The Impossibility of Realizable Ideal Functionality

Anupam Datta
Stanford University
danupam@cs.stanford.edu

Ante Derek
Stanford University
aderek@cs.stanford.edu

John C. Mitchell
Stanford University
jcm@cs.stanford.edu

Ajith Ramanathan
Stanford University
ajith@cs.stanford.edu

Andre Scedrov
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. One reason that an ideal functionality is appealing is that if an implementation cannot be distinguished from an ideal functionality, by any feasible attack in any environment, then the mechanism is therefore secure in any larger system that uses it. We make accepted aspects of ideal functionality precise by relating ideal functionality to game specifications, and show that bit commitment, group signatures, and other cryptographic concepts do not have any realizable ideal functionality. This suggests that in order to develop composable security conditions, either alternative notions of ideal functionality must be developed, or another specification method must be used.

1 Introduction

Generally, asserting that a system is *secure* involves identifying a set of properties that are preserved in the face of a set of attacks. Many such 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) [5] 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 are preserved in the face of attack. Another way of specifying security properties, advanced in the work on Universal Composability (UC) [7, 8, 11, 14] and relevant to systems such as spi-calculus [1] based on process equivalence, uses ideal functionalities. In this approach, an idealized way of achieving some goal is presented, possibly using mechanisms such as private channels and trusted third parties that are not available in practical deployment. 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 [1, 7]. In contrast, if a mechanism satisfies a game condition, there is no guarantee about 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-functionality specifications and show that games specify more primitives than ideal functionalities. In particular, we provide a precise definition of *ideal* functionality and prove that while bit-commitment may be specified using games, there is no realizable ideal functionality for bit-commitment. This is a negative result about ideal functionality, since there are constructions of bit-commitment protocols that are provably correct under reasonable assumptions (see, e.g., [17]). We also show that there is no realizable ideal functionality for other reasonable and implementable cryptographic conditions, including a form of group signatures and symmetric encryption, under certain conditions that allow the encryption key to be revealed after it is used.

In order to state and prove these results, we must give a precise definition of the *ideal functionality* corresponding to a game specification. Since the purpose of giving an ideal functionality is to provide a point of reference for judging potential implementations (generally presented as high-level algorithms rather than executable source code), an ideal functionality must be "secure by construction," or at least secure in a stronger sense than would be required of a sample implementation. Based on standard practice in the literature, and the absence of any compelling examples to the contrary, we believe that an ideal functionality must be a process or set of processes that satisfy the game specifications in an information-theoretic sense. Applied to encryption, for example, this means that an ideal functionality for encryption must not provide any information about the plaintext to the adversary. Since an implementation of the ideal functionality need only be computationally indistinguishable, a realization of this ideal functionality may depend in some way on the plaintext, but not in a way that is recognizable by a computationally bounded adversary.

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 [17]. Except for this final step, all steps in this proof rely on a hybrid argument and can be applied to any cryptographic primitive and corresponding ideal functionality with conflicting games. Other concrete examples considered in this paper include forms of group signatures and symmetric encryption. While impossibility results for these primitives could be proved by instantiating the general proof method, we present simpler proofs by reducing bit-commitment to these primitives.

From a technical perspective, our results strengthen and generalize Canetti and Fischlin's proof that a particular ideal functionality for bit-commitment is not realized by any protocol without a trusted third party [9]. We strengthen this result by defining ideal functionality in general by reference to game conditions, and then showing that *no* ideal functionality for bit-commitment can be realized. In other words, Canetti and Fischlin leave open the possibility that some other ideal functionality for bit-commitment is realizable, while our results show that this is not possible. Previous investigators have also proved negative results for specific ideal functionalities. For example, in addition to a giving a number of positive results, Canetti [7] shows that particular functionalities

for ideal coin tossing, zero-knowledge, and oblivious transfer are not realizable. Canetti et al [12] show that a class of specific functionalities for secure multi-party computation are not realizable, while Canetti and Krawczyk [10] compare indistinguishability-based and simulatability-based definitions of security in the context of key-exchange protocols. Some of these papers also modify UC in order to present realizable ideal functionalities. We regard the technical results in the present paper as an explanation of why these alternatives to UC are needed.

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 [7, 17, 18] is a straightforward way of modularizing games that use a functionality. In the IND-CCA2 game, for example, the encryption and decryption primitives could be simple functions on bit strings. However, in defining bit-commitment by games, we wish to impose conditions on protocols. Therefore, we define games that use "protocol subroutines," invoked by a form of function call that is definable using sends and receives on a dedicated communication channel. In principle, the same construction could be done using Turing machines, using a separate function-call-and-return tape similar to an oracle tape, and additional sequential machines and tapes.

One of the primary reasons to specify security properties using an ideal functionality is the composition property which implies that a primitive remains secure when used in any larger system [1, 4, 7, 14]. Since composability is highly desirable, our negative results suggest that either some variant of ideal functionality is needed, or a completely different approach is required. The difficulties encountered in formulating an ideal functionality for signatures [2, 8, 14], and the need in previous work to assume that a symmetric key is not transmitted or revealed after it is used [3], motivate investigation of alternative approaches, such as the widely accepted assume-guarantee paradigm for reasoning about distributed systems [22]. Intuitively, we see no reason to believe that all useful cryptographic primitives, or security mechanisms built from them, will compose in any sort of universal way. Most protocols, for example, work only under assumptions about the environment in which they are used. For example, a protocol in which signatures are used for authentication may only work properly if the signing principal does not also concurrently execute another protocol that produces similar signatures. Therefore, we believe it may be much more fruitful to develop methods for stating and proving conditional composability, guaranteeing that primitives and protocols will operate securely in any environment that satisfies certain conditions.

2 Preliminaries

2.1 Probabilistic Process Calculus

Process calculus is a standard framework for studying concurrency [21,31] that has proved useful for reasoning about security protocols [1,29]. 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 the communication network in a distributed system environment or the shared memory in a parallel processor.

A probabilistic polynomial-time process calculus (PPC) for security protocols is developed in [19, 23, 24] and updated in more recent papers [28, 29]. It consists of a set of *terms* that do not perform any communications and that represent probabilistic polynomial-time computation, *expressions* that can communicate with other expressions, 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. An *input* operator $\mathsf{in}(c,x).\mathcal{P}$ waits until it receives a value on the channel c and then substitutes that value for the free variable x in $\mathcal{P}$. Similarly, an *output* $\mathsf{out}(c,\mathrm{T}).\mathcal{P}$ evaluates the term T, transmits that value 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 model secure and/or authenticated channels. For convenience, we always α-rename channel names so that they are all distinct. The *match* operator [T] executes the expression bound to the match iff T evaluates to 1. The *parallel composition* operator, |, applied to two expressions allows them to evaluate concurrently, or communicate with one another over a shared channel. 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.

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. We repeat this procedure until no communication steps are possible. We note that there are subtleties involved with the choice of scheduler. For a full exposition of these issues we refer the reader to [removed].

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:

$$\text{Prob}\,[\text{evaluating } \mathcal{C}[\mathcal{P}] \text{ produces } o] - \text{Prob}\,[\text{evaluating } \mathcal{C}[\mathcal{Q}] \text{ produces } o] \;\text{ is negligible in } \eta$$

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 observer. 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:

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

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 built 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 c. This can be done by writing another process Q that receives a number n on channel d and then sends the bits of n on a channel c. The interaction between P and Q can be structured so that P initiates the actions by Q by sending a message $\mathsf{Send}(n)$ on channel d dedicated for this purpose. If P wants a return value, such as an indication that Q has completed sending the message, then P can execute an input action on channel d immediately after sending $\mathsf{Send}(n)$. This pattern of sends and received corresponds to an ordinary function call and return, implemented by communication actions on a dedicated channel between processes. To provide useful intuition, and to simplify terminology, we will refer to such communication between P and Q as a *function call and return.* To emphasize that this hides the structure of Q from the calling process P, we sometimes refer to this as a *black-box call.* Function calls and returns turn out to be a very useful concept in structuring games that specify properties of cryptographic primitives. 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. However, the process implementing the function call may perform observable actions as a result of the call, including sending messages on public channels.

A function call in PPC will be written $\mathsf{Call}^{\eta}(\langle\mathtt{params}\rangle, \mathbb{C})$ $\mathsf{returns}$ $\langle\mathtt{vars}\rangle.\mathcal{P}$, which may be regarded as syntactic sugar for a combination of PPC sends and receives. This syntax indicates the invocation of the function Call with security parameter η and parameters $\mathtt{params}$. The execution of Call will occur over the channels in $\mathbb{C}$. The variables $\mathtt{vars}$ are assumed to be free variables of $\mathcal{P}$ that are bound by the call. On return from the execution of the blackbox call, values will be substituted into the free variables of $\mathcal{P}$ bound by the call. We write $\mathsf{Impl}[\mathbb{C}, \mathbb{D}]$ to denote an implementation of a blackbox call that communicates with the caller using the channels in $\mathbb{C}$, and implements the call with a protocol that executes over the channels in $\mathbb{D}$.

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. We assume that the interface of any primitive provides enough operations to allow the primitive to be defined and used. For example, if calls to a key generation function are not allowed by the interface, then we cannot use the encryption scheme (since there is no way to get keys) and we cannot state semantic security properly.

A *protocol* for a primitive is process that responds to a set of function calls and supplies the associated returns, without any additional assumptions such as a trusted third party (TTP). 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 a trusted third party (see Section 3.3) or other mechanisms not normally available in practice. An *ideal functionality* 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). In addition to implementing the interface of the primitive, a functionality may communicate with the attacker in an arbitrary way. For example, an ideal functionality for signatures [8] will let the attacker choose the bitstrings for signatures.

2.4 Universal Composability

Universal composability [7, 9, 11–13] 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 [15, 16]. 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 $\mathcal{P}_i$ $(i \geq k)$ 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 $\mathcal{E}$:

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

Here the first 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

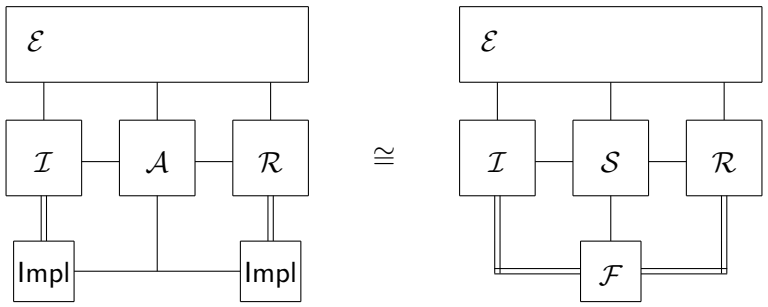

Figure 1: Real and ideal configurations with two honest participants

makes the ideal and real configurations are 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 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 [17, 25], and that they can be realized using standard cryptographic assumptions such as the existence of pseudorandom functions [25].

3.1 Commitment interface

A bipartite bit-commitment scheme provides four function calls:

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

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 open the 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—correctness, hiding, and binding—on bit-commitment [17, 25]. 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 [9] 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 (acting as a responder or otherwise) can gain, with non-negligible advantage, information about the committed bit. The probability P_{Adv} that an attacker Adv successfully extracts information about the committed bit from interaction with an honest initiator is the probability it successfully guesses the value of an honest commitment given a commitment to a randomly chosen bit. Writing this property as an equivalence yields:

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

Both expressions select a random bit and commit to it. They then wait for an adversary (expressed as a context) 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.

We note that in order to prevent the adversary from unfairly interfering with the communication between the challenger and the implementation Impl, we make the channels between the challenger and Impl private. 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:

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

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 $\bot$) 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.

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

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. 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* just when it does not act as a trusted third party. This means that a given copy of $\mathcal{P}$ only communicates with one principal over a set of private channels, in addition to any public communication it may perform. 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", if the function calls are implemented with $\mathcal{P}$.

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

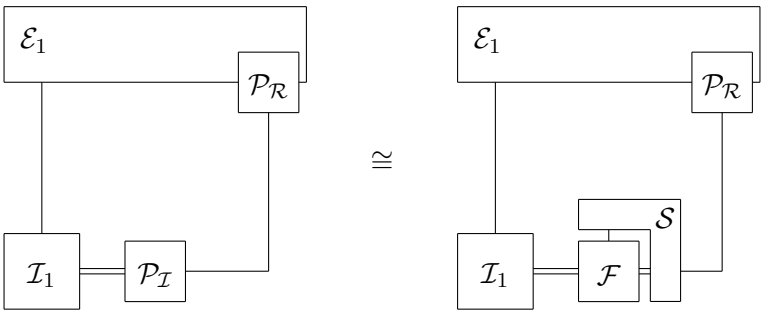

Figure 2: Configurations for the first step

Intuitively, if we see both synchronization messages in a run which synchronizes after commitment, on the message "go", and then synchronizes after decommitment, on the message "done", then each phase must consist of two roles (initiator and responder) that are implemented by expressions that run for only a finite time.

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 [17] 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 2. *There does not exist a terminating real protocol $\mathcal{P}$ which is correct with high probability, and both perfectly hiding and perfectly biding.*

Lemma 3. *If $\mathcal{P}$ is a terminating real protocol that securely realizes $\mathcal{F}$, then $\mathcal{P}$ is correct with high probability.*

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 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:

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

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 uncorrupted principal, 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 3 the environment in the real configuration must register success with high probability, since the adversary does nothing (after corrupting the responder to be a forwarder). 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 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 decommiting. 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:

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

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 3, and the fact

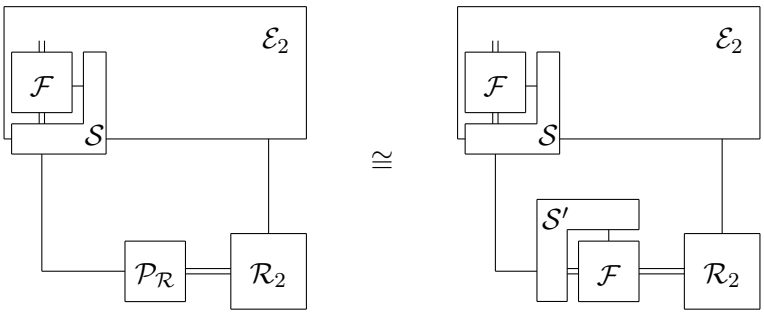

Figure 3: Configurations for the second step

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 2. □

4 Generalization of the Impossibility Result and Other Examples

In this section we 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 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 For the purpose of stating a general theorem, 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 are calls to $\mathcal{G}$'s interface are handled

by the implementation $\mathcal{Q}$ (real or ideal).

Theorem 4. *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.

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

In addition to the obvious correctness property, we assume, as in [3], 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 [30]. Informally, we can describe the properties as follows:

- *CCA-security* means that it 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 5. *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.

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

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 [6] (though [6] does briefly discuss this variant); [6] 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 6. *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 the ideal functionality associated with 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. The proof appears applicable to other primitives, which we expect to explore in the near future.

Since universal composability has this inherent limitation, there are two possible directions for further work. One possibility is to modify the UC framework. For example, it could be fruitful to investigate a more general version of ideal functionality obtained by replacing information-theoretic equivalence with the indistinguishability of random systems in the sense of [20]. 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 [26, 27], 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* composability. In conditional composability, primitives and protocols would be guaranteed to operate securely only in environments that satisfies certain conditions. At present, this seems the more promising general research direction.

References

[1] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 143:1–70, 1999. Expanded version available as SRC Research Report 149 (January 1998).

[2] Michael Backes and Dennis Hofheinz. 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*, pages 61–72. Springer-Verlag, 2004.

[3] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 204–218. IEEE Computer Society, 2004.

[4] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 220–230. ACM Press, 2003.

[5] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. 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*, pages 259–274. Springer-Verlag, 2000.

[6] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. 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*, pages 614–629. Springer-Verlag, 2003.

[7] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 136, 2001. Full version available at `http://eprint.iacr.org/`.

[8] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 219–233. IEEE Computer Society, 2004.

[9] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 2001.

[10] Ran Canetti and Hugo Krawczyk. 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*, pages 453–474. Springer-Verlag, 2001.

[11] Ran Canetti and Hugo Krawczyk. 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*, pages 337–351. Springer-Verlag, 2002.

[12] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. 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*, pages 68–86. Springer-Verlag, 2003.

[13] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC '02: Proceedings of the 34th annual ACM symposium on Theory of computing*, pages 494–503. ACM Press, 2002.

[14] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer-Verlag, 2003.

[15] Anupam Datta, Ralf Küsters, John C. Mitchell, and Ajith Ramanathan. 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*, pages 476–494. Springer-Verlag, 2005.

[16] Anupam Datta, Ralf Küsters, John C. Mitchell, Ajith Ramanathan, and Vitaly Shmatikov. 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.

[17] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2000.

[18] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[19] Patrick D. Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. Probabilistic polynomial-time equivalence and security protocols. In *Formal Methods World Congress, vol. I*, number 1708 in Lecture Notes in Computer Science, pages 776–793. Springer-Verlag, 1999.

[20] Ueli M. Maurer. 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*, pages 110–132. Springer-Verlag, 2002.

[21] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[22] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, 1981.

[23] John C. Mitchell, Mark Mitchell, and Andre Scedrov. 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*, pages 725–733. IEEE Computer Society, 1998.

[24] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. 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.

[25] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[26] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In *STOC '04: Proceedings of the 36th annual ACM symposium on Theory of computing*, pages 242–251. ACM Press, 2004.

[27] Manoj Prabhakaran and Amit Sahai. Relaxing environmental security: Monitored functionalities. In *TCC '05: Proceedings of the 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 104–127. Springer-Verlag, 2005.

[28] Ajith Ramanathan, John C. Mitchell, Andre Scedrov, and Vanessa Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. Unpublished, see `http://www-cs-students.stanford.edu/~ajith/`, 2003.

[29] Ajith Ramanathan, John C. Mitchell, Andre Scedrov, and Vanessa Teague. 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*, pages 468–483. Springer-Verlag, 2004.

[30] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. 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*, pages 196–205. ACM Press, 2001.

[31] Robert J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative, and stratified models of probabilistic processes. *International Journal on Information and Computation*, 121(1), August 1995.

A Proof Sketches

Proof sketch of Lemma 3. 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 sketch of Theorem 4. 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 sketch of Corollary 5. 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 4, functionality $\mathcal{F}$ cannot be realized. □

Proof sketch of Corollary 6. 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 4, functionality $\mathcal{F}$ cannot be realized. □