Games and the Impossibility of Realizable Ideal Functionality

Anupam Datta[1], Ante Derek[1], John C. Mitchell[1], Ajith Ramanathan[1], and Andre Scedrov[2]

[1] Stanford University {danupam,aderek,jcm,ajith}@cs.stanford.edu
[2] University of Pennsylvania scedrov@math.upenn.edu

Abstract. A cryptographic primitive or a security mechanism can be specified in a variety of ways, such as a condition involving a game against an attacker, construction of an ideal functionality, or a list of properties that must hold in the face of attack. While game conditions are widely used, an ideal functionality is appealing because a mechanism that is indistinguishable from an ideal functionality is therefore guaranteed secure in any larger system that uses it. We relate ideal functionalities to games by defining the *set* of ideal functionalities associated with a game condition and show that under this definition, which reflects accepted use and known examples, bit commitment, a form of group signatures, and some other cryptographic concepts do not have any realizable ideal functionality.

1 Introduction

Many security conditions about cryptographic primitives are expressed using a form of game. For example, the condition that an encryption scheme is semantically secure against chosen ciphertext attack (IND-CCA2) [1] may be expressed naturally by saying that no adversary has better than negligible probability to win a certain game against a challenger. In this definition, the game itself clearly identifies the information and actions available to the adversary, and the condition required to win the game identifies the properties that must be preserved in the face of attack. Another way of specifying security properties uses ideal functionalities [2–5]. In this approach, usually referred to as Universal Composability [3] (UC) or Reactive Simulatability [6] an idealized way of achieving some goal is presented, possibly using mechanisms such as authenticated channels and trusted third parties that are not basic primitives in practice. An implementation is then considered secure if no feasible attacker can distinguish the implementation from the ideal functionality, in any environment. An advantage of this approach is that indistinguishability from an ideal functionality leads to composable notions of security [3, 5, 7]. In contrast, if a mechanism satisfies a game condition, there is no guarantee regarding how the mechanism will respond to interactions that do not arise in the specified game.

In this paper, we develop a framework for comparing game specifications and ideal functionalities, and prove some negative results about the existence of ideal functionalities in certain settings. While most known primitives have game-based definitions (see, e.g., [8]), it has proven difficult to develop useful ideal functionalities for some natural primitives. Some interesting issues are explored in [9, 10], which describe a series of efforts to develop a suitable ideal functionality for digital signatures. In brief, there is a widely accepted game condition for digital signatures, existential unforgeability against chosen message attacks, formulated in [11]. However, there are many possible ideal functionalities that are consistent with this game condition. For example, a functionality could either explicitly disclose information about messages that were signed in the past, or not disclose this information. More generally, given a game condition, it is often feasible to formulate various functionalities that satisfy the game condition yet reveal varying kinds of "harmless" information that does not seem relevant to the goals of the mechanism.

If we have a game or set of games that define a concept like secure encryption, digital signature, or bit-commitment, then we would like to identify precisely the set of possible ideal functionalities associated with each game condition. Since an ideal functionality is intended to be evidently secure by construction, we propose that an *ideal* functionality must satisfy the given game condition on information-theoretic grounds, rather than as a result of computational complexity arguments. Applied to encryption, for example, this means that an ideal functionality for encryption must not provide *any* information about bits of the plaintext to the adversary. Our definition of *ideal functionality* for a set of game conditions is consistent with all examples we have found in the literature, and reflects the useful idea that it should be easier to reason about systems that use an ideal functionality than about systems that use a real protocol. Using our definition, we show that while bit-commitment may be specified using games, there is no realizable ideal functionality for bit-commitment. This may be seen as a negative result about specification using ideal functionality, since there are constructions of bit-commitment protocols that are provably correct under modest cryptographic assumptions (see, e.g., [12]). We also show that there is no realizable ideal functionality for other reasonable and implementable cryptographic primitives, including a form of group signatures and a form of symmetric encryption with integrity guarantees, under certain conditions that allow the encryption key to be revealed after it is used.

The intuition behind our impossibility result is relatively simple. Illustrated using bit-commitment, a good commitment scheme must have two properties: the commitment token must not reveal any information about the chosen bit, while subsequent decommitment must reveal a verifiable relationship between the chosen bit and the commitment token. These are contradictory requirements because the first condition suggests that tokens must be chosen randomly, while the second implies that they are not. Similar "decommitment" issues arise in symmetric encryption or keyed hash, if the encryption key is revealed after some messages using the key have been sent on the visible network. At a more technical level, our proof by contradiction works by showing that if there was a realization of the ideal functionality for bit-commitment, it could be transformed into a protocol for bit-commitment that achieves perfect hiding and binding without using a trusted third party. However, it is well known that such a protocol does not exist [12]. While impossibility results for group signatures and symmetric encryption could be proved by instantiating the general proof method, we present simpler proofs by reducing bit-commitment to these primitives.

In a previous study of ideal functionality for bit commitment, Canetti and Fischlin show that a particular ideal functionality for bit-commitment is not realized by any real protocol [13]. In related work, Canetti [3] shows that particular functionalities for ideal coin tossing, zero-knowledge, and oblivious transfer are not realizable. Canetti et al [14] show that a class of specific functionalities for secure multi-party computation are not realizable, while Canetti and Krawczyk [15] compare indistinguishability-based and simulatability-based definitions of security in the context of key-exchange protocols. Our results are more general since we prove that, given a *game* definition of a primitive, there is *no* realizable ideal functionality associated with that game condition. In addition, our proof is different in that it provides a reduction to a previous negative result independent of universal composability [12], and appears to apply immediately to many primitives. A related issue is the choice of so-called "setup assumptions," such as public-key infrastructure, and common reference string. Our negative results hold under some setup assumptions, such as the absence of shared private information, or the presence of a trusted certificate authority (or PKI), and fail for other setup assumptions, such as the assumption of a common reference string. This is expected, since [13] construct a realizable ideal functionality in the common reference string model. We have yet to characterize precisely the set of possible setup assumptions under which our negative results hold.

While our general proof could be carried out using a number of computational models, we adopt a setting based on a form of process calculus. One advantage of this setting over interacting Turing machines [3, 11, 12] is a straightforward way of modularizing games that use a functionality. This is useful for defining primitives that are protocols, as opposed to local functions, by games. In principle, some version of our proof could be carried out using some version of Turing machines, augmented with separate function-call-and-return tapes for interacting with some form of oracle that performs public communication visible to the adversary.

Preliminary definitions are presented in Section 2, followed by definitions of bit-commitment functionalities and the main impossibility proof in Section 3. Reductions from other primitives are given in Section 4, with concluding remarks in Section 5.

2 Preliminaries

2.1 Probabilistic Process Calculus

Process calculus is a standard framework for studying concurrency [16, 17] that has proved useful for reasoning about security protocols [2, 18]. This is more of a "software" model than a "machine" model, since process calculus expressions are a form of program defining a concurrent system. Two main organizing ideas in process calculus are actions and channels. *Actions* occur on channels and are used to model communication flows. *Channels* provide an abstraction of the communication medium. In practice, channels might represent an IP address and port number in distributed computing, or a region of shared memory in a parallel processor.

A probabilistic polynomial-time process calculus (PPC) for security protocols is developed in [19–21] and updated in more recent papers [18, 22]. The syntax consists of a set of *terms* that represent local sequential probabilistic polynomial-time computation and do not perform any communication with other processes, process *expressions* that can communicate with other processes, and *channels* that are used for communication. Terms contain variables that receive values over channels. There is also a special variable η called the *security parameter*. Each expression defines a set of *processes*, one for each choice of value for the security parameter. Each channel name has a bandwidth polynomial in the security parameter associated with it. The bandwidth ensures that no message gets too large and, thus, ensures that any expression can be evaluated in time polynomial in the security parameter.

Syntax of PPC: Expressions of PPC are constructed from the following grammar.

$$\mathcal{P} ::= \oslash \;\Big|\; \nu(c)\mathcal{P} \;\Big|\; \mathsf{in}(c,x).(\mathcal{P}) \;\Big|\; \mathsf{out}(c,\mathrm{T}).(\mathcal{P}) \;\Big|\; [\mathrm{T}].(\mathcal{P}) \;\Big|\; (\mathcal{P} \mid \mathcal{P}) \;\Big|\; !_{q(\eta)}(\mathcal{P})$$

Intuitively, $\oslash$ is the *empty process* taking no action. A process $\mathsf{in}(c,x).\mathcal{P}$ with an *input* operator waits until it receives a value for input variable x on the channel c and then proceeds with process $\mathcal{P}$. Similarly, an *output* $\mathsf{out}(c,\mathrm{T}).\mathcal{P}$ transmits that value of the term T on the channel c and then proceeds with $\mathcal{P}$. Channel names that appear in an input or an output operation can be either public or private, with a channel being private if it is bound by the *private-binding* operator, ν and public otherwise. Actions on a private channel bound by a ν are not observable outside the scope of the ν operator. Hence private channels can be used to provide a form of secure communication. The *match* operator [T], a form of "if", executes the expression that follows it iff T evaluates to 1. The *parallel composition* operator, $|$, applied to two expressions allows them to evaluate concurrently, possibly communicating over any shared channels. The *bounded replication* operator has bound determined by the polynomial q affixed as a subscript. The expression $!_{q(\eta)}(\mathcal{P})$ is expanded to the $q(\eta)$-fold parallel composition $\mathcal{P} \mid \cdots \mid \mathcal{P}$ before evaluation. There is also a syntactic notion of context in PPC. A *context* $\mathcal{C}[\,\cdot\,]$ is an expression with a hole $[\,\cdot\,]$ such that we can substitute any expression into the hole and obtain a well-formed expression. Contexts may be used to represent the environment or adversary that interacts with a protocol or process.

Evaluating PPC expressions: To evaluate an expression in PPC we choose a probabilistic scheduler that selects communication steps. We then evaluate every term and match that is not in the scope of an input expression. When we can no longer evaluate terms and matches, we select a pair of input and output expressions on the same channel according to the scheduler, erase the output expression and substitute the value transmitted by the output (truncated suitably by the bandwidth of the channel) for the variable bound by the input. This procedure is repeated until no communication steps are possible. Further discussion, and explanation of a number of issues related to probabilistic scheduling, are explained in [18, 22–24].

Equivalence relations over PPC: Two equivalence relations over PPC will prove useful for studying security issues. The first relation, *computational observational equivalence*, written $\cong$, relates two expressions just when, in any context, the difference between the distributions they induce on observable behavior (messages over public channels) is negligible in the security parameter η. Formally $\mathcal{P} \cong \mathcal{Q}$ just when $\forall$ contexts $\mathcal{C}[\ \cdot\ ].\forall$ observables o:

$$\text{Prob}\left[\mathcal{C}[\mathcal{P}] \text{ produces } o\right] - \text{Prob}\left[\mathcal{C}[\mathcal{Q}] \text{ produces } o\right] \text{ is negligible in } \eta$$

Since the evaluation of all expressions and contexts in PPC are guaranteed to terminate in polynomial-time, $\cong$ is a natural way to state that two expressions are computationally indistinguishable to a poly-time attacker. The second relation, *information-theoretic observational equivalence*, written $=$, relates two expressions just when they induce exactly the same distribution on observable behavior in all contexts. Formally $\mathcal{P} = \mathcal{Q}$ just when $\forall$ contexts $\mathcal{C}[\ \cdot\ ].\forall$ observables o:

$$\text{Prob}\left[\mathcal{C}[\mathcal{P}] \text{ produces } o\right] - \text{Prob}\left[\mathcal{C}[\mathcal{Q}] \text{ produces } o\right] = 0$$

As a consequence, we can use $=$ to state that two expressions are indistinguishable even to unbounded attackers.

2.2 Function calls and returns

Process calculus allows processes to be programmed in a modular way, with one process relying on another for certain computations or actions. For example, one process P might wish to send a number bit-by-bit on a channel d. This can be done by writing another process Q that handles all the communication on channel d for P. This process Q receives a number n on some channel c used only for communication between P and Q, and then sends the bits of n on a channel d as required. If P wants a return value, such as notification that Q has finished sending the message, then P can execute an input action on channel c immediately after sending the number n to Q. This pattern of sends and receives essentially works like an ordinary remote procedure call and return. If the channel c is private, we can think of this as a remote procedure call between one process and another on the same processor, through a loopback interface, or a remote procedure call between two processors behind a firewall that makes LAN traffic invisible to an external attacker.

We will refer to the pattern of sends and receive just described for processes P and Q as a *function call and return.* Function calls and returns turn out to be a very useful concept in structuring games that specify properties of cryptographic primitives. To give a relatively concise notation, we will write $\mathsf{Call}^{\eta}(\langle \mathtt{params} \rangle, \mathbb{C})$ $\mathsf{returns}$ $\langle \mathtt{vars} \rangle.\mathcal{P}$ for a call that sends (outputs) parameters $\mathtt{params}$ on calling channels $\mathbb{C}$, and then waits to receive (input) return values $\langle \mathtt{vars} \rangle$ before executing process $\mathcal{P}$. To emphasize that a function call and return hides the structure of Q from the calling process P, we sometimes refer to this as a *black-box call.* Since PPC provides private channels, a function call and return will always be done on a private channel to avoid exposing the parameters or return values to an adversary.

For every function call and return to proceed, there must be a process that waiting to receive the call and then send a return value. Rather than write out all the input and output actions associated with responding to a remote procedure call, we will simply write $\mathsf{Impl}[\mathbb{C}, \mathbb{D}]$ for a process that responds to blackbox calls on channels $\mathbb{C}$, possibly using channels in $\mathbb{D}$ for some other purpose. For example, the process Q described above has the form $\mathsf{Impl}[c, d]$. since it receives function calls on channel c and performs public communication on channel d.

2.3 Interfaces and cryptographic primitives

In this paper, a *cryptographic primitive* is defined by an interface and a set of required security or correctness conditions that are expressible using the interface. The *interface* is the set of actions defined and applicable to the primitive, expressed as a set of function calls and returns. For example, the interface to an encryption primitive consists of calls to three probabilistic functions: key-generation, encryption, and decryption. A correctness condition for encryption is that the decryption of an encryption under the correct key returns the message encrypted. A semantically-secure encryption primitive must also satisfy a security condition stating that no probabilistic polynomial-time adversary can win a game that involves guessing which of two messages has been encrypted.

A *protocol* for a primitive is a process that responds to a set of function calls and supplies the associated returns, without using any additional private communication. For example, RSA can be formulated as an encryption protocol that implements key-generation, encryption, and decryption. A *functionality* for a primitive similarly supports the given interface, but may use additional private communication (such as used for a trusted third party; see Section 3.3). These restrictions on private communication are meant to prevent abusing the security associated with private channels, which are not a realistic primitive on the open public network. However, there are no restrictions on the way a functionality can communicate or reveal information to the adversary. For example, a functionality for signatures [10] could let the attacker choose the bitstrings for signatures.

An *ideal functionality* for an interface and a set of game conditions is a functionality that satisfies the correctness conditions with high probability and satisfies the security conditions in an information-theoretic way (i.e., against an unbounded adversary).

2.4 Universal Composability

Universal composability [3, 13, 14, 25, 26] involves a protocol to be evaluated, an ideal functionality, two adversaries, and an environment. The protocol realizes the ideal functionality if, for every attack on the protocol, there exists an attack on the ideal functionality, such that the observable behavior of the protocol under attack is the same as the observable behavior of the idealized functionality under attack. Each set of observations is performed by the same environment. The intuition here is that the ideal functionality 'obviously' possess a desired security property, possibly because the ideal functionality is constructed using a central authority, trusted third party, or private channels. Therefore, if a protocol is indistinguishable from an ideal functionality, the protocol must have the desired security property. In previous work, that which makes an ideal functionality "ideal" appears not to have been characterized precisely.

Universal composability can be expressed as a relation in process calculus [23, 24]. To give a form appropriate for the present paper, let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ be n principals. We will assume that for some k, every principal $\mathcal{P}_i$ $(i > k)$ is in collusion with the adversary. Given an expression $\mathcal{P}$, we will write $\mathcal{P}[\mathbb{C}]$ to denote an instance of P running over the channels in $\mathbb{C}$. We say that an implementation Impl *securely realizes* a functionality $\mathcal{F}$ just when for any real world adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$:

$$\begin{aligned}
&\nu(\mathbb{C}_1, \ldots, \mathbb{C}_k)(\mathcal{P}_1[\mathbb{C}_1, \mathbb{D}] \mid \cdots \mid \mathcal{P}_n[\mathbb{C}_n, \mathbb{D}] \mid \mathsf{Impl}[\mathbb{C}_1, \mathbb{D}] \mid \cdots \mid \mathsf{Impl}[\mathbb{C}_k, \mathbb{D}]) \mid \\
&\mathcal{A}[\mathbb{C}_{k+1}, \ldots, \mathbb{C}_n, \mathbb{D}] \mid \mathcal{E} \\
\cong\ &\nu(\mathbb{C}_1, \ldots, \mathbb{C}_k)(\mathcal{P}_1[\mathbb{C}_1, \mathbb{D}] \mid \cdots \mid \mathcal{P}_n[\mathbb{C}_n, \mathbb{D}] \mid \mathcal{F}[\mathbb{C}_1, \ldots, \mathbb{C}_k, \mathbb{D}]) \mid \\
&\mathcal{S}[\mathbb{C}_{k+1}, \ldots, \mathbb{C}_n, \mathbb{D}] \mid \mathcal{E}
\end{aligned}$$

Here the first[3] k principals are assumed to be honest, and the remainder are assumed to be dishonest and acting in collusion with the adversary. To prevent the adversary/simulator from unfairly interfering with communications between the honest principals and the implementations (real or ideal), we make the links between the honest principals and the implementations private. Specifically, participant $\mathcal{P}_i$ uses private channels $\mathbb{C}_i$ to communicate with the implementation (real or ideal). The set of network channels $\mathbb{D}$ is used for communication between different participants. Both the adversary and the simulator have access to these channels.

Secure realisability requires that if we replace the real implementations Impl with an ideal implementation $\mathcal{F}$ (the functionality), there exists a simulator (that can interact with $\mathcal{F}$) which makes the ideal and real configurations indistinguishable. Another way to state this is that every real attack can be translated, using the simulator, into an attack on the functionality. We note that the principals that act in collusion with the attacker execute arbitrary programs and, in the ideal world, interact directly with the simulator (which mounts the ideal attack). Example configurations with two honest participants $\mathcal{I}$ and $\mathcal{R}$ are given in Figure 1.

[3] Since parallel composition is associative, the order in which we write the processes does not matter, and we may assume without loss of generality that the k honest principals occur first in the list.

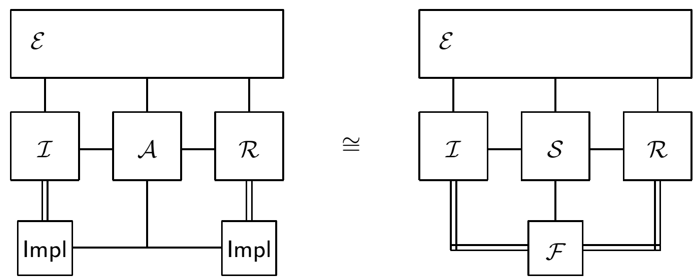

Fig. 1. Real and ideal configurations with two honest participants

3 Functionalities for Bipartite Bit-Commitment

A bipartite bit-commitment protocol allows a principal A to commit on a bit b to the principal B. However, B gains no information about the bit b until A later opens the commitment. We therefore formulate bit-commitment using four function calls, one call for each principal in each phase of the commitment. After defining the interface for bipartite bit-commitment, we define the game conditions for bit commitment and prove that no ideal functionality for these game conditions is realizable. We stress that the game conditions for bit-commitment as formulated in this paper are equivalent to standard security notions [12, 27], and that they can be realized using standard cryptographic assumptions such as the existence of pseudorandom functions [27].

3.1 Commitment interface

A bipartite bit-commitment scheme provides four function calls:

$$\begin{array}{ll} \mathsf{SendCommit}^{\eta}(b, \mathbb{C}) \text{ returns } \langle \sigma \rangle & \mathsf{GetCommit}^{\eta}(\mathbb{C}) \text{ returns } \langle \sigma \rangle \\ \mathsf{Open}^{\eta}(\sigma, \mathbb{C}) \text{ returns } \emptyset & \mathsf{Verify}^{\eta}(\sigma, \mathbb{C}) \text{ returns } \langle \mathbf{r} \rangle \end{array}$$

The initiator A commits to a bit using the call $\mathsf{SendCommit}^{\eta}(b, \mathbb{C})$ returns $\langle \sigma \rangle$, which communicates the commitment value over the channels in $\mathbb{C}$. Some state information σ is generated that can, amongst other things, be used to differentiate between different commitments and is needed to open the corresponding commitment. A responder B may receive a commitment from A by executing a call $\mathsf{GetCommit}^{\eta}(\mathbb{C})$ returns $\langle \sigma \rangle$ over the channels in $\mathbb{C}$, which may also returns some state information σ.

In the decommitment phase, the initiator A may open the commitment using the function call $\mathsf{Open}^{\eta}(\sigma, \mathbb{C})$ returns $\emptyset$, which uses the state information from the initial call to indicate which commitment is to be opened. The responder B can then verify the committed value by making the call $\mathsf{Verify}^{\eta}(\sigma, \mathbb{C})$ returns $\langle \mathbf{r} \rangle$. If verification succeeds, $\mathbf{r}$ contains the value of the committed bit. Otherwise, $\mathbf{r}$ is a symbol $\perp$ indicating failure.

3.2 Commitment correctness and security conditions

There are three conditions on bit-commitment [12,27] — correctness, hiding, and binding. After explaining each condition, we show that each can be stated as an equivalence. The equivalences are written using $\cong$, which give the game condition required of any implementation. With $\cong$ replaced by $=$, the same equivalences can be used to state the information-theoretic properties required for an ideal functionality. More precisely, an *ideal functionality for bipartite bit-commitment* is an implementation for the four function calls listed in the interface above such that the correctness property below is satisfied with high probability, and the hiding and binding properties of bipartite bit-commitment below are satisfied with an information-theoretic equivalence. It is easy to verify that the concrete functionality considered in [13] is an instance of the ideal functionality for bipartite bit-commitment.

Given a game condition, there is a canonical way of writing it as an indistinguishability between expressions. The basic idea is that, since $\cong$ quantifies over all contexts, any successful attack on the game condition can be translated into a similarly successful context that distinguishes between the two sides of the equivalence. Conversely, since all expressions and contexts in PPC are guaranteed to evaluate in polynomial time and since the class of terms is precisely the class of probabilistic poly-time functions, every successfully distinguishing context can be translated into a successful attack on the corresponding game conditions.

Hiding: An implementation Impl is *hiding* if for an honest initiator, no adversary can gain, with non-negligible advantage, information about the committed bit. In other words, probability P_{Adv} that the attacker Adv, after interacting with an honest initiator committing to a randomly chosen bit b, successfully guesses the bit b should be close to a half. Writing this property as an equivalence yields:

$$\begin{aligned}
&\nu(\mathbb{C}, c).(\mathsf{Impl}[\mathbb{C}, \mathbb{D}] \mid \mathsf{out}(c, \mathsf{rand}) \mid \mathsf{in}(c, b).\\
&\quad \mathsf{SendCommit}^{\eta}(b, \mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{in}(d, b').\mathsf{out}(dec, b' \stackrel{?}{=} b))\\
\cong\ &\nu(\mathbb{C}, c).(\mathsf{Impl}[\mathbb{C}, \mathbb{D}] \mid \mathsf{out}(c, \mathsf{rand}) \mid \mathsf{in}(c, b).\\
&\quad \mathsf{SendCommit}^{\eta}(b, \mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{in}(d, b').\mathsf{out}(dec, \mathsf{rand}))
\end{aligned}$$

Both expressions select a random bit and commit to it. The adversary (expressed as a context) interacts with the commitment protocol and tries to guess the committed-to value. The difference between the two expressions is that the LHS tests, over the channel dec, whether the adversary's guess matches the chosen bit, while the RHS assumes, again over the channel dec, that the adversary fails with probability 1/2. Clearly, any successfully distinguishing context must guess the bit with non-negligible advantage, thereby proving the existence of an adversary that violates the hiding property. Hence, we can naturally express the hiding condition that for all Adv, the probability $P_{Adv} - \frac{1}{2}$ is negligible in η as a process calculus equivalence. To say that an implementation is perfectly or information-theoretically secure we require that $\forall Adv\colon P_{Adv} - \frac{1}{2} = 0$, which is the same as replacing $\cong$ by $=$ in the equivalence above.

Binding: The *binding* property is that no adversary can open a commitment to an arbitrary value. This condition can be restated using a game in which the adversary commits to a challenger (an honest responder), who then picks a random b and challenges the adversary to open the commitment to b. As an equivalence, it is stated as:

$$\begin{aligned}
&\nu(\mathbb{C}, d)(\mathsf{GetCommit}^{\eta}(\mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{out}(d, \mathsf{rand}) \mid \mathsf{in}(d, b).\mathsf{out}(c, b).\\
&\quad \mathsf{Verify}^{\eta}(\sigma, \mathbb{C}) \text{ returns } \langle\mathbf{r}\rangle.\mathsf{out}(dec, \mathbf{r} \stackrel{?}{=} b) \mid \mathsf{Impl}[\mathbb{C}, \mathbb{D}])\\
\cong\ &\nu(\mathbb{C}, d)(\mathsf{GetCommit}^{\eta}(\mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{out}(d, \mathsf{rand}) \mid \mathsf{in}(d, b).\mathsf{out}(c, b).\\
&\quad \mathsf{Verify}^{\eta}(\sigma, \mathbb{C}) \text{ returns } \langle\mathbf{r}\rangle.\mathsf{out}(dec, \mathsf{if}\ \mathbf{r} \stackrel{?}{=} \perp \text{ then false else rand}) \mid \mathsf{Impl}[\mathbb{C}, \mathbb{D}])
\end{aligned}$$

Here both expressions wait for a commitment, and then challenge the adversary to open the commitment to a randomly chosen bit. The LHS tests whether the adversary successfully does so, whilst the RHS assumes that if the attempt to open does not fail (i.e., the result of Verify is not $\perp$) the adversary fails with probability 1/2. Perfect binding is expressed by replacing $\cong$ with $=$ in the equivalence above.

Correctness: An implementation Impl is *correct* if an honest responder is able to verify an opened commitment by an honest initiator with overwhelming probability. This correctness property may be expressed as the process calculus equivalence.

$$\begin{aligned}
&\nu(\mathbb{C}, c)(\mathsf{out}(c, \mathtt{rand}) \mid \mathsf{in}(c, b).\mathsf{SendCommit}^{\eta}(b, \mathbb{C}) \text{ returns } \langle\sigma_I\rangle\\
&.\mathsf{Open}^{\eta}(\sigma_I, \mathbb{C}) \text{ returns } \emptyset.\mathsf{in}(d, b').\mathsf{out}(dec, b \stackrel{?}{=} b') \mid \mathsf{Impl}[\mathbb{C}, \mathbb{D}]) \mid\\
&\nu(\mathbb{C}')(\mathsf{GetCommit}^{\eta}(\mathbb{C}') \text{ returns } \langle\sigma_R\rangle.\mathsf{Verify}^{\eta}(\sigma_R, \mathbb{C}') \text{ returns } \langle\mathbf{r}\rangle.\mathsf{out}(d, \mathbf{r}) \mid \mathsf{Impl}[\mathbb{C}', \mathbb{D}])\\
\cong\ &\nu(\mathbb{C}, c)(\mathsf{out}(c, \mathtt{rand}) \mid \mathsf{in}(c, b).\mathsf{SendCommit}^{\eta}(b, \mathbb{C}) \text{ returns } \langle\sigma_I\rangle\\
&.\mathsf{Open}^{\eta}(\sigma_I, \mathbb{C}) \text{ returns } \emptyset.\mathsf{in}(d, b').\mathsf{out}(dec, \mathsf{true}) \mid \mathsf{Impl}[\mathbb{C}, \mathbb{D}]) \mid\\
&\nu(\mathbb{C}')(\mathsf{GetCommit}^{\eta}(\mathbb{C}') \text{ returns } \langle\sigma_R\rangle.\mathsf{Verify}^{\eta}(\sigma_R, \mathbb{C}') \text{ returns } \langle\mathbf{r}\rangle.\mathsf{out}(d, \mathbf{r}) \mid \mathsf{Impl}[\mathbb{C}', \mathbb{D}])
\end{aligned}$$

Here, both expressions pick a random bit, commit to it, and then try to open it. The LHS checks whether the verifier obtained the correct value for the bit, whilst the RHS assumes that the verifier gets the right value all the time.

3.3 Impossibility of Bit-Commitment

In this section, we show that no ideal functionality for bit-commitment can be realized. This generalizes the impossibility result for one particular functionality given in [13]. Other plausible bit-commitment functionalities can be constructed by adjusting the level of information and possible actions provided to the attacker by the functionality. For example, the functionality may let the attacker change the identity of the committer, hence making the commitment unauthenticated. Alternatively, the functionality may let the attacker change the committed bit if the attacker manages to correctly guess an internal secret of the functionality (since this is a low probability event, correctness still holds). Our proof shows that all of these variants (as well as further variants discussed in [13]) and their combinations are not realizable. Although we have not yet obtained a general characterization, our theorem applies under some setup assumptions, and fails in the common reference string model in accordance with the construction given in [13].

Our proof by contradiction roughly works as follows: given a real protocol $\mathcal{P}$ that realizes an ideal functionality $\mathcal{F}$ for bit-commitment, we construct another real protocol $\mathcal{Q}$ which provides the same correctness guarantee. However, in protocol $\mathcal{Q}$ all calls to the bit-commitment interface by principals are handled by copies of $\mathcal{F}$. As a consequence, $\mathcal{Q}$ provides perfect hiding and binding, which is a contradiction.

In order to state the theorem formally, we require some definitions. We say that $\mathcal{P}$ is a *real protocol* if each instance of $\mathcal{P}$ only communicates with one principal over a set of private channels. Intuitively, since it cannot communicate with two separate parties over private channels hidden from the adversary, a real protocol cannot act as a secure trusted third party. We say that a protocol $\mathcal{P}$ for bit-commitment is *terminating* when the following expression will, with high probability, produce the messages "go" and "done".

$$\nu(\mathbb{C})(\mathsf{SendCommit}^\eta(b,\mathbb{C}) \text{ returns } \langle\sigma_I\rangle.\mathsf{in}(c,z).\mathsf{Open}^\eta(\sigma_I,\mathbb{C}) \text{ returns } \emptyset.\mathsf{in}(d,z) \mid \mathcal{P}[\mathbb{C},\mathbb{D}] \mid$$
$$\nu(\mathbb{C}')(\mathsf{GetCommit}^\eta(\mathbb{C}') \text{ returns } \langle\sigma_R\rangle.\mathsf{out}(c,\text{“go”}).$$
$$\mathsf{Verify}^\eta(\sigma_R,\mathbb{C}') \text{ returns } \langle\mathbf{r}\rangle.\mathsf{out}(d,\text{“done”}) \mid \mathcal{P}[\mathbb{C}',\mathbb{D}])$$

Intuitively, if the function calls are implemented with $\mathcal{P}$, in the absence of the attacker two honest parties should be able to first finish the commitment stage, synchronize, and then finish the decommitment stage.

Theorem 1. *If $\mathcal{F}$ is an ideal functionality for bilateral bit-commitment, then there does not exist a terminating real protocol $\mathcal{P}$ that securely realizes $\mathcal{F}$.*

Before giving the proof, the following two lemmas will be useful. The first lemma states the well known fact [12] that perfect hiding and binding protocols for bit-commitment do not exist without a trusted third party. We omit the proof here. The second lemma states that any realization of $\mathcal{F}$ will also be correct for bit-commitment. The proof sketch is in Appendix A. Similarly, any realization of $\mathcal{F}$ will enjoy complexity-theoretic hiding and binding guarantees; however, we do not require this fact for the impossibility result.

Lemma 1. *There does not exist a terminating real protocol $\mathcal{P}$ which is correct with high probability, and both perfectly hiding and perfectly binding.*

Lemma 2. *If $\mathcal{P}$ is a terminating real protocol that securely realizes $\mathcal{F}$, then $\mathcal{P}$ is correct with high probability.*

Proof (Proof of Theorem 1). We assume that $\mathcal{P}$ securely realizes $\mathcal{F}$. It follows that for any configuration involving principals making use of $\mathcal{P}$, there exists a simulator $\mathcal{S}$ such that replacing the calls to $\mathcal{P}$ with calls to the simulator in conjunction with the functionality yields an indistinguishable configuration.

Consider the following real configuration when the environment plays the role of the responder honestly. It selects a bit and sends that bit to the initiator. The initiator then commits to that bit using a copy of the implementation $\mathcal{P}_\mathcal{I}$. The responder is corrupted by the adversary to simply forward messages to the environment. After corrupting the responder, the adversary simply forwards

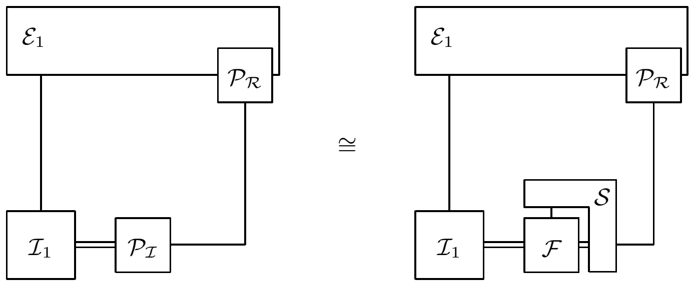

Fig. 2. Configurations for the first step

messages. The environment then honestly plays the responder's role using a copy of the real implementation $\mathcal{P}_\mathcal{R}$. At the conclusion of the commitment phase, the environment initiates decommitment by instructing the initiator to open. The environment then verifies the initiator's attempt to open, and then decides if the bit the initiator opened to was the bit the environment selected at the start of the run. The programs of the four principals are given below, where $\mathsf{Forward}(\mathbb{C} \leftrightarrow \mathbb{D})$ is an expression that forwards in an order-preserving way messages received on the channels $\mathbb{C}$ to channels $\mathbb{D}$ and vice versa:

$$\mathcal{E}_1 \equiv \nu(\mathbb{C}, c)(\mathsf{out}(c, \mathsf{rand}) \mid \mathsf{in}(c, b).\mathsf{out}(\mathrm{IO}_I, b).\mathsf{GetCommit}^\eta(\mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{out}(\mathrm{IO}_I, \mathsf{open}).$$
$$\mathsf{Verify}^\eta(\mathbf{r}, \mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{out}(dec, b \stackrel{?}{=} \mathbf{r}))$$
$$\mathcal{I}_1 \equiv \nu(\mathbb{C}')(\mathsf{in}(\mathrm{IO}_I, b).\mathsf{SendCommit}^\eta(b, \mathbb{C}') \text{ returns } \langle\sigma'\rangle.\mathsf{in}(\mathrm{IO}_I, x).\mathsf{Open}^\eta(\sigma', \mathbb{C}') \text{ returns } \emptyset)$$
$$\mathcal{A}_1 \equiv \mathsf{Forward}(\mathrm{Net}_I \leftrightarrow \mathrm{Net}_R)$$
$$\mathcal{R}_1 \equiv \mathsf{Forward}(\mathrm{IO}_R \leftrightarrow \mathrm{Net}_R)$$

This real configuration and its corresponding ideal configuration are shown in Figure 2 on the left and right, respectively (omitting the forwarders for clarity). Let us consider the ideal configuration. Here, the initiator uses the ideal functionality $\mathcal{F}$, whilst the environment continues using the real protocol. A simulator $\mathcal{S}$ must exist such that it can "convert" the messages of the functionality into messages that $\mathcal{P}_\mathcal{R}$ understands and vice versa. This simulator sits between $\mathcal{P}_\mathcal{R}$ and $\mathcal{F}$ and is connected to $\mathcal{F}$ via the bit-commitment interface and the unspecified interface of $\mathcal{F}$. Since $\mathcal{P}$ securely realizes $\mathcal{F}$, it follows that the configurations are indistinguishable. Furthermore, by Lemma 2 the environment in the real configuration must register success with high probability, since the adversary does nothing. Whence the expression $\mathcal{Q}$ consisting of $\mathcal{F}$ and $\mathcal{S}$ wired in the way that they are must be able to commit to $\mathcal{P}_\mathcal{R}$ and, then, successfully open the commitment.

Let us now consider another real configuration (Figure 3) where the initiator is corrupted to be a forwarder but the responder is honest. As before, the adversary, after corrupting the initiator, does nothing. The environment selects a bit and then runs the initiator's role directly. However, instead of using $\mathcal{P}$ to implement the initiator's role, the environment uses the expression $\mathcal{Q}$ from the first part of the argument. To commit, the environment sends the bit to the

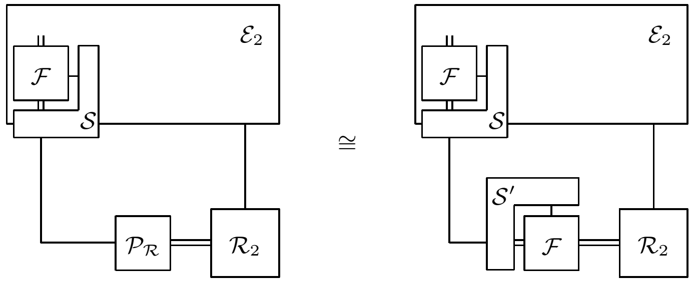

Fig. 3. Configurations for the second step

functionality whose messages are then translated by the simulator into messages suitable for the copy of the implementation $\mathcal{P}_{\mathcal{R}}$ used by the honest responder. After committing, the environment waits for a receipt from the responder, before decommitting. It then waits for the responder to send the bit it believes the initiator committed to and the environment checks that the bit it received was the same as the bit to which it committed. The responder, for its part, receives a commitment, sends a receipt to the environment, then verifies a commitment, and forwards the result to the environment. The programs are given below:

$$\begin{aligned}
\mathcal{E}_2 &\equiv \nu(\mathbb{C}, c)(\mathsf{out}(c, \mathsf{rand}) \mid \mathsf{in}(c, b).\mathsf{SendCommit}^{\eta}(b, \mathbb{C}) \text{ returns } \langle\sigma\rangle.\mathsf{in}(\mathrm{IO}_R, x).\\
&\quad \mathsf{Open}^{\eta}(\sigma, \mathbb{C}) \text{ returns } \emptyset.\mathsf{in}(\mathrm{IO}_R, b')\mathsf{out}(dec, b \stackrel{?}{=} b') \mid \mathcal{Q}[\mathbb{C}, \mathrm{IO}_R])\\
\mathcal{R}_2 &\equiv \nu(\mathbb{C}')(\mathsf{GetCommit}^{\eta}(\mathbb{C}') \text{ returns } \langle\sigma'\rangle.\mathsf{out}(\mathrm{IO}_R, \mathsf{receipt}).\\
&\quad \mathsf{Verify}^{\eta}(\mathbf{r}, \mathbb{C}') \text{ returns } \langle\sigma'\rangle.\mathsf{out}(\mathrm{IO}_R, \mathbf{r}))\\
\mathcal{A}_2 &\equiv \mathsf{Forward}(\mathrm{Net}_I \leftrightarrow \mathrm{Net}_R)\\
\mathcal{I}_2 &\equiv \mathsf{Forward}(\mathrm{IO}_I \leftrightarrow \mathrm{Net}_I)
\end{aligned}$$

In this scenario, the simulator $\mathcal{S}'$ sits between the expression $\mathcal{Q}$ (consisting of simulator $\mathcal{S}$ and functionality) and the functionality $\mathcal{F}$. Again, from secure realizability, Lemma 2, and the fact that $\mathcal{Q}$ looks like an initiator running the implementation $\mathcal{P}$, we know that in the real configuration, the environment will, with high probability, register a success. Therefore, so must the ideal configuration, whence the expression $\mathcal{Q}'$ consisting of $\mathcal{F}$ and $\mathcal{S}'$ must correctly play the role of the responder running the implementation $\mathcal{P}$.

If we look at the ideal configuration, we notice that the functionality is no longer working as a trusted third party. Every message is run through the simulators $\mathcal{S}$ and $\mathcal{S}'$. Thus, we have an implementation of bit-commitment that is a real protocol. The initiator executes the code given by the expression $\mathcal{Q}$ while the responder executes the code given by the expression $\mathcal{Q}'$. From the above argument it follows that the implementation $\mathcal{Q} \mid \mathcal{Q}'$ is a correct implementation. Furthermore, the $\mathcal{Q} \mid \mathcal{Q}'$ has to be information-theoretically hiding and binding because of the way they make use of the functionality. For example, to commit to a bit, the caller passes the bit to the functionality which, by definition, reveals no information about the bit regardless of the other parties in the configuration until the open step. Thus, we have a correct with high probability, and information-theoretically hiding and binding implementation of the bit-commitment interface that does not make use of trusted third parties. This contradicts Lemma 1.

4 Generalization of the Impossibility Result and Other Examples

In this section state a more general impossibility result: if $\mathcal{G}$ is a functionality and P is a protocol which uses $\mathcal{G}$ to achieve bit-commitment with perfect hiding and binding, then the functionality $\mathcal{G}$ cannot be realized. Intuitively, the functionality $\mathcal{G}$ together with protocol P constitutes an ideal functionality for bit-commitment $\mathcal{F}$, and any realization of $\mathcal{G}$ will lead to the realization of $\mathcal{F}$. Therefore, we would expect that all primitives that can be used to build bit-commitment are not realizable as functionalities. We illustrate this by showing that certain (rather strong) variants of symmetric encryption and group signatures cannot have realizable ideal functionalities. Due to space constrains, the security definitions are mostly informal and proof sketches have been moved to Appendix A.

Hybrid Protocols: We will consider implementations of primitives which, in addition to public channels, may use a particular functionality. Let $\mathcal{G}$ be any functionality, *a $\mathcal{G}$-hybrid protocol P* for a primitive is an implementation of the primitive's interface which does not make use of the trusted third party except maybe by making calls to $\mathcal{G}$'s interface. We will write $P[\mathcal{Q}]$ to denote an instance of P where calls to $\mathcal{G}$'s interface are handled by the implementation $\mathcal{Q}$ (real or ideal).

Theorem 2. *If $\mathcal{G}$ is a functionality and P is a terminating $\mathcal{G}$-hybrid protocol for bit-commitment which is correct with high probability and provides perfect hiding and perfect binding, then no protocol realizes functionality $\mathcal{G}$.*

Symmetric Encryption: Symmetric encryption primitive is defined by the standard interface for key generation, encryption and decryption.

$\mathsf{KeyGen}^\eta(\mathbb{C})$ returns $\langle K\rangle$ $\mathsf{Encrypt}^\eta(K,p,\mathbb{C})$ returns $\langle c\rangle$ $\mathsf{Decrypt}^\eta(K,c,\mathbb{C})$ returns $\langle p\rangle$

In addition to the obvious correctness property, we assume, as in [28], that the encryption scheme is CCA-secure and that it provides ciphertext integrity. Provably secure schemes with respect to these two properties exist under reasonable assumptions [29]. Informally, we can describe the properties as follows:

- *CCA-security* means that it is hard for an adaptive attacker with access to the decryption oracle to distinguish the plaintext from a random value of the same length given the ciphertext. *Perfect CCA-security* means that the probability of success is exactly half.

– *Integrity of ciphertexts* means that it is hard for an attacker to find a ciphertext c which will successfully decrypt unless that ciphertext has been produced by the encryption algorithm for some key and plaintext. *Perfect integrity of ciphertexts* means that the probability of an attacker finding such a ciphertext is zero.

Corollary 1. *If $\mathcal{F}$ is a functionality for symmetric encryption providing perfect CCA-security and perfect integrity of ciphertext then $\mathcal{F}$ cannot be realized.*

Group Signatures: Group signature primitive is defined by the interface for key generation, group signing, group signature verification and opening. For simplicity we will assume that the group is always of size two.

$\mathsf{GKeyGen}^{\eta}(\mathbb{C})$ returns $\langle gpk, gmsk, gsk_0, gsk_1\rangle$ $\mathsf{GSign}^{\eta}(m, gsk, \mathbb{C})$ returns $\langle sig\rangle$
$\mathsf{GVerify}^{\eta}(gpk, m, sig, \mathbb{C})$ returns $\langle result\rangle$ $\mathsf{GOpen}^{\eta}(gmsk, m, sig, \mathbb{C})$ returns $\langle identity\rangle$

In addition to the obvious correctness properties, we assume that the group signature scheme provides anonymity and traceability even against dishonest group managers. This is a stronger security requirement than the version principally considered in [30] (though [30] does briefly discuss this variant); [30] also shows that schemes with these properties exist if trapdoor permutations exist. Informally, we can describe the properties as follows:

– *Anonymity* means that it is hard for an adaptive attacker with access to an opening oracle to recover the identity of the signer given a signature and a message, even if the attacker has all the signing keys. *Perfect anonymity* means that the probability of success is exactly half (assuming, as we do, only two possible signers).
– *Traceability* means that it is hard for an attacker that adaptively corrupts a coalition of signers and has access to a signing oracle to produce a valid message-signature pair that opens to a signer not in the coalition, even when the group manager is dishonest. *Perfect traceability* means that the probability of an attacker forging such a signature is zero.

Corollary 2. *If $\mathcal{F}$ is a functionality for group signatures providing perfect anonymity and perfect traceability then $\mathcal{F}$ cannot be realized.*

5 Conclusion and Future Directions

We articulate accepted practice in the literature by giving a precise definition of an *ideal* functionality satisfying any given game specification: An ideal functionality must be a process or a set of processes that realize the game conditions in an information-theoretic, rather than computational complexity, sense. Using this definition we show that bit commitment, group signatures, and other cryptographic concepts that are definable using games do not have any realizable ideal functionality.

This proof appears applicable to other functionalities, and to a range of so-called "setup assumptions." However, we have not yet characterized the applicable setup assumptions precisely. Some examples of setup assumptions are pre-shared keys, certificate authorities, random oracles or common reference strings. These additional assumptions can be captured by a hybrid model where parties have access to an additional ideal functionality "implementing" the assumption, such as a trusted certificate authority. As demonstrated in [13], there are realizable ideal functionalities for bit-commitment when parties have access to one form of common reference string functionality. One possibility is to restrict attention to functionalities that are used only in the initialization phase. Our intuition suggests that a similar impossibility proof can be constructed for this case as long as these setup functionalities are global, i.e. all honest parties can access them. This does not contradict the common reference string construction in [13] since a fresh instance of the common reference string functionality is required for each pair of participants engaged in a session and cannot be accessed by other honest parties. We hope to develop this idea more fully in future work.

An appealing property of indistinguishability-based specifications is the connection with composability: if a real security mechanism is indistinguishable from an ideal one, then any larger system using the real mechanism will behave in the same way as the same system using the ideal mechanism instead. In light of the limitations on indistinguishability-based specifications explored in this paper, there are several modifications to the basic theory that might provide useful forms of composability. One direction is to relax or modify the requirements for ideal functionality. For example, information-theoretic equivalence could be replaced with the indistinguishability of random systems in the sense of [31]. This would allow adaptive, computationally unbounded distinguishers to query the system at most polynomially many times in the security parameter. Another possible direction involves the modification of the Universal Composability framework recently considered in [32, 33], which allows a commitment functionality. In the modified framework, parties are typed in a certain way, and the typing must be respected by the simulator. On the other hand, since the intuition for some of these directions is not clear, it may be more productive to develop methods for stating and proving *conditional* forms of composability. More precisely, primitives and protocols could be guaranteed to operate securely only in environments that satisfies certain conditions. Games currently provide a very limited form of conditional composability, since a game condition provides guarantees for any system whose actions can be regarded as (or reduced to) moves in a relevant game. We also consider the work on protocol composition logic [34, 35] a potentially relevant form of conditional composability, since protocols or primitives proved correct in that framework carry guarantees that apply to any environment respecting certain invariants expressed explicitly in the logic.

References

1. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding. Volume 1807 of Lecture Notes in Computer Science., Springer-Verlag (2000) 259–274
2. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: the spi calculus. Information and Computation **143** (1999) 1–70 Expanded version available as SRC Research Report 149 (January 1998).
3. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science. (2001) 136 Full version available at `http://eprint.iacr.org/`.
4. Lincoln, P., Mitchell, J.C., Mitchell, M., Scedrov, A.: A probabilistic poly-time framework for protocol analysis. In: ACM Conference on Computer and Communications Security. (1998) 112–121
5. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: ACM Conference on Computer and Communications Security. (2000) 245–254
6. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations. In: CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, ACM Press (2003) 220–230
7. Backes, M., Pfitzmann, B., Waidner, M.: A general composition theorem for secure reactive systems. In: TCC '04: Proceedings of the 1st Theory of Cryptography Conference. Volume 2951 of Lecture Notes in Computer Science., Springer-Verlag (2004) 336–354
8. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004) `http://eprint.iacr.org/2004/332`.
9. Backes, M., Hofheinz, D.: How to break and repair a universally composable signature functionality. In: Information Security, 7th International Conference, ISC 2004, Proceedings. Volume 3225 of Lecture Notes in Computer Science., Springer-Verlag (2004) 61–72
10. Canetti, R.: Universally composable signature, certification, and authentication. In: CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop, IEEE Computer Society (2004) 219–233
11. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1) (1989) 186–208
12. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2000)
13. Canetti, R., Fischlin, M.: Universally composable commitments. In: Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Proceedings. Volume 2139 of Lecture Notes in Computer Science., Springer-Verlag (2001) 19–40
14. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings. Volume 2656 of Lecture Notes in Computer Science., Springer-Verlag (2003) 68–86

15. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding. Volume 2045 of Lecture Notes in Computer Science., Springer-Verlag (2001) 453–474
16. Milner, R.: Communication and Concurrency. International Series in Computer Science. Prentice Hall (1989)
17. van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. International Journal on Information and Computation **121**(1) (1995)
18. Ramanathan, A., Mitchell, J.C., Scedrov, A., Teague, V.: Probabilistic bisimulation and equivalence for security analysis of network protocols. In: Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Proceedings. Volume 2987 of Lecture Notes in Computer Science., Springer-Verlag (2004) 468–483
19. Mitchell, J.C., Mitchell, M., Scedrov, A.: A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In: FOCS '98: Proceedings of the 39th Annual IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society (1998) 725–733
20. Lincoln, P.D., Mitchell, J.C., Mitchell, M., Scedrov, A.: Probabilistic polynomial-time equivalence and security protocols. In: Formal Methods World Congress, vol. I. Number 1708 in Lecture Notes in Computer Science, Springer-Verlag (1999) 776–793
21. Mitchell, J.C., Ramanathan, A., Scedrov, A., Teague, V.: A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report). In: 17th Annual Conference on the Mathematical Foundations of Programming Semantics. Volume 45., Electronic notes in Theoretical Computer Science (2001)
22. Ramanathan, A., Mitchell, J.C., Scedrov, A., Teague, V.: Probabilistic bisimulation and equivalence for security analysis of network protocols. Unpublished, see `http://www-cs-students.stanford.edu/~ajith/` (2003)
23. Datta, A., Küsters, R., Mitchell, J.C., Ramanathan, A., Shmatikov, V.: Unifying equivalence-based definitions of protocol security. In: 2004 IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security (WITS 2004). (2004)
24. Datta, A., Küsters, R., Mitchell, J.C., Ramanathan, A.: On the relationships between notions of simulation-based security. In: TCC '05: Proceedings of the 2nd Theory of Cryptography Conference. Volume 3378 of Lecture Notes in Computer Science., Springer-Verlag (2005) 476–494
25. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding. Volume 2332 of Lecture Notes in Computer Science., Springer-Verlag (2002) 337–351
26. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC '02: Proceedings of the 34th annual ACM symposium on Theory of computing, ACM Press (2002) 494–503
27. Naor, M.: Bit commitment using pseudorandomness. Journal of Cryptology **4**(2) (1991) 151–158

28. Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In: CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop, IEEE Computer Society (2004) 204–218
29. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A block-cipher mode of operation for efficient authenticated encryption. In: CCS '01: Proceedings of the 8th ACM Conference on Computer and Communications Security, ACM Press (2001) 196–205
30. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings. Volume 2656 of Lecture Notes in Computer Science., Springer-Verlag (2003) 614–629
31. Maurer, U.M.: Indistinguishability of random systems. In: Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding. Volume 2332 of Lecture Notes in Computer Science., Springer-Verlag (2002) 110–132
32. Prabhakaran, M., Sahai, A.: New notions of security: Achieving universal composability without trusted setup. In: STOC '04: Proceedings of the 36th annual ACM symposium on Theory of computing, ACM Press (2004) 242–251
33. Prabhakaran, M., Sahai, A.: Relaxing environmental security: Monitored functionalities. In: TCC '05: Proceedings of the 2nd Theory of Cryptography Conference. Volume 3378 of Lecture Notes in Computer Science., Springer-Verlag (2005) 104–127
34. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security **13** (2005) 423–482
35. He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.C.: A modular correctness proof of TLS and IEEE 802.11i. In: ACM Conference on Computer and Communications Security. (2005)

A Proof Sketches

Proof (Proof sketch of Lemma 2). Consider a configuration consisting of an honest initiator running the implementation $\mathcal{P}$, an honest responder running the implementation $\mathcal{P}$, and an adversary that does nothing. The initiator waits for a bit from the environment and then commits to that bit. It then waits for a message from the environment and then opens its commitment. The responder, after receiving a commitment, sends a receipt to the environment. After a successful verification of an attempt to open the commitment, it sends the opened-to value to the environment. The environment selects a bit, sends it to the initiator and waits for a receipt from the responder. Once it gets this message, it instructs the initiator to open its commitment, and then waits for the responder to reveal the bit to which the initiator committed. If that bit matches the bit the environment selects at the start of the run, the environment registers success. Otherwise it registers failure.

By the terminating property of $\mathcal{P}$, we know that this run will complete. The ideal configuration has the initiator talking to the functionality which talks directly to the responder. Though a simulator exists in the ideal configuration, it can do nothing since both the initiator and responder are connected directly to the functionality. By virtue of the functionality's correctness, we know that in the ideal configuration the environment will register success with high probability. Since $\mathcal{P}$ securely realizes $\mathcal{F}$, the environment must register success in the real configuration with high probability. Whence the correctness of $\mathcal{P}$ is established.

Proof (Proof sketch of Theorem 2). Assume that P is a terminating $\mathcal{G}$-hybrid protocol for bit-commitment, which is correct with high probability and provides perfect hiding and binding. A functionality $\mathcal{F} = P[\mathcal{G}]$ is clearly an ideal functionality for bit-commitment. Let Q be a real protocol which is a realization of $\mathcal{G}$, consider a real protocol $R = P[Q]$ in which all the calls of P to the functionality $\mathcal{G}$ are implemented with Q. We claim that R is a secure realization of $\mathcal{F}$. Choose any real configuration for R, consisting of an attacker A, and parties $P_1, \ldots, P_n$. We need to show that there is a simulator S such that for any environment E this configuration is indistinguishable from one where parties call functionality $\mathcal{F}$ instead of R. This configuration is also a real configuration for the protocol Q. Therefore, there is a simulator such that when all calls to Q are replaced with calls to $\mathcal{G}$, the two configurations are indistinguishable for any environment. Since this ideal configuration is exactly the ideal configuration for $\mathcal{F}$ we are done with the proof, because by Theorem 1 there can be no protocol realizing any ideal functionality for bit-commitment.

Proof (Proof sketch of Corollary 1). Assume that $\mathcal{F}$ is an ideal functionality for symmetric encryption and construct a $\mathcal{F}$-hybrid protocol for bit-commitment providing perfect hiding and binding. The initiator can commit to b by generating a new key, encrypting b and sending the ciphertext via public channel. To open the commitment, initiator sends the key. This protocol provides perfect hiding because of the perfect CCA-security provided $\mathcal{F}$, and provides perfect binding because of the perfect integrity of ciphertexts provided by $\mathcal{F}$. By Theorem 2, functionality $\mathcal{F}$ cannot be realized.

Proof (Proof sketch of Corollary 2). Construct a $\mathcal{F}$-hybrid protocol for bit-commitment providing perfect hiding and binding. The initiator can commit to b by generating all the group keys, signing a random message with b's signing key, and then sending, as the signature, the tuple consisting of the b's signature, the message, the group public key, and all the signing keys. To open the commitment, the initiator sends the group manager's secret key. This protocol provides perfect hiding because of the perfect anonymity provided $\mathcal{F}$, and provides perfect binding because of the perfect traceability provided by $\mathcal{F}$. By Theorem 2, functionality $\mathcal{F}$ cannot be realized.