying system models differ in a number of respects, which significantly affect the expressibility and generality of the respective models (namely, the range of real-life situations and concerns that can be captured by the respective formalisms), as well as the simplicity and ease of use. In addition, the models provide different degrees of abstraction and different tools for arguing about security properties. We leave a more detailed comparison out of scope.

Finally, we note that the above notions of security do not leave room for non-determinism in protocol description and run-time scheduling. This is a natural choice, since non-determinism that is resolved arbitrarily at run-time seems inherently incompatible with security against computationally bounded adversaries. However, such modeling does not allow utilizing the traditional analytical advantages of non-determinism in modeling of distributed protocols. First steps towards incorporating in the model non-determinism that's resolved at runtime are taken in [C+06, C+06a]; the main idea here is to allow some parts of the protocol execution to be determined arbitrarily *after* all the algorithmic components are fixed.