Acknowledgements

Upon the accomplishment of this thesis, I would like to express my sincere thanks to those who offered me help, advice, guidance, and criticism during the progress of my work.

I'd like to extend my heartfelt gratitude to all those who have offered me kind help and valuable comments on the writing of my thesis. I'm particularly thankful to my academic supervisor, Dr. Bogdan Warinschi, who proposed scholarly guidance and extensive criticisms on the manuscript of this thesis. I am also thankful to my friend, Georgios Kafanas, who gave me his comments throughout the thesis.

Profound thanks and sincere appreciation are also extended to those who have taught me and given me valuable suggestions with their profound knowledge and experience during postgraduate study, from whose advice and lectures I have earned a lot.

Finally, I would like to thank to my beloved family and friends who never fail to give me great help, suggestions and encouragement.

Abstract

There are two frameworks for defining security in modern cryptography. One is based on notion of simulation which uses ideal functionalities. We refer it to as Universal Composability (UC). Another one is based on the notion of cryptographic games. There has been a lot of work on comparing these two different definitional styles recently. Some interesting negative results about the existence of the ideal functionalities are proved. We call these results the impossibility result. The goal of this thesis is to relate the game specifications to the ideal functionalities and to prove a more general form of the impossibility result. More specifically, we have the following things:

- Review the UC-security framework and the cryptographic games.
- Give the canonical form of the ideal functionality.
- Give a precise definition about the game satisfaction and the descriptions of the general hiding game and the general binding game.
- Prove two theorems which relate the game conditions to the ideal functionalities:
    - If the ideal functionality is syntactically hiding a variable, then the functionality satisfies the general hiding game.
    - If the ideal functionality is syntactically binding a variable, then the functionality satisfies the general binding game.
- Prove a more general impossibility result: the functionalities for the general hiding and general binding game are not realizable.

On the Limits of Realizability of Ideal Functionality Defined Through Games

Tianxiang Cui

September 11, 2011

Contents

1 Introduction

1.1 Background

The security conditions for the cryptographic primitives can be expressed in many ways. The mostly common used ones are cryptographic games and ideal functionalities. Usually cryptographic games describe the security conditions in a standalone manner. For instance the adaptive chosen ciphertext attack (IND-CCA2) game of a semantically secure encryption scheme can be expressed as saying that the probability of any adversary winning a certain game against a challenger is negligible [CHH+07, BBM00]. The definition of the game specifies the actions and information available to the adversary and also the corresponding winning conditions. However this definition does not consider the behaviour of the protocols in the complex environment. Another way to express the security conditions is to use the ideal functionalities [AG97, Can01, LMMS98]. Usually it refers to Universal Composability (UC)-security framework. The UC-security framework can capture the unexpected interactions during the complex protocol execution and therefore it provides a generic method for analyzing the security of protocols when they are composed. Informally we say a primitive is secure if there exists no adversary which can make a distinction between the real protocol and the ideal functionalities under any environment. The advantage of this is that the composable notions of security can be implied by the indistinguishability of the ideal functionalities.

Recently, there have been a lot of work on comparing the game conditions and the ideal functionalities [Can05, BDD+06]. Some interesting results about the realizability of the ideal functionalities are explored under certain assumptions [Can05, BDD+06, Can01, CHK+05, CKL06]. These results are called the impossibility results. Consider a concept like the commitment scheme, Canetti et al [CF01] shows that a particular ideal functionality for bit commitment is not realizable. Based on their work, Datta et al [BDD+06] shows that no commitment functionalities are realizable. Canetti et al [CHK+05] also shows that a particular functionality for the password-based key exchange protocol is not realizable.

1.2 Our contribution

Our main result is an even more general impossibility result based on the previous work [CF01, BDD+06]. It says that given any ideal functionality which satisfies both the general hiding game and general binding game, that ideal functionality is not realizable. It is more

general in the sense that [CF01] focus on a specific ideal functionality for the commitment games and they show that specific ideal functionality is not realizable, [BDD+06] focus on all the ideal functionalities for the specific games (the commitment games) and they show all the ideal functionalities for that specific games are not realizable. Our work focus on all the ideal functionalities for the class of the games and we show all the ideal functionalities for that class of the games are not realizable.

Another contribution is that we give the precise definition of the game satisfaction and the descriptions of the general hiding game and the general binding game. Intuitively, the general hiding game is the game where the information of the communications between the parties during the protocol execution can be erased and the general binding game is the game where only specified traces of the communication between parties are allowed during the protocol execution. Then we proved two theorems which relate those game conditions to the ideal functionalities. The first theorem says that if the functionality is syntactically hiding the message, then this functionality satisfies the hiding game. Similarly, the second theorem says that if the functionality is syntactically binding the message, then this functionality satisfies the binding game.

1.3 The structure of the rest of this work

In section 2 we review the Universal Composition framework and the main Universal Composition theorem. In section 3 we review the general idea behind the cryptographic games and we also review the commitment scheme along with its corresponding games. In section 4 we review some ideal functionalities for the commitment scheme and we also derive a canonical ideal functionality for this work. Then we give the precise definition of the game satisfaction. At the end of the section 4, we give the description of the general hiding game and the general binding game and then present our theorems which relate the game conditions to the ideal functionalities. In section 5 we review some previous impossibility work and then we present our impossibility result along with its corresponding proof. In section 6 we give some possible further directions for the research. In appendix, we give one form of the generalization of the games which can be potentially used for our work.

2 Universal Composition

2.1 Cryptographic primitives and the ideal functionality

Generally, the cryptographic primitives can be considered as the well-implemented low level algorithms. They are usually used to build the security systems. For our work, we take the idea from Datta et al [BDD+06] as the follows: the cryptographic primitives are the interface and a set of correctness conditions or a set of security conditions which can be expressed by using the interface. Then the interface can be defined as a list of actions which are available to the primitives. It can be represented as a set of algorithms. For instance, there are three probabilistic algorithms of the interface to an encryption primitive: the key generation algorithm, the encryption algorithm and the decryption algorithm [BDD+06]. The correctness condition says that it should give the origin message back when running the decryption algorithm of an encrypted message under the correct key. We can say that an encryption scheme is semantically secure if there exists no probabilistic polynomial-time adversary that can tell which of the two messages has been encrypted with probability significantly better than 0.5 (the probability of randomly guessing by tossing a coin)[BDD+06]. This is called the security condition.

A protocol for a cryptographic primitive can be considered as a process which could respond to the function calls and then provide the corresponding returns [BDD+06]. It should not involve any additional private communication during the process. Apart from supporting the interface as the protocol does, the functionality may also include some private communication such as a trusted third party under certain restrictions. Since the functionalities are unrealistic on the public network, the restrictions are used as an obstacle of the abuse of the security in the private communication. On the other hand, there are no restrictions on the functionalities about communicating or revealing information to the adversary. One example presented in [Can04] shows that a functionality of a signature scheme can allow the adversary choose bit-strings for signatures.

Recently it is very common to analysis the security of a protocol in terms of using a trusted party paradigm. In this paradigm, the execution of the real protocol is compared with an ideal process. All the inputs are visible to the trusted party and therefore it can compute the outputs [GKZ10]. Informally we say that a protocol is secure if running the protocol with a practical adversary is indistinguishable from running the protocol with the ideal process (using some trusted parties). In the Universal Composability (UC)-security framework, the programs run by the trusted party are referred to the ideal functionalities

[GKZ10]. More precisely, an ideal functionality is one which can satisfy both the correctness condition and the security conditions at the same time, with the latter in a way that the adversary with an infinite computing power can not win the corresponding game, which is called the "information-theoretic way" in cryptography. [BDD+06].

2.2 The Universal Composition (UC)-security framework

In most previous study on the ideal functionality [Can01, PW01, PSW00, BPW04, DKMR05, MRST01], although the protocol is a component of a polynomial bounded distributed system, we can always find the existence of the security of the protocol [Kus06]. These works attempt to analyze the protocols in a modular way. And one idea based on this method was first presented in [GMW87]. The idea is as follows: with the purpose of determining whether a given protocol is secure of a specific cryptographic task, firstly, we assume an ideal process which can perform this cryptographic task in a secure way. Then the security of a protocol can refer to an ideal process where all parties may forward their inputs to a trusted party. The trusted party here is to perform some computations correspondingly and to interact with the parties. Then it will return corresponding output to each party. Therefore, this ideal process is considered to be the component which formally specifies the necessary security conditions in a given cryptographic task. Informally, a protocol is said to be secure if the protocol can emulate the ideal process in the sense that we can convert every attack on the real protocol into an equivalent attack on the ideal protocol [Kus06].

The standard UC-model of protocol execution is presented in [Can98]. This framework is built on certain famous theoretical framework applied in the evaluation of function based on certain common tasks [PW01, DM00, Can06, PW94, Bea91, GL91]. It provides the evidence that is used to consider the security of the protocol in cryptographic tasks practically and it is an example of secure multi-party computation (MPC). MPC is aiming to enabling all parties to hand their inputs to the trusted party jointly while the trusted party keeps the input of each party separately at the same time.

There are two main components of the standard UC model: one is the parties in the protocol execution and the other one is the adversary. These components are modeled as the interactive Turing Machines (ITMs). Formally we have the following definition for the ITM in the UC-model:

Definition 2.1. The ITMs in the UC-model are probabilistic Turing Machines which are parameterized with some externally written tapes. Generally there are three types of the

externally written tapes:

- The tapes with the security parameter and the identity of the party running ITM.
- The written input tapes when ITM is first used.
- The communication tapes where the output is written.

And we also have the following definition for the systems of ITMs:

Definition 2.2. The system of ITMs is a set of ITMs. In the UC-model, the execution of the protocol proceeds from one configuration to the other configuration. In each configuration a single ITM $\mathcal{I}$ is activated and the output of $\mathcal{I}$ addresses to some other ITM $\mathcal{I}'$. When the output of $\mathcal{I}$ is written on some appropriate communication tape, $\mathcal{I}'$ will be activated.

The parties running the protocol are represented as a system of ITMs while the adversary is represented as a single ITM. During the execution of the protocol, a proportion of the parties can be selected to be controlled by the adversary. Further, the adversary also exerts an influence on the information transfer process, ensuring the communication to be synchronous. The adversary and the selected parties may have some interaction in a set of inputs, and then the parties may produce their separate local outputs. In [Can05], the concatenation of each party's local outputs and the adversary's are termed as the "global outputs". In order to evaluate some function f, each party transfer the inputs to a trusted party in the ideal process. The trusted party here is used to compute the functions and send the results to the specified parties. It is noteworthy that there is a limitation in the interaction between the adversary and the trusted party: the adversary can only interact with the trusted party with a form of the corrupted parties. [Can05].

Both the execution of the real protocol and the execution of the ideal process can be considered as probability distributions. First we want to given the definition of probability ensembles and the notion of indistinguishability. Originally it is presented in [Can05].

Definition 2.3. A distribution ensemble $\mathcal{X} = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$. The distribution $\{X(k,a)\}$ is associated with $a \in \{0,1\}^*$ where in the UC-framework, k is the security parameter and a is the input.

Definition 2.4. Two ensembles $\mathcal{X}$ and $\mathcal{Y}$ are called indistinguishable (denoted by $\cong$), if $\forall c, d$, $\exists k_0 \in \mathbf{N}$ such that $\forall k > k_0$ and $\forall a \in \cup_{\kappa\leq k^d\{0,1\}^\kappa}$, the following equation holds:

$$|Pr(X(k,a) = 1)Pr(Y(k,a) = 1)| < k^{-c}$$

Now we can say that it is secure for a protocol to evaluate a function if it is unable to distinguish the real protocol execution from the execution of using an ideal functionality (see figure 1).

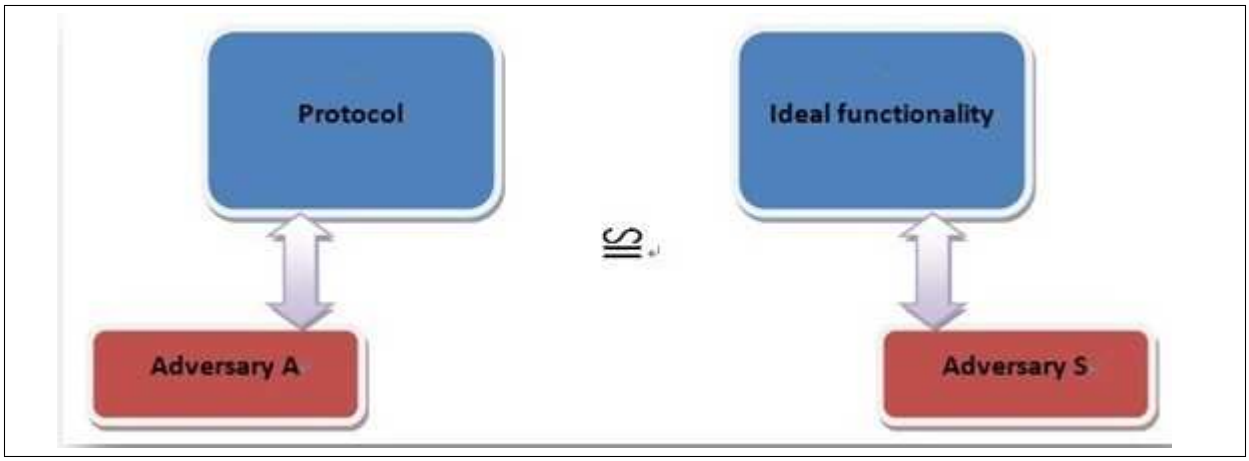

Figure 1: The execution of the real protocol and adversary $\mathcal{A}$ is indistinguishable from the execution of the ideal functionality and adversary $\mathcal{S}$.

More precisely, we have the following definition [Can05]:

Definition 2.5. Protocol π securely evaluates a function f if for any adversary $\mathcal{A}$ which interacts with the protocol there exists an ideal process adversary $\mathcal{S}$ such that for any set of inputs to the parties, the global output of running π with $\mathcal{A}$ is indistinguishable from the global output of the ideal process for f with adversary $\mathcal{S}$.

More precisely, we can say that if protocol π securely evaluates the function f, then the outputs generated by the parties running π are guaranteed to be indistinguishable from the outputs of f on the same set of the inputs. This means any information collected by the adversary which has an interaction with protocol π can be also collected by the adversary which can only get the inputs and outputs from the function f.

This definition is used to ensure the security under the non-concurrent composition situation[Can05]. If the instances of the protocols run concurrently, this definition no longer holds [Can05].

The Universal Composition(UC)-security framework captures the whole picture of the fundamental framework. Based on that, there is a new emergence of an algorithm entity. We call this entity the environment machine. Informally we can consider the environment machine as anything external to the current execution of the protocol. For instance it might be the executions of the other protocols, human users and so on. There are two

interactions between the environment machine and the execution of the protocol. The first one is that the environment machine selects certain random inputs and then transfers them to the adversary and the parties. The second one is that the environment machine then gets the outputs back from the adversary and the parties. At the end of the execution, the environment machine produces the output of a single bit, which can be used to determine whether it is the real protocol or the ideal process that the environment machine is considered to have interacted with.

Using the idea in the standard UC-model, similarly we can also infer that a protocol can make secure evaluation for a function if no environment machine can tell whether it has interacted with the real protocol or with the ideal functionality (see figure 2).

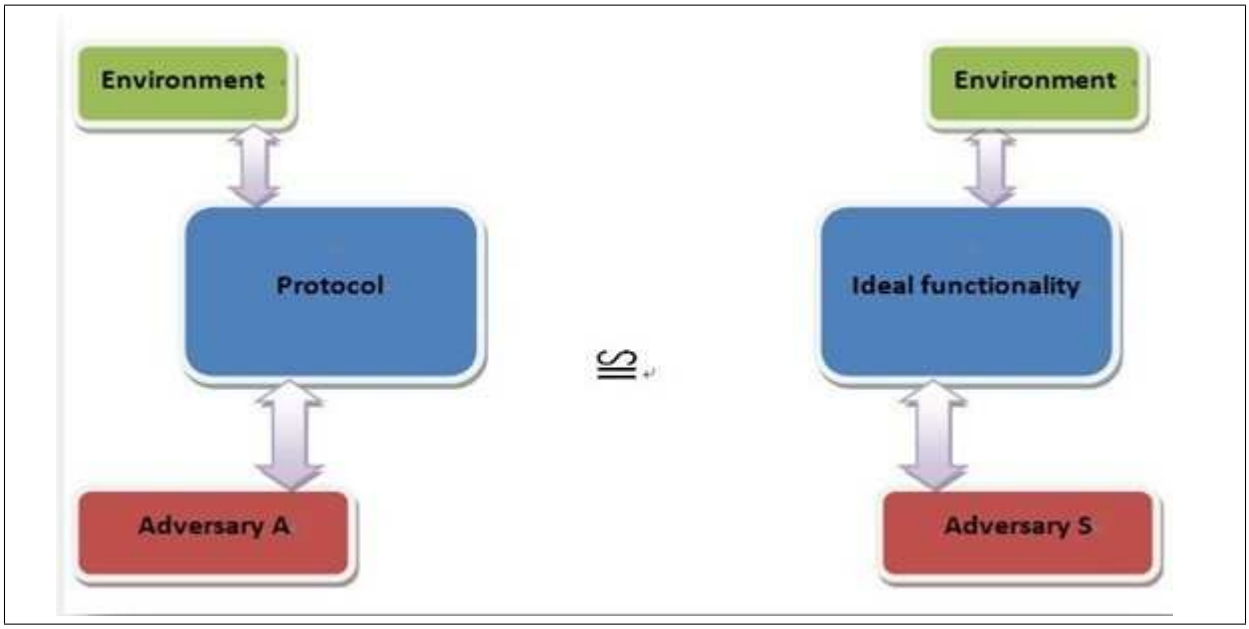

Figure 2: The environment machine can not tell whether it has interacted with the real protocol and the corresponding adversary $\mathcal{A}$ or with the ideal functionality and the corresponding adversary $\mathcal{S}$.

Before we proceed, we want to review a common definition in cryptography: the *negligible* function, informally we say a function is negligible if it decreases than the inverse of any polynomial function. More precisely we have the definition as follows:

Definition 2.6. The function f is called *negligible* if:

$$(\forall p)\ (\exists n_p \in \mathbb{N})\ (\forall n \geq n_p) \text{ we have } f(n) \leq \frac{1}{p(n)}$$

where $p(n)$ is any polynomial function.

Now we can present the following definition for function emulation [Can05]:

Definition 2.7. Protocol π securely evaluates a function f if for any adversary $\mathcal{A}$ which interacts with the protocol there exists an ideal process adversary $\mathcal{S}$ such that there exist no environment machine $\mathcal{Z}$ which can tell whether it has interacted with π and $\mathcal{A}$ or has interacted with the ideal process and $\mathcal{S}$ with non-negligible probability.

In the Universal Composition-security framework, it is allowed for the adversary and the environment machine to have the unrestricted interaction in the execution. Particularly, the information between the adversary and the environment machine is exchangeable after the parties which run the protocol have generated the messages or the outputs. We call this kind of environment machine the interactive environment machine [Can05]. The following definition is for the notion "UC-realize" [Can05]:

Definition 2.8. If protocol π securely realizes function f with respect to the interactive environment machine, then protocol π UC-realizes function f.

We can apply the same idea to a more general notion, we call it the protocol emulation. Informally, we can assume there are two protocols: protocol π and protocol ϕ. We say that protocol π emulates protocol ϕ if for any environment machine $\mathcal{Z}$, $\mathcal{Z}$ considers protocol π to be the same as the protocol ϕ. It means that the interaction between protocol π and the adversary $\mathcal{A}$ can not be distinguished from the interaction between the protocol ϕ and another adversary $\mathcal{S}$ [Can05]. The following definition will give a more precise explanation for protocol emulation [Can05]:

Definition 2.9. Protocol π UC-emulates protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$, the difference between the probability of $\mathcal{Z}$ outputting 1 after interacting with $\mathcal{A}$ and parties running π on any inputs and the probability of $\mathcal{Z}$ outputting 1 after interacting with $\mathcal{S}$ and ϕ on the same set of inputs should be negligible.

The adversary $\mathcal{S}$ is often called the simulator. Informally, we can say protocol π securely evaluates a functionality $\mathcal{F}$ if there is always a simulator $\mathcal{S}$ for any adversary $\mathcal{A}$ which makes any environment machine $\mathcal{Z}$ unable to differentiate whether it has interacted with protocol π and the corresponding adversary $\mathcal{A}$ or it has interacted with the ideal functionality $\mathcal{F}$ and the corresponding simulator $\mathcal{S}$. The idea of realizing an ideal functionality can be defined as follows [Can05]:

Definition 2.10. Protocol π UC-realizes an ideal functionality $\mathcal{F}$ if π emulates $IDEAL_{\mathcal{F}}$ where $IDEAL_{\mathcal{F}}$ is the ideal protocol for $\mathcal{F}$.

As it is showed in many work such as [Can98], if protocol π UC-realizes an ideal functionality $\mathcal{F}$, then it is guaranteed that for the same set of the inputs, the outputs generated by the parties running π are indistinguishable from the outputs generated by the ideal functionality $\mathcal{F}$. That is to say, any information which is revealed by the adversary interacting with protocol π is revealed by the adversary interacting with the ideal functionality $\mathcal{F}$ as well.

2.3 The Universal Composition theorem

One major finding in the Universal Composition-security framework is called the Universal Composition theorem highlighted by Canetti [Can05]. First of all, according to his theorem, a protocol composition is required. Then we may constitute a combined protocol composed of two protocols in a way as follows: using the above definition, we consider a protocol ρ such that ρ UC-realizes an ideal functionality $\mathcal{F}$. Then, consider the arbitrary protocol π as a high level protocol. The parties running protocol π ideally call various instances of $\mathcal{F}$ as well as the normal interactions. Therefore protocol π is considered as a hybrid protocol which consists of the forms of various instances of the ideal functionality $\mathcal{F}$ and the normal communications. Thus we call protocol π $\mathcal{F}$-hybrid protocol. There is no global coordination of the execution of different instances of $\mathcal{F}$ at the same time. Protocol π may generate some special session identifiers in order to distinguish the different instances of $\mathcal{F}$ [Can05].

We can then build the composed protocol π^{ρ} using protocol π as follows: we replace the calls made to new instances of $\mathcal{F}$ with the calls to the new instances of ρ. In this way, we can also replace the inputs to the current instance of $\mathcal{F}$ with a given message of the corresponding instance of $\mathcal{F}$. Therefore the output value of the instances of protocol ρ could be considered as the inputs received from the corresponding instances of the ideal functionality $\mathcal{F}$. It is worth mentioning that a variety of instances of $\mathcal{F}$ can be unrestrictedly used by the protocol π simultaneously, as a result, a variety of instances of ρ can run on those relevant messages of protocol π^{ρ} concurrently at the same time. Informally, the Universal Composition theorem says that to run the protocol π^{ρ} without accessing to the ideal functionality $\mathcal{F}$ is of no difference from running the $\mathcal{F}$-hybrid protocol π[Can05].

Formally, the composition process can be defined by using an operator on protocols. We call this operator the universal composition operator $UC()$. We have the following

definition for $UC()$ [Can05].

Definition 2.11. Given a protocol ϕ, a protocol π which makes subroutine calls to ϕ and a protocol ρ which UC-emulates ϕ. The composed protocol $\pi^{\rho/\phi} = UC(\pi, \rho, \phi)$ is identical to protocol π. There are two possible modifications:

1. If the protocol π contains an instruction to pass input x to an ITI (an instance of ITM (interactive Turing Machine)) which is running ϕ with identity (sid, pid), then the composed protoco $\pi^{\rho/\phi}$ contains another instruction to pass input x to an ITI which is running ρ with identity (sid, pid).

2. If the composed protocol $\pi^{\rho/\phi}$ receives an output which is passed from $\rho_{(sid,pid')}$ (an ITI running ρ with identity (sid, pid')), then it proceeds as the protocol π proceeds when it receives an output passed from $\phi_{(sid,pid')}$.

One special case of the above definition is that if the protocol ϕ is the ideal protocol $IDEAL_{\mathcal{F}}$ for some ideal functionality $\mathcal{F}$, then the resulting composed protocol can be represented as $\pi^{\rho/\mathcal{F}}$ [Can05].

Now we can state the Universal Composition theorem formally [Can05]. It says if protocol ρ UC-emulates protocol ϕ, then the composed protocol $\pi^{\rho/\phi}$ UC-emulates protocol π for any protocol π.

Theorem 2.1. *Let π, ρ, ϕ be Probabilistic Polynomial Time multi-party protocols such that ρ UC-emulates ϕ and both ϕ and ρ are subroutines. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol π. There are two special cases:*

- *Let π, ρ be Probabilistic Polynomial Time protocols such that ρ UC-realizes a Probabilistic Polynomial Time ideal functionality $\mathcal{F}$ and both ϕ and ρ are subroutines. Then protocol $\pi^{\rho/\mathcal{F}}$ UC-emulates protocol π.*

- *Let $\mathcal{F}$, $\mathcal{G}$ be the ideal functionalities such that $\mathcal{F}$ is Probabilistic Polynomial Time. Let π be a subroutine respecting protocol UC-realizes $\mathcal{G}$, let ρ be a subroutine respecting protocol that securely realizes $\mathcal{F}$. Then the composed protocol $\pi^{\rho/\mathcal{F}}$ securely realizes $\mathcal{G}$.*

The Universal Composition theorem highlighted that for any adversary $\mathcal{A}$ it is guaranteed to have the corresponding adversary $\mathcal{A}_{\mathcal{F}}$ such that for any environment machine $\mathcal{Z}$, $\mathcal{Z}$ can only distinguish whether it has interacted with parties which run protocol π^{ρ} and

adversary $\mathcal{A}$ or it has interacted with parties which run protocol π and adversary $\mathcal{A}_{\mathcal{F}}$ with negligible probability [Can05].

The UC theorem is helpful as it ensures the security of protocols when being composed with the use of other protocols. This is a strong guarantee since it holds regardless of the environment machine and the interactions between the main protocol and its subroutines. Also, protocols which satisfy the Universal Composition definition are guaranteed to be secure in other protocol environment. This guarantee can be applied in the area where the protocols run in a complicated and unpredictable situations such as the contemporary network communication. Furthermore, it allows for the modular design of protocols since we can always divide the security tasks into some subtasks and then design the sub protocols to perform such tasks.

2.4 Extending the original Universal Composition-security framework

There are some problems of the ideal functionalities because of their general nature. Recent work on the relationship between cryptographic games and UC-security framework [GKZ10] shows that it is cumbersome to define the ideal functionalities and also it is not possible for the ideal functionalities to specific the desired security property of a given cryptographic task every time. Therefore several extensions are proposed [GKZ10, Kus06] in order to fix the weakness of the ideal functionalities.

2.4.1 The canonical ideal functionalities

In the original Universal Composition-security framework [Can05], the ideal functionalities are not structured in a consistent way. The disadvantage of this is that there are multiple ideal functionalities with the same property for a given cryptographic task. As showed in some work on digital signatures [BH03, Can04], it can make the analysis of the cryptographic task complex and error prone. Also as presented in one recent work [GKZ10], the ideal functionalities are used as a single component for specifying the security requirements of a given cryptographic task therefore they are unstable since the whole ideal process can be effected by a single small change. On the same work [GKZ10] they propose a class of the ideal functionalities called the canonical ideal functionalities. It provides some simple rules which can easily structure the ideal functionalities and by adding some restrictions we can get a unique ideal functionality for any given cryptographic task. Without the loss

of generality, we are going to use the canonical ideal functionalities for our proof in this work.

2.4.2 Inexhaustible interactive Turing machines

Another work [Kus06] present a simpler protocol execution framework. The order of the executions of the parties (which are represented as the interactive Turing machines) are provided in a more structured way. In the original Universal Composition-security framework [Can05], a centralized mapping controls the order of the execution of the parties while in this model, the order of the execution of the parties only depends on the code of the parties. More precisely, in this model, the last activated party determines which party is going to be activated next. This is not always the case in the original Universal Composition-security framework [Kus06]. The advantage of this is that it can simplifies the proof of the security of the protocols.

3 Cryptographic games

3.1 General cryptographic games

Usually, the security of some cryptographic primitives can be represented in terms of using an attack game. There are usually two players of the game: the adversary and the challenger [BR04]. The basic model is presented in [BR04] and the idea is as follows: we represent the attack game $\mathcal{G}$ as a set of programs where each program can be written in different programming languages. There is also a common set of static variables and global variables for each program. Consider the following game $\mathcal{G}$ with the adversary $\mathcal{A}$: $\mathcal{A}$ can run game $\mathcal{G}$ by calling out to the provided programs (see figure 3)

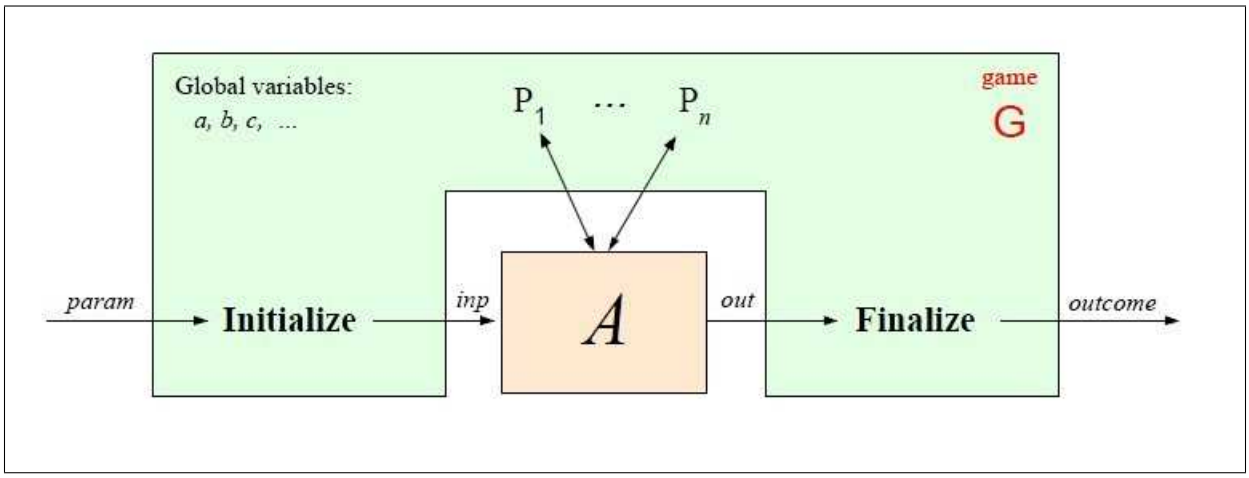

Figure 3: A simple cryptographic game $\mathcal{G}$ with the adversary $\mathcal{A}$. Source:[BR04]

The game $\mathcal{G}$ in the above figure is represented by the box which surrounds $\mathcal{A}$. $\mathcal{G}$ consists of several programs such as $Initialize, P_1, \cdots, P_n$ and $Finalize$. $\mathcal{A}$ then receives an input from the game. It produces the output after interacting with its oracle $P_1, \cdots, P_n$. The result of the game is determined by the $Finalize$ program.

Usually, the cryptographic game is formulated as a probability space because both the adversary and the challenger are the probabilistic processes which can interact with each other [Sho04]. Usually the definition of the security is represented in terms of some particular event $\mathcal{E}$. A cryptographic primitive is then considered to be secure if for any efficient adversary (means the adversary runs in polynomial time), the probability that the event $\mathcal{E}$ happens is negligibly close to certain target probability. Usually it is 0 or 0.5, or the probability of some other event $\mathcal{T}$ such that the same adversary is playing some other attack game with a different challenger [Sho04].

3.2 The commitment games

Many recent work [CF01, BDD+06] on the impossibility are based on the commitment scheme. Therefore we are going to talk about the commitment scheme in general first and the corresponding hiding game and binding game.

3.2.1 The commitment scheme

One of the most commonly used cryptographic primitives is the commitment scheme. It is the useful construction block in many cryptographic protocols like the Zero-Knowledge protocols, electronic commerce, etc [Dam99, GMW91, BCC88, Dam90]. The idea behind it is as the follows: a digital equivalent of a sealed envelope which contains a value x is provided by a party called the committer. The committer then sends this envelope to another party called the receiver. After the envelope is sent, the value inside can not be changed by the committer. The receiver can not get any information about x from the envelope until the committer asks the receiver to open the envelope. The value x can be recovered fully when both two parties agree with each other [CF01]. Informally we say that a good commitment scheme should have the following two properties [Dam99]:

- The commitment can not reveal any information about the original message.
- The decommitment can be verified in order to determine the relationship between the commitment and the original message.

We refer the first property to the hiding property and the second property to the binding property. We will discuss them later.

We use a simple model presented in [Dam99] to describe the syntax of a two-party commitment protocol. The two parties are $\mathcal{P}$ and $\mathcal{V}$. There are the following three steps corresponding to the protocol execution.

1. $\mathcal{P}$ chooses a message m and then writes down m on a piece of paper. After that he puts the paper into a sealed envelope.
2. $\mathcal{P}$ then sends the envelope to $\mathcal{V}$.
3. $\mathcal{P}$ can provide $\mathcal{V}$ with a method to open the envelope whenever he wants.

More precisely, we say that the commitment scheme can be divided into two phases, the commit phase and the decommit phase:

- The commit phase:
  - The sender selects a message m and then run some algorithm on m to produce a commitment on m: c and its decommit information: d.
  - The sender may send this tuple (c, d, m) to the receiver afterwards.
- The decommit phase:
  - The decommitment is achieved by releasing the decommitment value d.
  - The receiver then can verify if the message m matches the commitment c by using a verification process: $\phi = verify(c, d, m)$:
    - If the commitment c matches the message m then output $\phi = 1$.
    - If the commitment c does not match the message m then output $\phi = 0$.

As we described above, there should be two properties for a good commitment scheme. More precisely, using our simple protocol model above, we can have the following descriptions for the two properties [Dam99]:

- There is no way for $\mathcal{V}$ to tell about what is inside the envelope before $\mathcal{P}$ provides a method to open it. This property is called *hiding property*. In other words, we say that it is impossible to determine the committed message m until decommit it. There are two types of the hiding property:
  - *computational hiding*: meaning $\mathcal{V}$ is polynomially bounded and it is impossible for $\mathcal{V}$ to determine the message inside the envelope. In other words, there is no efficient way to determine the message.
  - *perfectly hiding*: meaning $\mathcal{V}$ has an infinite computing power and it is still not possible for $\mathcal{V}$ to determine the message inside the envelope. More formally, it requires the distributions of the commitment on all the messages to be the same and therefore the commitment reveals no information about the message.
- After sending out the envelope, $\mathcal{P}$ can not change anything inside it. Therefore, we can know that what is recovered is actually the message which $\mathcal{P}$ chose at the first time after the envelope is opened. This property is called *binding property*. In other words, we say that it is impossible to have two different messages with the same commitment c. Similarly, there are two types of the binding property:

– *computational binding*: meaning $\mathcal{P}$ is polynomially bounded and it is impossible for $\mathcal{P}$ to change the committed value after sending it away. In other words, the chance of having the two different messages with the same commitment value is small.

– *perfectly binding*: meaning $\mathcal{P}$ has an infinite computing power and it is still impossible for $\mathcal{P}$ to change the committed value after sending it away. In other words, the probability of having the two different messages with the same commitment value is exactly 0.

General, there are two distinct basic flavors of commitment scheme have been formalized. One is called the unconditionally secret commitment protocols and another one is called the unconditionally binding commitment protocols [CF01]. However, it turns out that it is inadequate to have the basic definitions in some situations while there are some stronger variants which can allow the secure composition of the commitment protocols in a way both calling the protocol and using other invocations of the protocol. The examples are the chameleon commitments [BCC88], the trapdoor commitments [FS90], the equivocable commitments [Bea96] and non-malleable commitments [DDN91].

3.2.2 The commitment games

We are now going to present two games of the commitment scheme, corresponding to its hiding property and binding property. These two games follow the standard description of the commitment scheme like [Dam99, BCC88, Nao89]. We are going to present the more general hiding game and binding game in the later section.

The hiding game for commitment scheme We give a hinding game for the commitment scheme as follows (see figure 4): there are two parties here: the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ has the access to the *commit* oracle. The adversary $\mathcal{A}$ sends two messages (m_0, m_1) to the challenger $\mathcal{C}$ first. Then $\mathcal{C}$ will select a bit b from $\{0, 1\}$ in random and then send m_b to the *commit* oracle. The oracle will commit the message m_b and send the commitment of message c along with the decommitment information d to $\mathcal{C}$. Then $\mathcal{C}$ will parse the pair (c, d) to $\mathcal{A}$ and then $\mathcal{A}$ will output b^*. The adversary $\mathcal{A}$ wins the game if the probability of its output $b^* = b$ is significantly better than 0.5 (the probability of randomly guessing by tossing a coin).

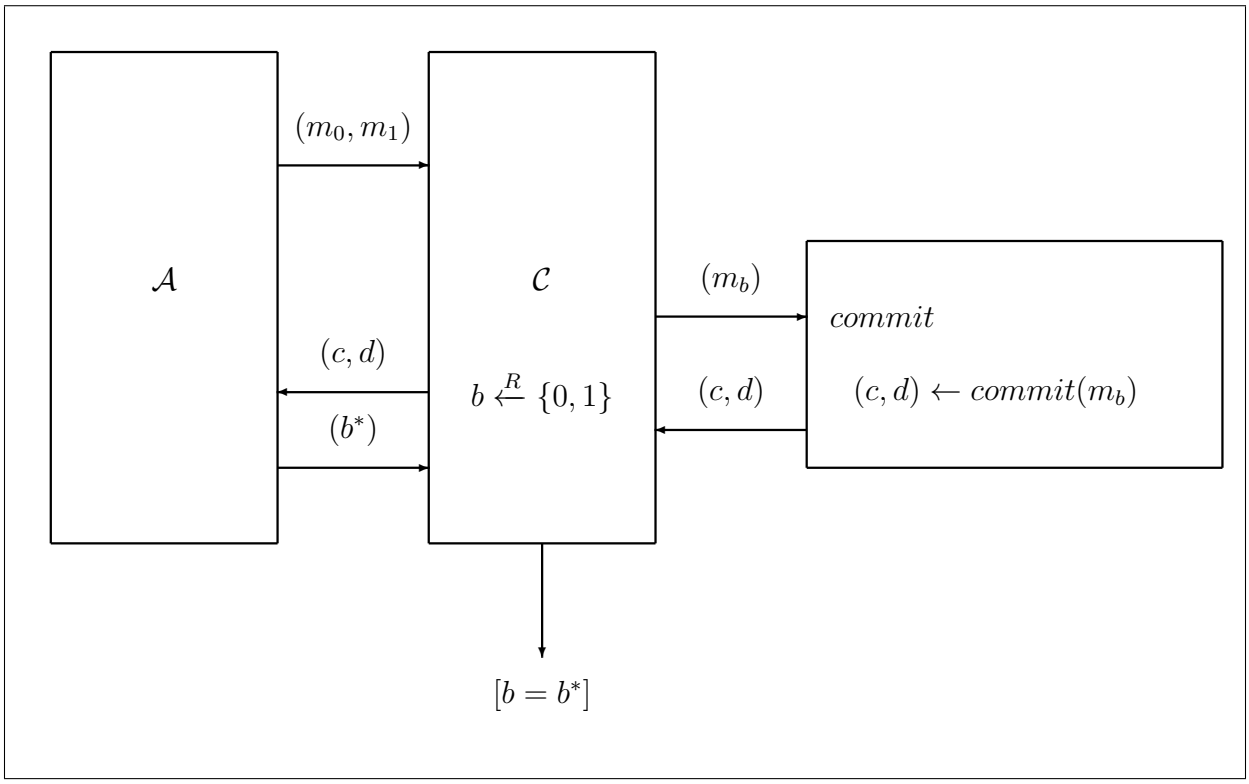

Figure 4: The hinding game for commitment scheme.

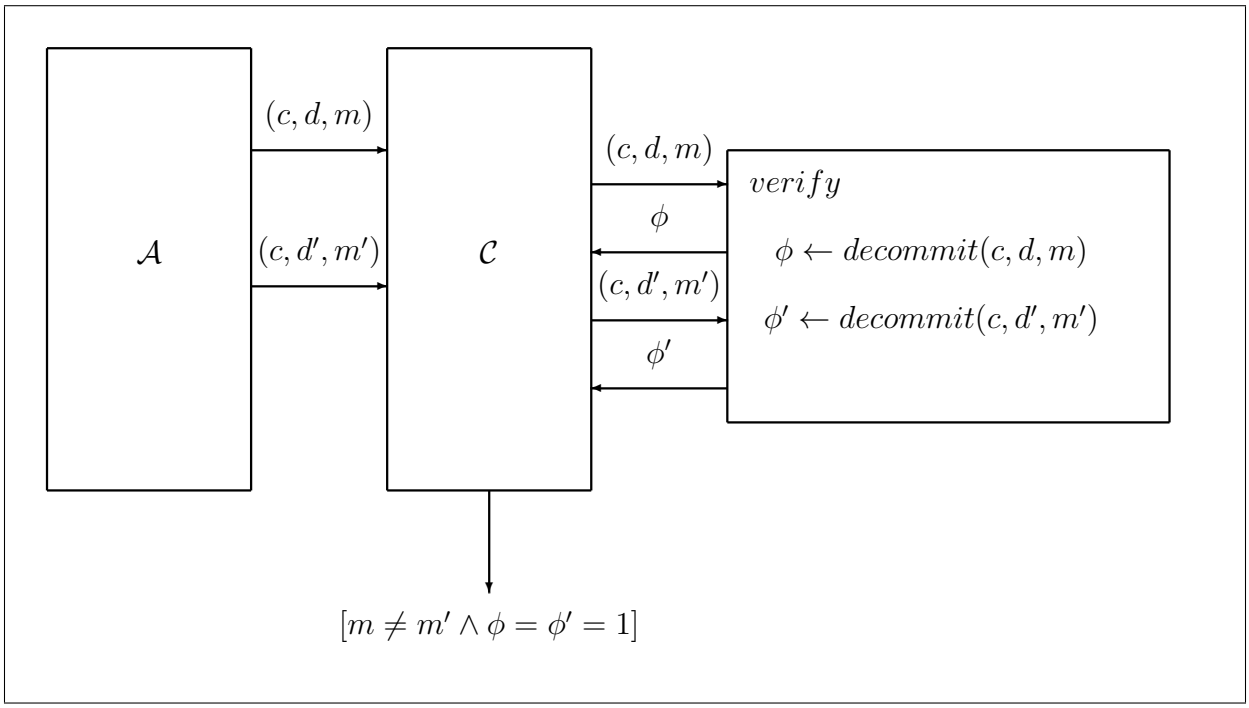

Figure 5: The binding game for commitment scheme.

The binding game for commitment scheme We give a binding game for the commitment scheme as follows (see figure 5): similarly there are two parties here, the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$. This time the challenger $\mathcal{C}$ has the access to the $verify$ oracle to check if the message matches the commitment. The adversary $\mathcal{A}$ sends two tuple including two messages m and m' and their corresponding commitment and decommitment messages (c, d) and (c', d') to the challenger $\mathcal{C}$. Then $\mathcal{C}$ will use the $verify$ oracle to check the commitment and decommitment information. The adversary $\mathcal{A}$ wins the game if $[m \neq m' \wedge \phi = \phi' = 1]$. In other words, adversary $\mathcal{A}$ produces two different valid messages with the same commitment value.

4 Cryptographic games and Universal Composition

Both the Universal Composition-security framework and the cryptographic games can provide the methods to define the cryptographic task of certain properties in a way that complement each other. The UC approach is useful since the composable notions of security can be implied by the indistinguishability property of an ideal functionality. However the complexity of the analysis may increase. On the other hand, the game based definition is simpler and easier to analyze in terms of the computational complexity arguments. The problem is that if there is a paradigm which can satisfy one game specification, it is not clear what the response of such a paradigm to the unexpected interactions of that game is. There are many work on the relationship between these two approaches [CF01, BDD+06, CHK+05]. We are going to introduce a work on the commitment scheme [CF01]. Then we will use a methodology presented in [GKZ10] to show how to derive the canonical ideal functionalities from the games. After that we may introduce our work on the general hiding game and general binding game.

4.1 The commitment functionalities

We pick three typical examples presented in [CF01].

4.1.1 The common reference string (CRS) model

There is a high security requirement for the commitment scheme protocol [CF01]. Canetti et al [CF01] proved that there exists no Universal Composition commitment protocol which only has the committer and the receiver in the standard computational model (without using setup assumptions). However, things are much better in the common reference string (CRS) model. The common reference string (CRS) model is a general form of the common random string (CRT) model. In CRS model, there is a common string crs which is chosen according to some distribution D ahead of time. crs should be available to all parties involved in the protocol execution before any interactions start.

The CRS model can be referred as a composition model, which has ideal access to a functionality $\mathcal{F}_{CRS}$. $\mathcal{F}_{CRS}$ can be parameterized by a distribution D (see figure 6) [CF01]

.

The following two properties of the CRS model using this formalization are presented in [CF01]:

Functionality $\mathcal{F}_{CRS}$

$\mathcal{F}_{CRS}$ is parameterized by a distribution D and it proceeds as follows:

1. When activated on input $(value, sid)$ for the first time, a value $d \xleftarrow{R} D$ is chosen and d is sent back to the activating party. In each other activation, d is returned to the activating party.

Figure 6: The CRS functionality. Source: [CF01]

1. The parties have access to a common public string which is chosen in advance according to some distribution D in the real life model of computation. D is is specified by the parties running the protocol.

2. The random string is not used anywhere in the ideal process for some functionality. Therefore an adversary which play the role of $\mathcal{F}_{CRS}$ may be simulated in order to run the ideal process adversary. In other words, the ideal process adversary can chose whatever common string it likes.

4.1.2 The ideal functionality for a single commitment

Functionality $\mathcal{F}_{COM}$

$\mathcal{F}_{COM}$ is running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$ and it proceeds as follows:

1. Upon receiving a value $(commit, sid, P_i, P_j, b)$ from P_i. Record the value b and send the message $(recepit, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent *commit* messages.

2. Upon receiving a value $(open, sid, P_i, P_j)$ from P_i proceeds as follows:

   - If some value b was previously recorded, send the message $(open, sid, P_i, P_j, b)$ to P_j and $\mathcal{S}$. Halt.
   - Halt otherwise.

Figure 7: The ideal functionality for a single commitment. Source: [CF01]

We present a functionality $\mathcal{F}_{COM}$ which can manage a single commitment and decommitment process [CF01]. As shown in figure 7, the commitment phase can be modeled as follows: $\mathcal{F}_{COM}$ receives the value $(commit, sid, P_i, P_j, b)$ from the committer P_i where sid is the session identifier utilized to differentiate different versions of $\mathcal{F}_{COM}$, P_j represents

the receiver and b is the value which is committed to. Then $\mathcal{F}_{COM}$ delivers the message $(receipt, sid, P_i, P_j)$ to the receiver P_j and the adversary $\mathcal{S}$ in order to tell P_j and $\mathcal{S}$ that a value has been committed to with the session identifier sid by the committer P_i. Then the committer P_i may send a value $(open, sid, P_i, P_j)$ to $\mathcal{F}_{COM}$ in order to initiate the decommitment phase. $\mathcal{F}_{COM}$ may send the $(open, sid, P_i, P_j, b)$ to the receiver P_j and the adversary $\mathcal{S}$ in response [CF01].

4.1.3 The ideal functionality for multiple commitments

Functionality $\mathcal{F}_{MCOM}$

$\mathcal{F}_{MCOM}$ is running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$ and it proceeds as follows:

1. Upon receiving a value $(commit, sid, cid, P_i, P_j, b)$ from P_i. Record the tuple (cid, P_i, P_j, b) and send the message $(recepit, sid, cid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent $(commit, sid, cid, P_i, P_j, \ldots)$ values.
2. Upon receiving a value $(open, sid, cid, P_i, P_j)$ from P_i proceeds as follows:
   - If the tuple (cid, P_i, P_j, b) was previously recorded, send the message $(open, sid, cid, P_i, P_j, b)$ to P_j and $\mathcal{S}$.
   - Do nothing otherwise.

Figure 8: The ideal functionality for multiple commitments. Source: [CF01]

We present a functionality $\mathcal{F}_{MCOM}$ which handles multiple commitment and decommitment processes [CF01]. As shown in figure 8, $\mathcal{F}_{MCOM}$ follows the similar operation from $\mathcal{F}_{COM}$. $\mathcal{F}_{MCOM}$ uses another identifier cid which represents the commitment ID as well as using the session identifier sid. cid can help to identify different commitments which occur in the same single run of $\mathcal{F}_{MCOM}$. Therefore, cid, the identifier of the committer P_i and the identifier for the receiver P_j need to be recorded for a single commitment value. It is not allowed for the two commitments with the same committer and receiver to have the same cid. The requests $commit$ and $open$ can come with different orders (i.e. they do not need to follow the specified order such as $(commit, open)$). Also for the same receiver, a commitment is allowed to open many times by the committer [CF01].

4.2 The Universal Composition commitment protocol

The recent study [CF01] developed a new way to quantify the security of the commitment protocols. It can guarantee the hiding property and the binding property of the commitment protocols even when they are composed with an arbitrary set of protocols in a concurrent way. We call these protocols the Universal Composition commitment protocol. More precisely, we apply the Universal Composition-security framework to the commitment protocols as follows: the functionality $\mathcal{F}_{COM}$ we described above should capture any expected behavior of the ideal commitment service. Then the Universal Composition commitment protocol are the protocols that can make secure realization of the ideal functionality $\mathcal{F}_{COM}$. Therefore we can have the following definition [CF01].

Definition 4.1. A protocol is a universally composable (UC) commitment protocol if it securely realizes functionality $\mathcal{F}_{COM}$. If the protocol securely realizes the functionality $\mathcal{F}_{MCOM}$ then it is called a reusable-CRS UC commitment protocol.

Based on the Universal Composition theorem, we can see that no matter what tasks the calling protocol is involved in, the run of a universally composable commitment protocol π is indistinguishable from its interaction with the similar quantity of copies of the ideal functionality $\mathcal{F}_{COM}$.

4.3 Derive the canonical ideal functionality from the cryptographic games

We now use the methodology presented in [GKZ10] to derive the canonical ideal functionalities from the game conditions.

4.3.1 The communication languages

We now present two communication languages for the ideal functionalities. One is called *party-functionality language* and the other one is called *the extended language of the functionality*. For the convenience of our presentation, we assume the functionality is only able to perform a finite number of different actions. For each action there should be an ordered list of the parties and some input. According to the way of how the functionality is instructed (i.e. by the adversary or by the parties), the actions can be parameterized and split into two distinct languages. The corruption of the parties by the adversary is not considered here in order to simplify the results.

Party-functionality language

- A party can ask a functionality to perform an action using a tuple $(\text{ACTION}, \mathbf{P}, x)$ where ACTION is the action like encryption, commit, etc, $\mathbf{P}$ is the list of parties involved in the action and x is the input of the action.
- For each action the functionality will send a tuple $(\text{ACTIONRETURN}, \mathbf{P}, y)$ to some parties in return where ACTIONRETURN is the return action of ACTION like decryption, decommit, etc, $\mathbf{P}$ is the list of parties involved in the action and y the output of the action.
- Therefore the I/O language of the functionality consists of the set of all possible tuples of above two forms.
- We can denote the party-functionality language of a given task $\mathcal{T}$ by $\Sigma_{\mathcal{T}}$.

The extended language of the functionality

- Each time when the ideal functionality receives s tuple $(\text{ACTION}, \mathbf{P}, x)$ from a party, it will send another tuple $(\text{LEAKACTION}, \mathbf{P}, x')$ to the adversary where LEAKACTION is the action of the ideal functionality perform in order to notify the ideal adversary for every ACTION, $\mathbf{P}$ is the list of parties involved in the action and x' is dependent on x according to the program of the ideal functionality.
- Upon receiving such a tuple, an output tuple $(\text{INFLACTION}, \mathbf{P}', y)$ will be generated and sent it back to the ideal functionality by the adversary where INFLACTION is the communication occurs between the ideal functionality and the ideal adversary, $\mathbf{P}'$ is the list of parties involved in the communication and y is the output value.
- Therefore the I/O language of the adversary-functionality language consists of the set of all possible tuples exchanged during the communication between the adversary and the ideal functionality.
- We call the union of the I/O language of the party-functionality and the I/O language of the adversary-functionality the extended language of the functionality.
- We can denote the extended language of a given task $\mathcal{T}$ by $\Sigma_{\mathcal{T}}^{ext}$.

4.3.2 The canonical functionality of a given cryptographic task

Now we present our canonical functionality of a given cryptographic task. First, we may make some assumptions regarding the communication of the ideal functionality. There are two assumptions:

1. For all input tuples $(\text{ACTION}, \mathbf{P}, x)$ of the functionality, the ideal adversary will be notified by a tuple $(\text{LEAKACTION}, \mathbf{P}, x')$.

2. The return action $(\text{ACTIONRETURN}, \mathbf{P}, y)$ of the tuple $(\text{ACTION}, \mathbf{P}, x)$ will be performed only after the influence tuple $(\text{INFLACTION}, \mathbf{P}', y)$ is returned by the adversary.

There is a common algorithm procedure presented in [GKZ10] to derive the canonical ideal functionalities. A set of predicates and functions is used to capture the different operations of the functionalities in a given cryptographic task. The functionality will keep a record of the messages exchanged with the parties during the protocol execution. It will also keep a record of which party may be influenced by the adversary. There are two main functions used in the canonical functionalities, one is called *the suppress function* and the other one is called *the validate function*.

The suppress function

- For each input message tuple $(\text{ACTION}, \mathbf{P}, x)$ of the functionality, the simulator will be notified by another tuple $(\text{LEAKACTION}, \mathbf{P}, x')$ by the adversary.

- x' is dependent on the suppress function where x' is x with the symbols in the same position replaced with "$-$", no matter what value of x is. "$-$" is a special character which does not occur anywhere else in the alphabet of x.

- The suppress functionality is independent from the history. It captures the hiding property of a given cryptographic task since it can put the restrictions on the information can be gained of each message by the adversary.

The validate function

- For each influence tuple $(\text{INFLACTION}, \mathbf{P}', y)$ of the functionality, the output tuple $(\text{ACTIONRETURN}, \mathbf{P}', y')$ will be generated and sent back to the corresponding parties.

- The validate function will check the validity of the output at the time of its execution for the functionality. More precisely, when a new message is added to the history of the functionality, it can check if the resulting history is still valid in terms of some restrictions.

- The valid function captures the binding property of a given cryptographic task since it can guarantee some invalid events will not happen during the protocol execution.

Additional, three more functions are needed in order to let the functionality work accurately. These three functions are *The public output predicate* (PO_t), *The secret output predicate* (SO_t) and *The well formatness predicate* (WF_t).

The public output predicate It is used to make sure that the adversary can gain some information about the internal state of the functionality in order to enable the informed influence of the adversary.

The secret output predicate It is used to keep the output value which is not necessarily known by the adversary.

The well formatness predicate It is used to check if the order of the actions is sensible with respect to the ideal functionalities. The unexpected actions are simply ignored.

Now we are ready to present the canonical functionality $\mathcal{F}_T^{\text{suppress,validate}}$ (see figure 9).

4.4 The game satisfaction

A concept such as encryption, bit-commitment or digital signature can be defined using a game condition. There might more than one possible ideal functionalities corresponding to each game condition. By using the method describe above, we can get a canonical functionality.

As an ideal functionality is supposed to be secure because of its construction, therefore it has to meet the specific requirement of the given game in an information-theoretic way (the adversary is unbounded) instead of in a computational complexity way (the adversary is polynomial bounded). We introduce a new notion "$\models$" to represent the "satisfy" here: if a functionality $\mathcal{F}$ satisfy a game $\mathcal{G}$, we can write it as $\mathcal{F} \models \mathcal{G}$. Intuitively, $\mathcal{G}$ can be considered as a game machine which receives the action query from adversary and forward

Canonical functionality $\mathcal{F}_T^{\text{suppress,validate}}$

1. Initially: $\mathsf{history} := \varepsilon, \mathsf{binding} := \varepsilon$.

2. **Upon receiving:** $msg = (\text{ACTION}, \mathbf{P}, m)$ from some party P_i:
   $x' \leftarrow \text{suppress}(msg), msg' \leftarrow (\text{LEAKACTION}, \mathbf{P}, x')$

   - **if** $\text{WF}_t(\mathsf{history}_{P_i}, msg) = 1$ **then**
     - **send** $(msg', \text{PO}_t(\mathsf{history}, msg))$ to the simulator $\mathcal{S}$
     - $\mathsf{history} \leftarrow \mathsf{history} \| msg$
     - $\mathsf{binding}[|\mathsf{history}|] \leftarrow \{P_i\}$
   - **else if** $\text{WF}_t(\mathsf{history}_{P_i}, msg) = 0$ **then** ignore msg

3. **Upon receiving:** $msg = (\text{INFLACTION}, \mathbf{P}, y')$ from the simulator $\mathcal{S}$:
   infer P_i from the list of parties $\mathbf{P}$ where P_i is the first party receives the output
   $y \leftarrow y' \| \text{SO}_t(\mathsf{history}, msg), msg' \leftarrow (\text{ACTIONRETURN}, \mathbf{P}, y)$

   - **if** $\text{WF}_t(\mathsf{history}_{P_i}, msg') = 1$ **then**
     - **if** $\text{validate}(\mathsf{history} \| msg') = 1$ **then**
       - **send** msg' to P_i
       - $\mathsf{history} \leftarrow \mathsf{history} \| msg'$
       - $\text{binding}[k] \leftarrow \text{binding}[k] \cup \{P_i\}$ for all $1 \leq k \leq |\mathsf{history}|$
     - **else if** $\text{validate}(\mathsf{history} \| msg') = 0$ **then send** an error message to P_i and **halt**
   - **else if** $\text{WF}_t(\mathsf{history}_{P_i}, msg') = 0$ **then** ignore msg

Figure 9: Canonical functionality $\mathcal{F}_T^{\text{suppress,validate}}$ Source: [GKZ10]

it to the functionality. It may select the message sent by the adversary and send it to the functionality or it may just forward all the messages sent by the adversary to the functionality. After the functionality perform the required action, the return value will be sent to $\mathcal{G}$ and $\mathcal{G}$ will forward it to the adversary. There might be several iterations of such interactions during the game execution. At the end of the game, a predicate in $\mathcal{G}$ will determine whether the adversary wins the game or not. The predicate will output 1 if the adversary wins and 0 otherwise.

We have the following definition for the game satisfaction:

Definition 4.2. Given a game $\mathcal{G}$ and an ideal functionality $\mathcal{F}$, we say that $\mathcal{F} \models \mathcal{G}$ if $\forall$

simulator $\mathcal{S}$ that $\mathcal{F}$ uses, the adversary $\mathcal{A}$ can not win the game $\mathcal{G}$ no matter what $\mathcal{S}$ does.

Note for the above definition, the probability of the adversary $\mathcal{A}$ winning the game $\mathcal{G}$ is exactly 0. For instance consider the commitment games we described previously, the probability of the adversary outputting the correct bit b is exactly 0.5 for the hiding game and the probability of the adversary producing two different valid messages with the same commitment is exactly 0 for the binding game.

4.5 The general hiding game and the general binding game

We give a description of protocols that during their operation it allows actions on the value stored in a given variable. We assume that each variable is a single bit and that there are countable many of them. We represent each variable with a position in the bit string inputs of the parties. For instance if v is the input of a party π then v_j will be the value of the j^{th} variable.

We will use the framework presented in [GKZ10] and that is an extension of the UC-framework [Can05]. Hiding information is expressed through a function suppress that erases any information in the communications between parties that we want to hide from the public view. And Binding properties are expressed in this framework through a predicate validate that validates only specific traces of communication between parties. Note we do not consider any setup assumptions for our work.

4.5.1 Hiding a variable

We first consider the hiding property of the value of a specific variable. The functionality $\mathcal{F}_{hiding(\text{ACTION},j)}$ that hides the j^{th} variable is defined as follows.

Definition 4.3. The functionality $\mathcal{F}_{hiding(\text{ACTION},j)}$ uses two functions: the validate function and the suppress function.

- For the validate function it returns 1 for every message.
- For the suppress function we consider the messages of form $msg = (\text{ACTION}, \mathbf{P}, x)$.
  - For every action of such messages, the suppress function returns x' where

$$x'_i = \begin{cases} x_i & \text{if } i \neq j \\ (-) & \text{if } i = j \end{cases}$$

The general hiding game looks as follows (see figure 10):

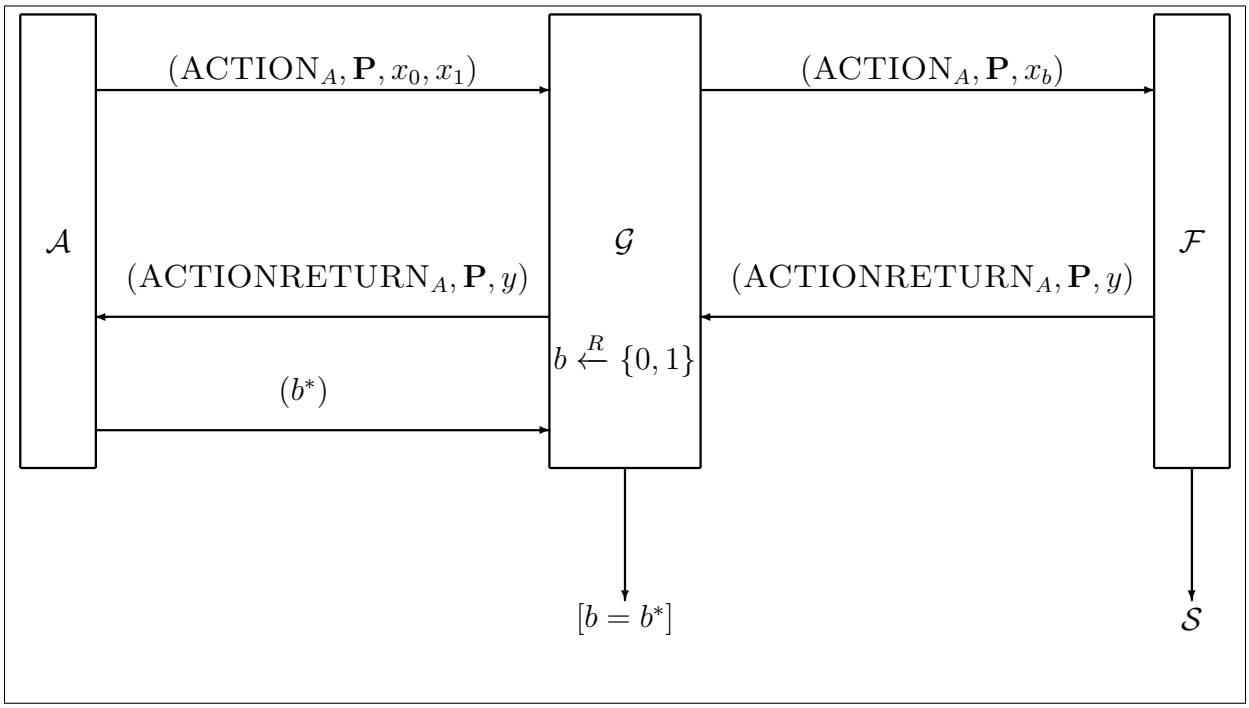

Figure 10: The hiding game

The general hiding game has three parties, the adversary $\mathcal{A}$, the game $\mathcal{G}$ and the functionality $\mathcal{F}$. In the beginning of the game, the adversary $\mathcal{A}$ sends a query ACTION_A and two messages x_0 and x_1 to the game $\mathcal{G}$ along with the list of the parties $\mathbf{P}$ involved in the action. Then $\mathcal{G}$ may send another query ACTION_A to the functionality $\mathcal{F}$ along with a message x_b where bit b is randomly selected from {0,1}. After that $\mathcal{F}$ will perform the required action and give the return value y as the answer to the query back to $\mathcal{G}$. $\mathcal{G}$ will simply forward y to $\mathcal{A}$. After $\mathcal{A}$ gets y, $\mathcal{A}$ will output a bit b^*. The adversary $\mathcal{A}$ wins the game if the probability of its output $b^* = b$ is different from 0.5.

The adversary's goal for the general hiding game is to determine which of the two messages has been used to perform the required action on. In our hiding game, if any information about the message is leaked to the return value of the required action, then the adversary can use that information to determine the message. Intuitively, our description captures the security requirement of the general hiding game and therefore we say it is a good hiding description.

Now we are ready to state our first theorem. It relates the ideal functionality to the general hiding game.

Theorem 4.1. *If $\mathcal{F}$ is syntactically hiding x_j, then $\mathcal{F} \models \mathcal{G}_{hiding}^{(ACTION,j)}$.*

Proof. According to the definition of the functionality $\mathcal{F}_{hiding(\text{ACTION},j)}$ above, the information about the j^{th} variable is replaced with a special character which is not appeared in the alphabet of x. This means the information is erased thus no information about the variable is leaked to the ideal world adversary. Therefore $\mathcal{F}$ hides x_j in a information-theoretic way.

As $\mathcal{F}$ hides x_j in a information-theoretic way and no information about the variable is leaked to the ideal world adversary, therefore no information about x is contained in the return value y of the action ACTION_A and as $\mathcal{G}_{hiding}^{(ACTION,j)}$ forward y to $\mathcal{A}$, $\mathcal{A}$ will therefore get no information about x. Thus the probability of adversary $\mathcal{A}$ winning the game $\mathcal{G}_{hiding}^{(ACTION,j)}$ is exactly 0.5 (it is the probability of the adversary $\mathcal{A}$ randomly guessing a bit from {0,1}). □

4.5.2 Binding a variable

We now consider the binding property of the value of a specific variable. A game binding a variable would allow a party to perform the following actions in the specified order:

- Perform an action ACTION (for instance *commit*) on the value of the variable j at a given message m.
- Perform the return action ACTIONRETURN (for instance *decommit*) in order to reveal the value m'_j and accept whenever $m'_j = m_j$.

We observe that the event where the contents of the j^{th} variable is changed and the message is accepted as valid is the event that breaks the security of the protocol.

Thus we derive the following bad language. The bad language corresponds to the event which the adversary can win a given cryptographic game. In our case, it contains the strings for which the functionality will accept and return 1.

$$B_{binding(x_j)} = \left\{ w \;\middle|\; \begin{array}{rcl} w & = & (\text{ACTION}_A, \mathbf{P}, sid, x) \\ & & (\text{ACTIONRETURN}_A, \mathbf{P}, sid, y) \\ & & (\text{ACTION}_B, \mathbf{P}, sid, x', y') \\ & & (\text{ACTIONRETURN}_B, \mathbf{P}, sid, \phi) \\ \text{such that} & & \\ (x_j \neq x'_j) \wedge (y_j = y'_j) \wedge (\phi = 1) & & \end{array} \right\}$$

The general binding game looks as follows (see figure 11):

Similarly, the binding game has three parties, the adversary $\mathcal{A}$, the game $\mathcal{G}$ and the functionality $\mathcal{F}$. In the beginning of the game, the adversary $\mathcal{A}$ sends a query ACTION_A

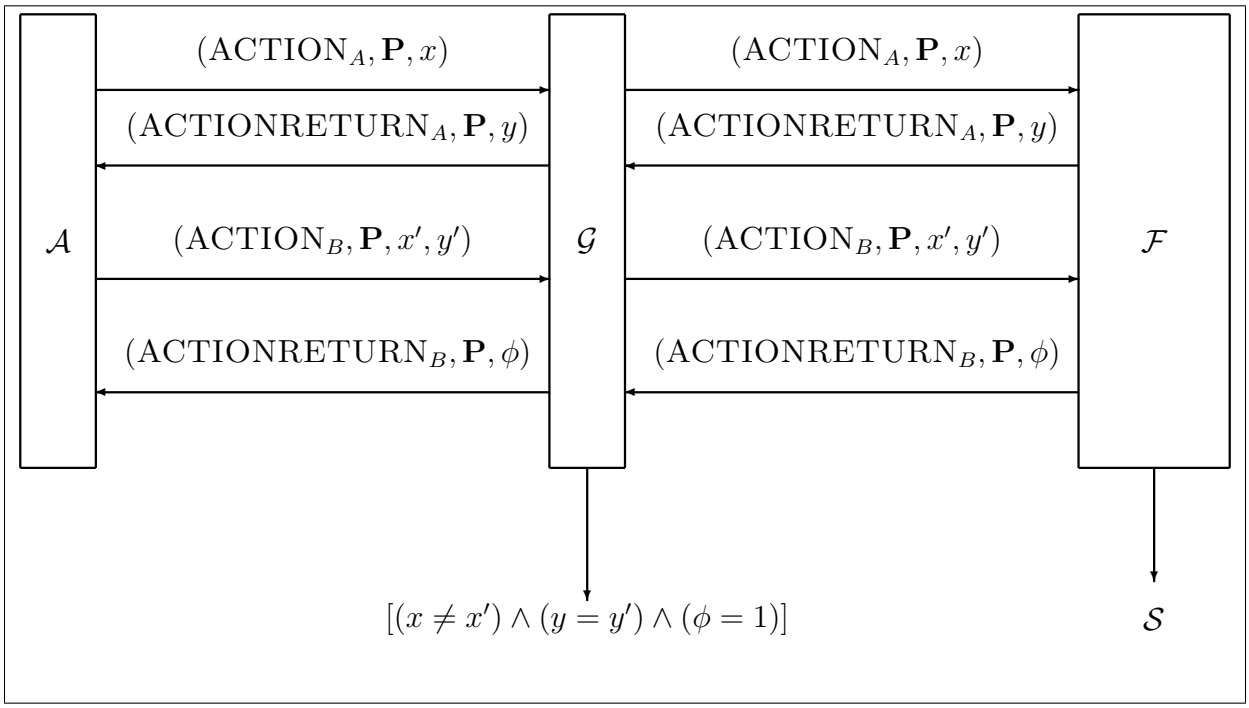

Figure 11: The binding game

and one message x to the game $\mathcal{G}$ along with the list of the parties $\mathbf{P}$ involved in the action. Then $\mathcal{G}$ will simply forward this query to the functionality $\mathcal{F}$. $\mathcal{F}$ will perform such an action ACTION_A and give the return value y of ACTION_A back to $\mathcal{G}$. Then $\mathcal{G}$ will forward y to $\mathcal{A}$. After $\mathcal{A}$ gets y, it will send another query ACTION_B and another message x' along with the corresponding return value y' to $\mathcal{G}$. Again, $\mathcal{G}$ may also forward this query to $\mathcal{F}$. Then $\mathcal{F}$ will perform the action ACTION_B and check the validity of the return value y'. If y' is valid, $\mathcal{F}$ will output 1, otherwise $\mathcal{F}$ will output 0. This result is stored in a variable ϕ and $\mathcal{F}$ may send ϕ back to $\mathcal{G}$. $\mathcal{G}$ simply forward ϕ to $\mathcal{A}$. The adversary $\mathcal{A}$ wins the game if the following conditions holds:$[(x \neq x') \wedge (y = y') \wedge (\phi = 1)]$. In other words, $\mathcal{A}$ produces two valid messages with the same return value.

The adversary's goal for the general binding game is to produce two different messages with the same return value for the required action. In our binding game, if the events:

- $(\text{ACTION}_A, \mathbf{P}, x)$
- $(\text{ACTIONRETURN}_A, \mathbf{P}, y)$
- $(\text{ACTION}_B, \mathbf{P}, x, y)$
- $(\text{ACTIONRETURN}_B, \mathbf{P}, \phi)$

happen in a sequence, then it corresponds to the situation when the adversary can produce such two messages. Intuitively, our description captures the security requirement of the general binding game and therefore we say it is a good binding description.

In order to define the canonical functionality which can capture the general binding game $\mathcal{G}_{Binding}^{(ACTION,j)}$, we need to show that the bad language $B_{binding(x_j)}$ described previously is polynomial-time decidable. Therefore, we can define our canonical functionality $\mathcal{F}_{Binding(\text{ACTION},j)}$ which captures $\mathcal{G}_{Binding}^{(ACTION,j)}$ by requiring the $validate()$ function $validate(w) = 0$ if and only if $w \in B_{binding(x_j)}$ where the decider D for $B_{binding(x_j)}$ is given. In other words, the decider D can be simulated by using $validate()$, the functionality halts when the decider D accepts.

More precisely, we have the following definition:

Definition 4.4. Consider the whole events $[A_1, A_2, A_3, \ldots, A_N]$ during the execution of the game $\mathcal{G}_{Binding}^{(ACTION,j)}$. $\nexists\, i, j, k, l$ such that

- $A_i = (\text{ACTION}_A, \mathbf{P}, sid, x)$
- $A_j = (\text{ACTIONRETURN}_A, \mathbf{P}, sid, y)$
- $A_k = (\text{ACTION}_B, \mathbf{P}, sid, x', y')$
- $A_l = (\text{ACTIONRETURN}_B, \mathbf{P}, sid, \phi)$

  and the functionality accepts these events in order $\forall i < j < k < l$.

Now we are ready to state our second theorem. It relates the ideal functionality to the general binding game.

Theorem 4.2. *If $\mathcal{F}$ is syntactically binding x_j, then $\mathcal{F} \models \mathcal{G}_{binding}^{(ACTION,j)}$.*

Proof. According to the definition above, the functionality $\mathcal{F}$ will not accept the four events in order. Those four events are corresponding to the events in the bad language $B_{binding(x_j)}$. Therefore the functionality $\mathcal{F}$ will not accept any string which is in the bad language $B_{binding(x_j)}$. Thus, $\mathcal{F}$ binds x_j in a information-theoretic way.

As the functionality $\mathcal{F}$ will not accept anything in $B_{binding(x_j)}$, the adversary $\mathcal{A}$ can not produce two messages such that $[(x \neq x') \wedge (y = y') \wedge (\phi = 1)]$. In other words, the adversary $\mathcal{A}$ can not produce two valid messages with the same return value. Thus, the probability of adversary $\mathcal{A}$ winning the game $\mathcal{G}_{binding}^{(ACTION,j)}$ is exactly 0.

□

5 The impossibility

Recently, there has been a lot of work on the negative results about realizing ideal functionalities. We call these results the impossibility result. As we described previously, the ideal functionalities of the commitment scheme are used as a common example. One basic example is that Canetti and Fishlin shows that a particular commitment ideal functionality ($\mathcal{F}_{COM}$) is not realizable [CF01]. Based on that, Datta et al [BDD+06] proved no commitment ideal functionality can be realizable. There are also other variants of the impossibility result, for instance Canetti shows a set of ideal functionalities for ideal coin tossing and zero knowledge are is not realizable [Can05]. And Canetti et al [CHK+05] also shows a particular functionality for password-based key exchange protocol is not realizable.

Our work is based on the previous result [CF01, BDD+06]. It is a more general impossibility result. We are going to introduce some previous work first and then we may introduce our result. Note we do not consider any setup assumptions (for instance using a trusted third party) for the rest of the presentation.

5.1 The impossibility of UC commitment in the plain model

In recent work [CF01], it shows that there exists no Universally Composable commitment protocol in the plain model (with no access to some ideal functionality). The following definitions are presented in [CF01].

Definition 5.1. Protocol π between n parties $P_1, \ldots, P_n$ is a bilateral protocol if there are only two parties interacting with each other and the rest parties stay idle.

Definition 5.2. A bilateral commitment protocol π is a terminating protocol if the commitment of the honest sender P_i can be accepted by the honest receiver P_j and P_j will output $(receipt, sid, P_i, P_j)$ with non-negligible probability. Moreover, if P_j receives a valid decommitment for a given message m and sid from P_i, it will output $(open, sid, P_i, P_j, m)$ with non-negligible probability.

The following theorem is presented in [CF01].

Theorem 5.1. *The bilateral, terminating protocol π which can securely realize functionality $\mathcal{F}_{COM}$ does not exist in the plain model. This holds even if the ideal-model adversary $\mathcal{S}$ is allowed to depend on the environment machine $\mathcal{Z}$.*

We omit the proof here.

5.2 The impossibility of the bit-commitment

Datta et al [BDD+06] shows that there is no realizable ideal functionalities for bit-commitment. This is a generalized version of the result we described above. The following theorem is presented in [BDD+06]

Theorem 5.2. *If $\mathcal{F}$ is an ideal functionality for bilateral bit commitment, then there exits no terminating real protocol $\mathcal{P}$ which securely realizes $\mathcal{F}$.*

Another more general result is presented in the same paper [BDD+06], it says given a functionality $\mathcal{G}$ and a protocol $\mathcal{P}$ which uses $\mathcal{G}$ to construct a perfectly hiding and perfectly binding bit-commitment protocol, then $\mathcal{G}$ is not realizable.

Definition 5.3. Protocol $\mathcal{P}$ is a $\mathcal{G}$-hybrid protocol for a primitive if it is an implementation of the primitive's interface without using any trusted third parties where $\mathcal{G}$ is a given functionality. The exception is that $\mathcal{P}$ can make calls to the functionality $\mathcal{G}$'s interface using the trusted third party.

Formally we have the following theorem presented in [BDD+06]:

Theorem 5.3. *If $\mathcal{G}$ is a functionality and $\mathcal{P}$ is a terminating $\mathcal{G}$-hybrid protocol for bit-commitment. The protocol can provide perfectly hiding and perfectly binding with a very high probability. Then $\mathcal{G}$ is not realizable.*

It is not difficult to find that the ideal functionality $\mathcal{F}$ for bit-commitment consists of the protocol $\mathcal{P}$ and the functionality $\mathcal{G}$. Therefore, if $\mathcal{G}$ is realizable, $\mathcal{F}$ is also realizable. Thus, all the primitives which can be used to construct a bit-commitment protocol are not expected to be realizable as functionalities [BDD+06]. This is quite useful in the study of the concepts like symmetric encryption [BP04] and group signatures [BMW03].

5.3 The impossibility of Universally Composable password-based key exchange

Another impossibility result is stated in [CHK+05]. Generally, the protocols for password-based key exchange can enable two parties to generate a high quality secret when the adversary can take the complete control of their communication network [KV10, CHK+05]. In their work [CHK+05], they define an ideal functionality $\mathcal{F}_{pwKE}$ for the password-based key exchange protocol as follows (see figure 12):

The ideal functionality $\mathcal{F}_{pwKE}$

The functionality $\mathcal{F}_{pwKE}$ is parameterized by using a security parameter k. The interaction between $\mathcal{F}_{pwKE}$ and the adversary $\mathcal{S}$ consists of the following three queries:

1. **Upon receiving a query $(newsession, sid, P_i, P_j, pw, role)$ from party P_i:**

   - Send $(newsession, sid, P_i, P_j, role)$ to adversary $\mathcal{S}$:
     - If it is the first *newsession* query, then record (P_i, P_j, pw) and also label the record $fresh$.
     - If it is the second *newsession* query and also there is a previous record (P_i, P_j, pw'), then record (P_i, P_j, pw) and also label the record $fresh$.

2. **Upon receiving a query $(testpassword, sid, P_i, pw')$ from the adversary $\mathcal{S}$:**

   - If there is a previous record (P_i, P_j, pw) and the record is $fresh$ then:
     - if $pw = pw'$ then label the record *compromised* and output 1 to adversary $\mathcal{S}$.
     - if $pw \neq pw'$ then label the record *interrupted* and output 0 to adversary $\mathcal{S}$.

3. **Upon receiving a query $(newkey, sid, P_i, sk)$ from the adversary $\mathcal{S}$ and $|sk|= k$:**

   - If there is a previous record (P_i, P_j, pw) and it is the first *newkey* query for P_i then:
     - If the record is *compromised* or any of P_i and P_j is corrupted, then output (sid, sk) to P_i.
     - If the record is $fresh$ and also there is a previous record (P_j, P_i, pw') with $pw = pw'$ and the secret key sk' sent to P_j and (P_j, P_i, pw) was $fresh$, then output (sid, sk') to P_i.
     - Pick a new random secret key sk' of the length k and output (sid, sk') to P_i.
   - Label the record *completed*.

Figure 12: The ideal functionality $\mathcal{F}_{pwKE}$ for password-based key exchange protocol

In the work [CHK+05], it has been proved that the above ideal functionality for the password-based key exchange protocol is not realizable. The following theorem is presented in [CHK+05]:

Theorem 5.4. *There exists no non-trivial protocol π which securely realizes the $\mathcal{F}_{pwKE}$ in the plain model.*

The idea behind their proof is proof by contradiction: they assume there is an environment machine $\mathcal{Z}$ which can run the code of the honest parties internally. According to the UC-security framework, for anything that the simulator $\mathcal{S}$ can do for the environment machine $\mathcal{Z}$, there exists another environment machine $\mathcal{Z}'$ which can do the same thing to the honest party in the real protocol. Particularly, the simulator $\mathcal{S}$ must set the session key of the environment machine $\mathcal{Z}$ to be the same as the session key of the ideal functionality in order to have a success simulation. In this case, the another environment machine $\mathcal{Z}'$ can also do the same thing which contradicts the security requirement for the protocol.

5.4 Our impossibility work

Now we are ready to state our main theorem. Before that, we may introduce two useful lemmas presented in [BDD+06]. We will give the proof for the first lemma as it is not done in the original work.

Lemma 5.5. *There exits no terminating real protocol $\mathcal{P}$ which provides both perfectly binding and perfectly hiding and is correct with a high probability.*

Proof. Assume protocol $\mathcal{P}$ is a two-party terminating real protocol and it provides perfectly hiding. For any given message x we represent an unbounded adversary $\mathcal{A}$'s view as $View(\mathcal{A}(x, \mathcal{R}))$ where $\mathcal{R}$ is some randomness $\mathcal{A}$ uses.

Therefore for any two messages x_1 and x_2 of the same length l, we should have $Prob[View(\mathcal{A}(x_1, \mathcal{R})) = v_i] = Prob[View(\mathcal{A}(x_2, \mathcal{R})) = v_i]$ where v_i is a particular view for the unbounded adversary $\mathcal{A}$. In other words, given a view v_i there are two different messages corresponding to it.

Now consider the binding game. The unbounded adversary $\mathcal{A}$ first fixs the randomness $\mathcal{R}$ and then chooses a view v_j. According to the above arguments, $\mathcal{A}$ can find two different messages since it has an infinite computing power. Both the two messages are legal and therefore the adversary $\mathcal{A}$ wins the binding game. Thus, it is not possible to achieve perfectly binding.

□

Lemma 5.6. *If $\mathcal{P}$ is a terminating real protocol which securely realizes $\mathcal{F}$, then $\mathcal{P}$ is correct with a high probability.*

The proof of this lemma is presented in [BDD+06] and we omit it here.

Now we can state our main theorem. It says that given a functionality which satisfies both general hiding game and general binding game, this functionality is not realizable. Recall that we do not consider any setup assumptions here.

Theorem 5.7. *If $\mathcal{F} \models \mathcal{G}_{hiding}^{(ACTION,j)}$ and $\mathcal{F} \models \mathcal{G}_{binding}^{(ACTION,j)}$, then $\mathcal{F}$ is not realizable.*

Our proof is based on [BDD+06]. The main idea is proof by contradiction. It is also the same idea as many other impossibility proofs: given a real protocol $\mathcal{P}$ which securely realizes the ideal functionality $\mathcal{F}$, we can construct another real protocol $\mathcal{Q}$ which is also a correct implementation. On the other hand, principally, all the calls to $\mathcal{Q}$ can be handled by using the copies of $\mathcal{P}$. Because $\mathcal{Q}$ uses the ideal functionality $\mathcal{F}$ internally to perform the requred actions and $\mathcal{F}$ satisfies both the general hiding game and the gereral binding game, as a result, $\mathcal{Q}$ can provide both perfectly hiding and perfectly binding at the same time and it contradicts the first lemma (lemma 5.5).

Proof. Assume protocol $\mathcal{P}$ is a terminating real protocol which securely realizes the ideal functionality $\mathcal{F}$. Therefore there exists a simulator $\mathcal{S}$ which can replace all the calls to $\mathcal{P}$ with the calls to $\mathcal{S}$ associated with $\mathcal{F}$ for any configuration making use of $\mathcal{P}$. The real configuration should be indistinguishable from the ideal configuration.

Therefore we consider the real configuration as follows: the environment ε_1 is the honest responder. At the beginning, it chooses a message x and then sends a query (for instance ACTION_A) along with x to the initiator. Then the initiator may use a copy of the implementation $\mathcal{P}_I$ in order to perform the required action. The adversary just simply forwards the messages to ε_1 in order to corrupt the responder. Then ε_1 may use a copy of the real implementation $\mathcal{P}_R$ in order to play its role as the responder. At the end of the action, ε_1 may instruct the initiator to perform the return action (for instance ACTIONRETURN_A). Then ε_1 will check if the return value of the action is valid.

To make it more clearly, we can have the following figure for the real configuration and the ideal configuration (see figure 13).

The left hand side is the real configuration and the right hand side is the ideal configuration. Consider the ideal configuration first, the initiator uses the ideal functionality $\mathcal{F}$ while the environment ε_1 uses the real implementation $\mathcal{P}_R$. There exists a simulator $\mathcal{S}$ since it can translate the messages of the ideal functionalities in a way that the real

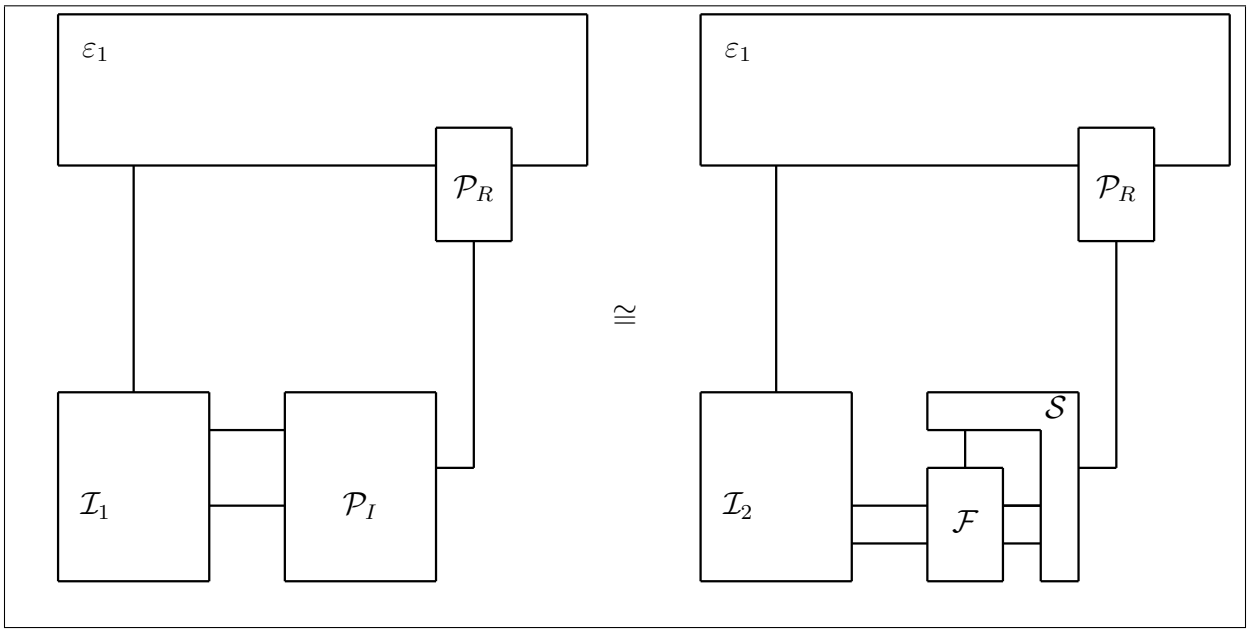

Figure 13: The first step of the configuration [BDD+06]

implementation $\mathcal{P}_R$ can understand. $\mathcal{S}$ sit between the ideal functionality $\mathcal{F}$ and the real implementation $\mathcal{P}_R$ and it is connected to the ideal functionality $\mathcal{F}$ using two interfaces of $\mathcal{F}$. As we assumed before, protocol $\mathcal{P}$ securely realizes the ideal functionality $\mathcal{F}$, therefore the real configuration should be indistinguishable from the ideal configuration. Since adversary does nothing but only forwarding the messages, according to the second lemma above (lemma 5.6), the real configuration is a correct implementation with a high probability. We denote the conjunction of the ideal functionality $\mathcal{F}$ and the simulator $\mathcal{S}$ by a system $\mathcal{Q}$ and $\mathcal{Q}$ is a correct implementation.

Now we can consider another real configuration as shown in the following figure (see figure 14).

This time the responder is honest but the initiator is corrupted. The initiator now acts as a forwarder. After the initiator is corrupted, the adversary does not do anything. This time the environment ε_2 plays as the initiator directly and it chooses a message x at the beginning. ε_2 will use the system $\mathcal{Q}$ instead of using the protocol $\mathcal{P}$ in order to run as the initiator. Then ε_2 may send a query to the functionality along with the message x to the functionality in order to perform some action (for instance ACTION_A). x can be translated into some other form which is suitable for the copy of the real implementation $\mathcal{P}_R$ by the simulator $\mathcal{S}$. After the action is performed, the responder should send a receipt to ε_2 before giving the return value (for instance ACTIONRETURN_A) back. Then it

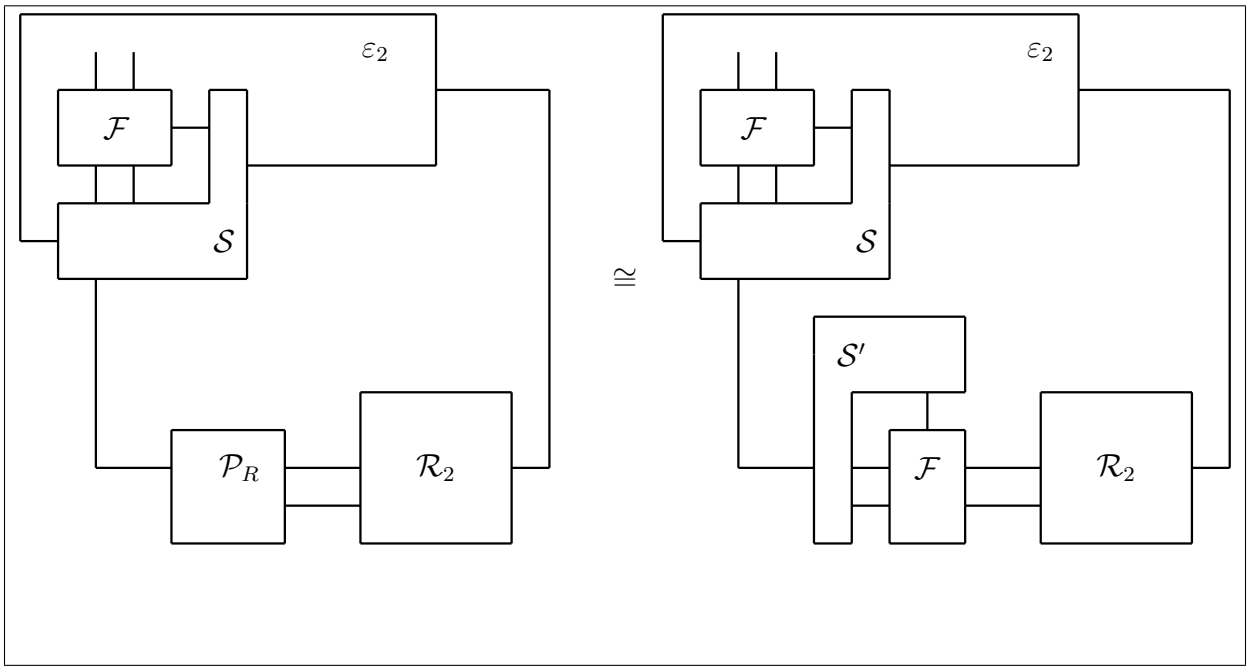

Figure 14: The second step of the configuration [BDD+06]

should send the message it believe the initiator has performed the action on that to ε_2. ε_2 will check if the message is the same as the original one.

For the second part of the configuration, a simulator $\mathcal{S}'$ should sit between the system $\mathcal{Q}$ and the ideal functionality $\mathcal{F}$. The system $\mathcal{Q}$ can be considered as the initiator running the protocol $\mathcal{P}$ and according to the second lemma (lemma 5.6), we can know that this real configuration is also a correct implementation with a high probability. We denote the conjunction of the ideal functionality $\mathcal{F}$ and the simulator $\mathcal{S}'$ by the system $\mathcal{Q}'$. Therefore $\mathcal{Q}'$ must play as the role of the responder running the protocol $\mathcal{P}$ correctly. In other words, $\mathcal{Q}'$ is also a correct implementation.

In the ideal configuration, all the messages go through the two simulators $\mathcal{S}$ and $\mathcal{S}'$. As a result, we have an implementation of a real protocol. System $\mathcal{Q}$ plays the role of the initiator and system $\mathcal{Q}'$ plays the role of the responder. From the above arguments we can see that $\mathcal{Q}|\mathcal{Q}'$ is a correct implementation. Also, because of the way of using the ideal functionality $\mathcal{F}$, the implementation $\mathcal{Q}|\mathcal{Q}'$ can provide both perfectly binding and perfectly hiding at the same time with a high probability. Clearly it contradicts our first lemma (lemma 5.5).

□

As we described previously, our result is more general than the result presented in

[CF01] and [BDD+06]. First, Canetti and Fischlin [CF01] focus on a specific ideal functionality for the commitment games, they show that specific ideal functionality is not realizable. Datta [BDD+06] obtain a more general result since they do not focus on the specific ideal functionality but to look at all the ideal functionalities that specific for the particular games (the commitment games), they show that all the ideal functionalities for the commitment games are not realizable. Our result is more general because we do not focus on the specific games but to look at the class of the games. The general hiding game and the general binding game we consider do not stop the adversary from interacting with the other parts of the functionality. We show that all the ideal functionalities for those two classes of games are not realizable.

6 Conclusion

We first review a new framework for the specification of the cryptographic task: the Universal Composition-security framework and the cryptographic games. Then we give an example of a common cryptographic game: the commitment game. Based on that, we review the relationship between the game conditions and the ideal functionalities. We give a precise definition about the game satisfaction and the descriptions of the general hiding game and the general binding game. Based on that, we prove two theorems. The first theorem says if the functionality is hiding the variable, then this functionality satisfies the general hiding game while the second theorem says if the functionality is binding the variable, then this functionality satisfies the general binding game. Then we review some recent impossibility work and based on that, we proved a more general result: the functionality for the general hiding and general binding game is not realizable.

For this work, we do not consider the setup assumptions. The setup assumptions can be the trusted third party, the random oracles, the common reference strings and so on. Our theorem holds without using thee setup assumptions. However, as stated in some work like [CF01], some ideal functionalities of the bit-commitment can be realized by using a common reference string model. We hope we can develop our work by using some setup assumptions in the future.

There are also some other modifications on providing some other forms of the composability. One possible direction is to have a more relax requirement of the ideal functionalities. As presented in [Mau02], we can replace information-theoretic equivalence with the indistinguishability of random systems. Another possible direction is a modification of the original Universal Composition-security framework like [PS04, PS05]. Also as stated in [BDD+06], it would be quite useful to develop some methods in order to prove some conditional forms of composability.

References

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: security proofs and improvements. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 259–274, Berlin, Heidelberg, 2000. Springer-Verlag.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37:156–189, October 1988.

[BDD+06] Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the impossibility of realizable ideal functionality. In *In theory of cryptography, 3rd theory of cryptography conference, tcc 2006, volume 3876 of lecture notes in computer science*, pages 360–379. Springer, 2006.

[Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122, 1991. 10.1007/BF00196771.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation (extended abstract). In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 629–638, New York, NY, USA, 1996. ACM.

[BH03] Michael Backes and Dennis Hofheinz. How to break and repair a universally composable signature functionality. In *In ISC 2004*, pages 61–72. Springer, 2003.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions, 2003.

[BP04] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Proceedings of the 17th IEEE workshop on Computer Security Foundations*, pages 204–, Washington, DC, USA, 2004. IEEE Computer Society.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *TCC'04*, pages 336–354, 2004.

[BR04] M. Bellare and P. Rogaway. The game-playing technique, 2004.

[Can98] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13:2000, 1998.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA, 2001. IEEE Computer Society.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings of the 17th IEEE workshop on Computer Security Foundations*, pages 219–, Washington, DC, USA, 2004. IEEE Computer Society.

[Can05] Ran Canetti. Universally Composable Security: A new paradigm for cryptographic protocols. Technical report, December 2005.

[Can06] Ran Canetti. Security and composition of cryptographic protocols: a tutorial (part i). *SIGACT News*, 37:67–92, September 2006.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 19–40, London, UK, 2001. Springer-Verlag.

[CHH+07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded cca2-secure encryption. In *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, ASIACRYPT'07, pages 502–518, Berlin, Heidelberg, 2007. Springer-Verlag.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Phil MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 557–557. Springer Berlin / Heidelberg, 2005. 10.1007/11426639˙24.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptol.*, 19:135–167, April 2006.

[Dam90] Ivan Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 17–27, London, UK, UK, 1990. Springer-Verlag.

[Dam99] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*, pages 63–86, London, UK, 1999. Springer-Verlag.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 542–552, New York, NY, USA, 1991. ACM.

[DKMR05] Anupam Datta, Ralf Ksters, John Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. In Joe Kilian, editor, *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 476–494. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-30576-7˙26.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 74–92, London, UK, 2000. Springer-Verlag.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 416–426, New York, NY, USA, 1990. ACM.

[GKZ10] Juan A. Garay, Aggelos Kiayias, and Hong-Sheng Zhou. A framework for the sound specification of cryptographic tasks. In *Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium*, CSF '10, pages 277–289, Washington, DC, USA, 2010. IEEE Computer Society.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, pages 77–93, London, UK, UK, 1991. Springer-Verlag.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38:690–728, July 1991.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21:666–677, August 1978.

[Kus06] Ralf Kusters. Simulation-based security with inexhaustible interactive turing machines. In *Proceedings of the 19th IEEE workshop on Computer Security Foundations*, pages 309–320, Washington, DC, USA, 2006. IEEE Computer Society.

[KV10] Jonathan Katz and Vinod Vaikuntanathan. One-round password-based authenticated key exchange, 2010.

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM conference on Computer and communications security*, CCS '98, pages 112–121, New York, NY, USA, 1998. ACM.

[Mau02] Ueli M. Maurer. Indistinguishability of random systems. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, pages 110–132, London, UK, UK, 2002. Springer-Verlag.

[MRST01] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols. In *Electronic Notes in Theoretical Computer Science*, 2001.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness (extended abstract). In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 128–136, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC '04, pages 242–251, New York, NY, USA, 2004. ACM.

[PS05] Manoj Prabhakaran and Amit Sahai. Relaxing environmental security: Monitored functionalities and client-server computation. In *TCC'05*, pages 104–127, 2005.

[PSW00] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems, 2000.

[PW94] Birgit Pfitzmann and Michael Waidner. A general framework for formal notions of "secure" systems. In *SYSTEM, HILDESHEIMER INFORMATIK-BERICHTE 11/94, UNIVERSITAT*, 1994.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 184–, Washington, DC, USA, 2001. IEEE Computer Society.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. http://eprint.iacr.org/.