Sound and Fine-grain Specification of Cryptographic Tasks

Juan A. Garay [*] Aggelos Kiayias [†] Hong-Sheng Zhou[†]

March 24, 2008

Abstract

The Universal Composability (UC) framework, introduced by Canetti, allows for the design of cryptographic protocols satisfying strong security properties, such as non-malleability and preservation of security under (concurrent) composition. In the UC framework (as in several other frameworks), the security of a protocol carrying out a given task is formulated via the "trusted-party paradigm," where the protocol execution is compared with an ideal process where the outputs are computed by a trusted party that sees all the inputs. A protocol is said to securely carry out a given task if running the protocol with a realistic adversary amounts to "emulating" the ideal process with the appropriate trusted party. In the UC framework the program run by the trusted party is called an *ideal functionality*.

However, while this simulation-based security formulation provides strong security guarantees, its usefulness is contingent on the properties and correct specification of the realized ideal functionality, which, as demonstrated in recent years by the coexistence of complex, multiple functionalities for the same task as well as by their "unstable" nature, does not seem to be an easy task. On the other hand, the more traditional, *gamed-based* definitions of cryptographic tasks, although providing a less satisfying level of security (stand-alone executions, or executions in very controlled settings), have been successful in terms of formalizing as well as capturing the underlying task's natural properties.

In this paper we address this gap in the security modeling of cryptographic properties, and introduce a general methodology for translating game-based definitions of properties of cryptographic tasks to syntactically concise ideal functionality programs. Moreover, taking advantage of a suitable algebraic structure of the space of our ideal functionality programs, we are able to "accumulate" ideal functionalities based on many different game-based security notions. In this way, we can obtain a well-defined mapping of all the game-based security properties of a cryptographic task to its corresponding UC counterpart. In addition, the methodology allows us to "debug" existing ideal functionalities, establish relations between them, and make some critical observations about the modeling of the ideal process in the UC framework. We demonstrate the power of our approach by applying our methodology to a variety of basic cryptographic tasks, including commitments, digital signatures, public-key encryption, zero-knowledge proofs, and oblivious transfer.

Instrumental in our translation methodology is the new notion of a *canonical functionality class* for a cryptographic task which is endowed with a bounded semilattice structure. This structure allows the grading of ideal functionalities according to the level of security they offer as well as their natural joining, enabling the modular combination of security properties.

Key words: Cryptographic protocols, universal composability, security definitions, lattices and partial orders.

[*]Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974, USA. E-mail: `garay@research.bell-labs.com`.

[†]University of Connecticut, Computer Science and Engineering, Storrs, CT 06269, USA. E-mail: `{aggelos,hszhou}@cse.uconn.edu`.

Contents

1 Introduction

The Universal Composability (UC) framework proposed by Canetti [Can05], culminating a long sequence of simulation-based security definitions (cf. [GMW87, GL90, MR91, Bea91, Can00]; see also [PW01] for an alternative framework), allows for arguing the security of cryptographic protocols in arbitrary settings where executions can be concurrent and adversarially interleaved. The framework is particularly attractive for the design of secure systems as it supports modular design, provides non-malleability across sessions [DDN00], and preserves security under composition.

In the UC framework, security is argued by providing a proof that a protocol realizes an *ideal functionality* $\mathcal{F}$ for the same cryptographic task. While this simulation-based formulation provides satisfying security guarantees, its usefulness is contingent on the properties of the realized ideal functionality. In particular, any ideal functionality is required to interact with an ideal-world adversary to whom it reveals aspects of its internal state. Thus, such a program can be quite far from an idealization of a given cryptographic task. To make things worse, the application of the framework to the analysis of many cryptographic schemes has shown that relatively complex ideal functionality programs are the norm. This has frequently led to successive revisions of ideal functionality programs, the simultaneous coexistence of multiple different ideal functionalities for the same task, and the discovery of errors in the ideal functionality programs, which in turn would lead to flawed security guarantees for the protocols realizing them. (A quick inspection of recent papers providing UC formulations of cryptographic tasks should suffice to support the complexity claim; see the treatment of digital signatures [Can01, BH04, Can04, Can05] for an example of need-to-revise and error-prone formulations of ideal functionalities.)

On the other hand, basic cryptographic schemes have enjoyed a rigorous security formalization through *gamed-based definitions.* While such definitions provide a less satisfying level of security guarantees (as they may exclude composition, adaptive corruptions, non-malleability and other properties offered by the UC framework), they are typically much easier to specify and understand as the appropriate formulations of the natural properties of the underlying cryptographic task. Examples include the existential unforgeability notion for digital signatures [GMR88], IND-CPA security for public-key encryption, the binding property of commitment schemes, the soundness property of zero-knowledge, and others. Thus, there exists a clear gap in the security modeling of cryptographic properties: strong simulation-based definitions as provided by the UC framework can provide adequate and much-needed security guarantees for protocol executions and interactions in today's open networks, but, as argued above, tend to be complex and error-prone, which makes the task of assessing the usefulness of a particular ideal functionality specification as a security definition particularly challenging.

In this paper we address this problem by introducing a general methodology for translating game-based definitions of properties of cryptographic tasks to syntactically concise ideal functionality programs. Moreover, taking advantage of a suitable algebraic structure of the space of our ideal functionality programs we are able to "accumulate" ideal functionalities based on different game-based security notions. In this way, we obtain a well-defined mapping of all the game-based security properties of a cryptographic task to their corresponding UC counterpart. Furthermore, the systematic approach we present enables the debugging and fine tuning of ideal functionalities to the intended underlying properties of cryptographic tasks.

Our results. First, we introduce the notion of the class of *canonical functionalities* for a cryptographic task T. Each member of this class has a simple, concise syntax built around two pass-through communication flows: one from the environment to the ideal-world adversary and another in the opposite direction. Every cryptographic task is associated to its corresponding canonical functionality class. We next define an operation over this class and show that the class has the algebraic structure of a semilattice which enables the joining of canonical functionalities. This algebraic structure is a unique feature that we take advantage of in our translation methodology. In fact, it imposes a natural preordering of canonical functionalities which enables the grading of canonical functionalities according to the level of security they offer. Finally, the syntactic conciseness of our canonical functionalities gives rise to a well-defined communication (formal) language between the functionality and the other entities in an ideal world simulation which is instrumental in our methodology.

These results are presented in Section 2.

Second, we present a general formulation of game-based definitions for properties of cryptographic tasks. We divide games into two general types: *consistency games* and *hiding games*. The former capture properties such as correctness, unforgeability and binding, while the latter capture properties such as IND-CPA security, commitment hiding and zero-knowledge. Depending on the type of game we present a translation methodology for obtaining its corresponding canonical functionality. Then, given a set of canonical functionalities that capture individual game-based properties of a cryptographic task, we provide the ideal functionality of the task by making use of the algebraic structure of our canonical functionality class. We demonstrate the soundness of our approach by showing that any scheme that realizes such ideal functionality would also possess the properties of the game-based definitions. These results are presented in Section 3. In some cases, our transformation is "tight," as we are also able to show the opposite direction while in others the resulting canonical functionality corresponds to stricter security notions due to the strength of the UC framework.

Third, we apply our methodology to a variety of basic cryptographic tasks, obtaining ideal functionalitites for commitments, digital signatures, public-key encryption, zero-knowledge proofs of knowledge, and oblivious transfer. While in some cases (commitment, zero-knowledge) the obtained functionalitites are essentially equivalent to existing ones in the literature, in others (signature, public-key encryption and oblivious transfer) they differ, allowing us to pinpoint their shortcomings with respect to ours. In fact, this "debugging" goes beyond the specification of particular tasks, as in the case of oblivious transfer we are able to point to a structural inadequacy in the UC notion of "delayed output" in the ideal process.

For conciseness, only the treatment of commitments, whose properties are expressed using both types of games, is presented in the main body of the paper (Section 4), while some of the other tasks — signatures, ZK and OT — appear in the appendix (Section D). (Public-key encryption will be added shortly.) Proofs, as well as a brief overview of the UC framework, also appear in the appendix.

Preliminaries. We first introduce some notation and review notions that will be used throughout the paper. Given a sequence w consisting of elements from an alphabet $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_k\}$, we let w_i denote the i-th element in w. We can obtain a *subsequence* of w, call it w', by erasing some of the elements in w without disturbing the relative positions of the remaining elements. We denote this by $w' \preccurlyeq w$ and we remark that $\epsilon \preccurlyeq w$ for any string w. If $\Sigma' \subseteq \Sigma$, for any string $w \in \Sigma^*$ we denote by $w|_{\Sigma'}$ the largest subsequence of w that belongs to $(\Sigma')^*$. For any $w \in \Sigma^*$ we write $w' \prec w$ when w' is derived from w after substituting at least one symbol of w with the special symbol "$-$". Finally, for a given set of strings S we define $S^{\prec} = \{w' \mid \exists w \in S : w' \prec w\}$.

A *monoid* $(A, +)$ is a semigroup with an identity element. Any monoid possesses a preorder relation denoted by $\lesssim$ such that $a \lesssim b$ iff $\exists c : a + c = b$.

2 Canonical Functionalities

Cryptographic tasks and environment communication. A (non-interactive) *cryptographic task* is a tuple $T = \langle A_1, \ldots, A_k \rangle$. Each A_i is called an *action* of the cryptographic task with $A_i = \langle \textsc{Action}_i, \{D_i^{(\lambda)}\}_{\lambda \in \mathbb{N}}, \{R_i^{(\lambda)}\}_{\lambda \in \mathbb{N}} \rangle$ where $\textsc{Action}_i$ is a label, $\lambda \in \mathbb{N}$ is the security parameter and $D_i^{(\lambda)}$ and $R_i^{(\lambda)}$ are sets of values called the domain and range of the action, respectively. An example of a non-interactive cryptographic task is a digital signature that has three actions: "key-generation," "signature-generation," and "signature-verification." An *interactive* cryptographic task between two players is defined similarly with the difference that any of the actions A_i may have pairs of values as domain and range elements; in such case, A_i is an action involving two parties. For example, a commitment is an interactive cryptographic task with two actions, "commit" and "open" in which two parties are involved. Multi-party cryptographic tasks are defined similarly.

A *cryptographic scheme* $\Sigma = \langle M_1, \ldots, M_k \rangle$ implements a cryptographic task $T = \langle A_1, \ldots, A_k \rangle$ if M_i is an algorithm that for input sizes λ, maps elements of $D_i^{(\lambda)}$ to $R_i^{(\lambda)}$. Note that in the case of an interactive task, M_i would be a protocol between (amongst) the parties that are involved in the action of the (multiparty) cryptographic task.

Recall that in the UC framework, the environment is creating processes which are entities maintaining state across actions. For this reason, we want to associate to any cryptographic task T a stateful abstract entity[1] π_T. In more detail, π_T is a "packaging" of the actions of the task T together with some data fields that are persistent across action invocations. For example, an abstract entity for the commitment task offers two actions, commit and open, and has a persistent data field that is generated by the commit action and used by the corresponding open action (the decommitment information). Similarly, an abstract entity for the digital signature task offers three actions, key-generation, signature-generation, and signature-verification, and has a persistent data field produced by the key-generation action and used by the signature-generation action (this is the signing key). If the cryptographic scheme Σ implements the cryptographic task T, then π_Σ would be a protocol implementation of π_T that is derived by implementing all actions of π_T with the algorithms of Σ. Note that in the UC framework, a π_Σ protocol is intended to approximate an "ideal functionality" $\mathcal{F}_T$ that shares with π_Σ the same I/O specifications (i.e., they possess the same communication interface with the environment).

Below we specify more explicitly the language of the communication between the π_T entities and their environment. For each $\lambda \in \mathbb{N}$, we define the set of symbols of the form $(\textsc{Action}_i, \mathbf{P}, x)$; here $\textsc{Action}_i$ is the label that corresponds to the i-th action. $\mathbf{P}$ is a tuple that designates the identifiers of the entities and their roles in the particular action (in particular which parties provided the input to the action and which parties should receive output). To differentiate multiple copies of the protocol run by the same group of entities, $\mathbf{P}$ may also include a session identifier sid. Notice that the same $\textsc{Action}_i$ may be sent by more than one entities in $\mathbf{P}$. The value $x \in D_i^{(\lambda)}$ is an encoding of the input to the action; note that whenever $x = \epsilon$, we will drop x from the symbol notation for ease of reading.

In response to a symbol $(\textsc{Action}_i, \mathbf{P}, x)$, entity π_T will return a symbol $(\textsc{ActionReturn}_i, \mathbf{P}, y)$ where $y \in R_i^{(\lambda)}$; note that more than one entities may be receiving an $\textsc{ActionReturn}_i$ symbol for a certain action. $(\textsc{ActionReturn}_i, \mathbf{P}, y)$, constitutes the finite I/O alphabet of the π_T entity and is denoted by Σ_T; note that we drop the parameterization by λ for simplicity.

The actions of a cryptographic scheme might make sense only in certain order; for this reason not all strings over Σ_T are valid as action sequences. To formalize this, we associate with π_T a predicate WF_T called the *well-formedness predicate*. For any string $w \in (\Sigma_T)^*$ and symbol $\mathtt{a} \in \Sigma_T$, the well-formedness predicate $\mathsf{WF}_T(w, \mathtt{a})$ decides whether the string $w\mathtt{a}$ is intelligible with respect to the cryptographic task T.

The functional dependency between action inputs and action outputs for the task T is captured by a string mapping $\mathsf{DO}_T : (\Sigma_T)^* \times (\Sigma_T) \to O^{(\lambda)}$ called the *default output*. DO_T is history dependent: given a pair $\langle w, \mathtt{a} \rangle$ that satisfies $\mathsf{WF}_T(w, \mathtt{a})$, it will return a value that is the intended output of the task on action symbol $\mathtt{a}$ given the history w. For example, in the case of a zero-knowledge task $T = \mathtt{ZK}$, upon receiving $(\textsc{Prove}, \langle P, V, sid \rangle, \langle x, m \rangle)$, $\mathsf{DO}_{\mathtt{ZK}}$ will output the pair $\langle x, \phi \rangle$ where $\phi = 1$ if and only if $\langle x, m \rangle$ belongs to the relation that parameterizes the zero-knowledge task. In the case of a commitment task $T = \mathtt{COM}$, assuming that the history contains $(\textsc{Commit}, \langle C, V, sid \rangle, m)$ $(\textsc{CommitReturn}, \langle C, V, sid \rangle)$ on action symbol $(\textsc{Open}, \langle C, V, sid \rangle)$, the mapping $\mathsf{DO}_{\mathtt{COM}}$ will output the value m as this is the intended output of a commitment scheme at this moment.

This completes the description of a cryptographic task and its communication with the environment. Looking ahead, in the UC framework, within a single session in the real world there can be many processes running the implementation π_Σ of a cryptographic task T activated by the environment. These objects and their interactions are idealized by a single entity in the ideal world called the ideal functionality $\mathcal{F}_T$ that corresponds to the cryptographic task T. Recall that $\mathcal{F}_T$ presents to the environment the same I/O interface that a collection of π_Σ processes provides in the real world. It follows that the communication language of $\mathcal{F}_T$ is over the alphabet Σ_T. Following the UC formalization, besides the interaction with the environment, $\mathcal{F}_T$ also communicates with an ideal world entity, called the ideal world adversary $\mathcal{S}$. This interaction defines another communication language that is not bound by the alphabet of the real world. Moreover, $\mathcal{F}_T$ may halt whenever it deems that a certain sequence of symbols is not consistent with some of the required properties of the cryptographic task T. Below

[1]In the UC framework [Can05], this entity would correspond to a "protocol" specification.

we explicitly provide a syntax for ideal functionalities by introducing the concept of a *canonical functionality* for a cryptographic task T.

The canonical functionality of a cryptographic task. In order to obtain an ideal functionality for a cryptographic task T, we also need to specify the communication alphabet with the ideal world adversary as well as the way this communication is defined based on the functionality's state. Moreover, we need to specify cases where the functionality may wish to halt its operation. In this section we provide an explicit syntax for a class of functionalities that idealize the cryptographic task T — this is the *class of canonical functionalities* for the cryptographic task T. In this first formulation of canonical functionalities we focus on a wide class of cryptographic tasks whose action outputs are not required to follow an ideal probability distribution. Such tasks include digital signatures, commitment, public-key encryption, secure message transmission, zero-knowledge proofs, secure deterministic function evaluation, etc.

First we define the communication language between a canonical functionality and the ideal world adversary. For each action symbol $(\textsc{Action}_i, \mathbf{P}, x)$, the canonical functionality will produce a "leaking-action" symbol $(\textsc{LeakAction}_i, \mathbf{P}, x')$ where $x' \prec x$ and send it to the ideal world adversary; this means that the functionality will pass control to the adversary and will provide to the adversary the value x' which is a "suppressed" version of the action input x (note that x' may only reveal the length of x). The default output of the cryptographic task (provided by the DO_T mapping) will also be sent out with the $\textsc{LeakAction}_i$ symbol to the adversary. We distinguish two types of default outputs: *public* and *private*. When the default output is public the adversary is allowed to see the output that is meant to be delivered to a party. On the other hand, if the default output is private, the adversary will be handed a pointer to the value of the default output. The functionality keeps track of the correspondence between pointers and actual values and the adversary can take advantage of such pointers to provide output to parties by dereferencing them. As examples of a public output case, recall the default outputs corresponding to the tasks of zero-knowledge and commitments in the previous section. Private outputs on the other hand are intrinsic to the cryptographic tasks of oblivious transfer and secure function evaluation.

At any point, the ideal world adversary may provide an "influence-action" symbol $(\textsc{InflAction}_i, \mathbf{P}, y')$ to the functionality; this symbol instructs the functionality to produce an $(\textsc{ActionReturn}_i, \mathbf{P}, y)$ symbol, where $y = y'$ if y' is a value (public output) and $y = *(y')$ if y' is a pointer to a value (private output), and to send it to the party P (note that the identity of P is inferred from $\mathbf{P}$). If more than one party is supposed to receive an output then the adversary will generate as many $\textsc{InflAction}$ symbols as necessary[2]. In addition to the above symbols, the communication between the canonical functionality and the adversary includes two other symbols, namely, $\textsc{Corrupt}$ and $\textsc{Patch}$, that enable the adversary to control corrupted parties (see below). The extended communication alphabet is denoted by Σ_T^{ext} and includes all I/O symbols of Σ_T as well as the corresponding $\textsc{InflAction}$, $\textsc{LeakAction}$, $\textsc{Corrupt}$ and $\textsc{Patch}$ symbols.

Whenever the canonical functionality receives an action symbol it recovers the history of previously received symbols for the party providing the input and tests the new symbol with the well-formedness predicate WF_T; it ignores any symbol that fails the well-formedness test WF_T.

Furthermore, the canonical functionality is parameterized by two functions: $\mathsf{suppress}()$ and $\mathsf{validate}()$. Given an action symbol $(\textsc{Action}_i, \mathbf{P}, x)$, the function $\mathsf{suppress}()$ will determine what information about x will the ideal world adversary learn (the output of the function will be passed into the $\textsc{LeakAction}$ symbol together with the default output). Specifically, the $\mathsf{suppress}()$ function is defined over the $(\textsc{Action}_i, \mathbf{P}, x)$ symbols and will output some $x' \prec x$. We require that the $\mathsf{suppress}()$ function will always substitute with "$-$" the same locations of x, independently of x. The $\mathsf{validate}()$ function, on the other hand, tells the functionality when to halt and is used whenever the functionality receiving an $\textsc{InflAction}$ symbol produces an $\textsc{ActionReturn}$ symbol as output. Specifically, the $\mathsf{validate}()$ predicate is defined over strings of Σ^*. We note that $\mathsf{suppress}()$ is history independent while the $\mathsf{validate}()$ predicate is not. The intuition is that the $\mathsf{suppress}()$ function abstracts what the adversary learns about the possibly private inputs of parties (i.e., it captures the hiding aspects of the

[2]In our current formalization we postpone the treatment of fairness.

functionality) whereas the $\mathsf{validate}()$ predicate makes sure that the outputs produced by the functionality are consistent with its history according to the intended consistency properties of the task.

A canonical functionality needs also to maintain state. The state of the functionality, denoted by $\mathsf{history}$, is the sequence of all I/O symbols ordered chronologically as received from and sent to the environment. We use $\mathsf{history}_{P_j}$ to denote all symbols associated to party P_j in history, i.e., the ACTION symbols that were provided by P_j and ACTIONRETURN symbols that were returned to P_j.

Canonical Functionality $\mathcal{F}_T^{\mathsf{suppress},\mathsf{validate}}$

Initially, $\mathsf{history} := \epsilon$ and $\mathsf{binding} := \epsilon$.

- Upon receiving $msg = (\textsc{Action}_i, \mathbf{P}, x)$ from some party P_j, if P_j is corrupted set $x' = x$ else compute $x' \leftarrow \mathsf{suppress}(msg)$, set $msg' \leftarrow (\textsc{LeakAction}_i, \mathbf{P}, x')$ and if $\mathsf{WF}_T(\mathsf{history}_{P_j}, msg) = 1$ send $\langle msg', \mathsf{DO}_T(\mathsf{history}, msg)\rangle$ to the adversary $\mathcal{S}$, record msg in history and in case P_j is uncorrupted, set $\mathsf{binding}[l] = \{P_j\}$ where $l = |\mathsf{history}|$; otherwise ignore msg.
- Upon receiving $msg = (\textsc{InflAction}_i, \mathbf{P}, y')$ from the adversary $\mathcal{S}$, infer P_j from $\mathbf{P}$, set $msg' \leftarrow (\textsc{ActionReturn}_i, \mathbf{P}, y)$, where $y = y'$ if y' is a value, or $y = *(y')$ if y' is a pointer, and if $\mathsf{WF}_T(\mathsf{history}_{P_j}, msg') = 1$ record msg' in history (otherwise ignore msg). If $\mathsf{validate}(\mathsf{history}) = 1$, then send msg' to party P_j; if P_j is uncorrupted, set $\mathsf{binding}[k] \leftarrow \mathsf{binding}[k] \cup \{P_j\}$, $1 \le k \le |\mathsf{history}|$. Otherwise ($\mathsf{validate}(\mathsf{history}) = 0$), if P_j is corrupted remove msg' from history (i.e., ignore msg) else send an error symbol to the party P_j and halt.
- Upon receiving $msg = (\textsc{Corrupt}, P_j)$ from the adversary $\mathcal{S}$, mark P_j as corrupted, return $\mathsf{history}_{P_j}$ to the adversary $\mathcal{S}$, and set $\mathsf{binding}[k] \leftarrow \mathsf{binding}[k] \setminus \{P_j\}$, $1 \le k \le |\mathsf{history}|$.
- Upon receiving $msg = (\textsc{Patch}, \mathsf{history}')$ from the adversary $\mathcal{S}$ where $\mathsf{history}' \in (\Sigma_T)^{|\mathsf{history}|}$ do the following: if $\mathsf{binding}[k] = \emptyset$ set $\mathsf{history}[k] \leftarrow \mathsf{history}'[k]$, $1 \le k \le |\mathsf{history}|$.

Figure 1: Definition of the class of canonical functionalities $\mathscr{F}_T$ for a task T quantifying over all admissible pairs $\mathsf{suppress}(), \mathsf{validate}()$.

The CORRUPT and PATCH symbols are used to handle the behavior of corrupted parties. When a party P_j is corrupted, we allow the adversary to learn $\mathsf{history}_{P_j}$[3]. Moreover, to handle adaptive corruptions, we allow the adversary to rewrite the history of corrupted parties using PATCH symbols in the following manner: a certain symbol that was provided by a corrupted party can be modified provided this symbol has not contributed to the view of any honest party. To facilitate this checking the canonical functionality uses an array called $\mathsf{binding}[\cdot]$ that for each symbol in history it records the set of honest parties whose view could have been affected by that symbol.

The canonical functionality operates with parties whose identities are selected from a namespace, within a session that is recognized by a session identifier which belongs to a session-id space. We define the class of canonical functionalities in Figure 1 and a pictorial representation can be found in Figure 2; each member of the class is specified by a pair of functions $\mathsf{suppress}(), \mathsf{validate}()$ as defined above. For a given cryptographic task T, the *dummy functionality* for T defined as follows:

Definition 2.1 (Dummy Functionality). *We call the canonical functionality $\mathcal{F}_T^{\mathrm{dum}} \in \mathscr{F}_T$* dummy *if (1) for all x and any* ACTION, $\mathsf{suppress}((\textsc{Action}, \mathbf{P}, x)) = x$*, and (2)* $\mathsf{validate}() = 1$ *always.*

It should be noted that the dummy functionality does not capture any of the intended correctness or security properties of the cryptographic task T. This means that any protocol π UC-realizing $\mathcal{F}_T^{\mathrm{dum}}$ will merely syntactically match the purpose of T but will lack any useful property.

[3]We also defer the treatment of forward security for now.

Theorem 2.2. *For any cryptographic task T there is a scheme Σ implementing T such that π_Σ UC-realizes the dummy functionality $\mathcal{F}_T^{\mathrm{dum}}$.*

Algebraic structure of the class of canonical functionalities. We define a conjunction operation denoted by $\wedge$ on the class of canonical functionalities for a task T. Observe that for any two members of the canonical functionality class that are parameterized by the functions $\mathsf{suppress}_1, \mathsf{suppress}_2$ respectively, for any symbol $\mathsf{a} = (\textsc{Action}_i, \mathbf{P}, x)$, it holds that $\mathsf{suppress}_1((\textsc{Action}_i, \mathbf{P}, \mathsf{suppress}_2(\mathsf{a}))) = \mathsf{suppress}_2((\textsc{Action}_i, \mathbf{P}, \mathsf{suppress}_1(\mathsf{a})))$. This fact will be handy in the definition below.

Definition 2.3 (Conjuncting Functionalities)**.** *Given $\mathcal{F}_1 = \mathcal{F}_T^{\mathsf{suppress}_1,\mathsf{validate}_1}, \mathcal{F}_2 = \mathcal{F}_T^{\mathsf{suppress}_2,\mathsf{validate}_2} \in \mathscr{F}_T$ we define the* conjunction *$\mathcal{F}_1 \wedge \mathcal{F}_2$ of the two functionalities as the functionality $\mathcal{F}_T^{\mathsf{suppress},\mathsf{validate}} \in \mathscr{F}_T$, where, (1) for any $\mathsf{a} = (\textsc{Action}, \mathbf{P}, x) \in \Sigma$, $\mathsf{suppress}(\mathsf{a}) = \mathsf{suppress}_1(\textsc{Action}_i, \mathbf{P}, \mathsf{suppress}_2(\mathsf{a}))$, and (2) $\mathsf{validate}() = \mathsf{validate}_1() \wedge \mathsf{validate}_2()$, i.e., the logical conjunction of the two validate predicates of $\mathcal{F}_1, \mathcal{F}_2$.*

Proposition 2.4. *$(\mathscr{F}_T, \wedge)$ is a commutative monoid with the dummy functionality $\mathcal{F}_T^{\mathrm{dum}}$ as the identity element.*

Any commutative monoid has an associated preordering relation denoted by $\lesssim$; in the case of $(\mathscr{F}_T, \wedge)$ we say that $\mathcal{F}_1 \lesssim \mathcal{F}_2$ iff there exists $\mathcal{F}_3$ such that $\mathcal{F}_2 = \mathcal{F}_1 \wedge \mathcal{F}_3$. The intuitive interpretation of $\mathcal{F}_1 \lesssim \mathcal{F}_2$ is that $\mathcal{F}_2$ is at least as strict from a security point of view as $\mathcal{F}_1$.

$\mathscr{F}_T$ together with $\wedge$ forms a bounded (join-)semilattice, i.e., every set of elements in $\mathscr{F}_T$ has a least upper bound. Note that (1) we use $\wedge$ in place of the standard $\vee$ in lattice theory as it is more consistent as an operator in our setting where lattice elements would capture security properties (and going higher in the lattice means that security increases), and (2) given that $\mathscr{F}_T$ as a commutative monoid lacks the antisymmetric property, the semilattice would be in fact over the quotient $\mathscr{F}_T/\asymp$ where $\asymp$ is the equivalence relation defined as $\mathcal{F}_1 \asymp \mathcal{F}_2$ iff $\mathcal{F}_1 \lesssim \mathcal{F}_2$ and $\mathcal{F}_2 \lesssim \mathcal{F}_1$. An example of such a lattice for the commitment task is given in Figure 3. In the following proposition we show an important property of $\mathscr{F}_T$: UC-realizing any point $\mathcal{F}$ of $\mathscr{F}_T$ would imply that any semilattice point dominated by $\mathcal{F}$ is also UC-realizable.

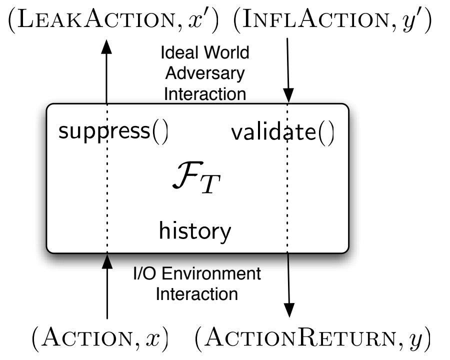

Figure 2: The canonical functionality: communication flows with the environment and adversary.

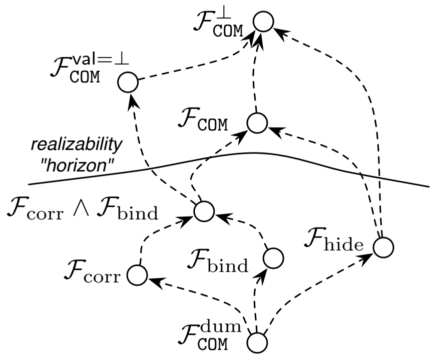
$\mathcal{F}_{\mathtt{COM}}^{\perp}$
$\mathcal{F}_{\mathtt{COM}}^{\mathrm{val}=\perp}$
$\mathcal{F}_{\mathtt{COM}}$
realizability "horizon"
$\mathcal{F}_{\mathrm{corr}} \wedge \mathcal{F}_{\mathrm{bind}}$
$\mathcal{F}_{\mathrm{hide}}$
$\mathcal{F}_{\mathrm{bind}}$
$\mathcal{F}_{\mathrm{corr}}$
$\mathcal{F}_{\mathtt{COM}}^{\mathrm{dum}}$

Figure 3: The lattice of canonical functionalities for the commitment task showing relations between the functionalities defined in Section 4.

Proposition 2.5. *If π_Σ UC-realizes $\mathcal{F}$, then π_Σ UC-realizes any $\mathcal{F}' \lesssim \mathcal{F}$.*

The language of a canonical functionality. For a given cryptographic task T, a canonical functionality for T defines a language over the symbols that are used by the functionality to communicate with the environment. We formalize the language as follows:

Definition 2.6. *Given a canonical functionality $\mathcal{F}_T$, an environment $\mathcal{Z}$ and an adversary $\mathcal{S}$, we define $L^{\mathrm{I/O}}_{\mathcal{F}_T,\mathcal{Z},\mathcal{S}} = \{w | w \in (\Sigma_T)^*$* so that w is a string of symbols equal to history of $\mathcal{F}_T$ in an execution with $\mathcal{Z}$ and $\mathcal{S}$ $\}$.

We may quantify the language over all possible environments $\mathcal{Z}$ and ideal world adversaries $\mathcal{S}$ in which case we will omit referencing them. Moreover we may consider only those strings in history of $\mathcal{F}_T$ for which the environment $\mathcal{Z}$ returns 1. In this case we will denote the language as $B^{\mathrm{I/O}}_{\mathcal{F}_T,\mathcal{Z},\mathcal{S}}$.

3 From Classical Security Definitions to Ideal Functionalities

Traditionally, correctness and privacy definitions of cryptographic tasks are given by specifying a game between the attacker and a "challenger" who controls different aspects of the cryptographic task. The attacker either tries to produce an undesired sequence of actions or attempts to deduce a hidden bit selected by the challenger. In the former case we call the interaction a *consistency game* while in the latter we call the interaction a *hiding game*. Examples of properties modeled with consistency games include completeness properties, the unforgeability of digital signatures, the binding property of commitments, the soundness property of zero-knowledge protocols etc. Hiding games, on the other hand are typically divided into two categories: *indistinguishability games* and *simulation games*. In the former case we have games such as the IND-CPA game for public-key encryption while in the latter we have games where the adversary tries to distinguish between a real or a simulated action; such games include the modeling of the zero-knowledge property. Here we show how to transform game-based definitions to ideal functionalities. In order to detail the transformation we first provide a formal definition of games below.

Game-based security definitions. A game-based definition G for a cryptographic task $T = \langle A_1, \ldots, A_k \rangle$ involves two PPT interactive Turing machines, the challenger C and the attacker A. The challenger additionally uses the actions of the cryptographic task as oracles. When the interaction terminates, a Turing machine called the judge[4] J reads the transcript of the interaction as well as the internal state of the challenger and decides which party won the game. We denote the success probability of the attacker when playing the game G by $\mathsf{Succ}^G_{\mathsf{A}}$. It equals the probability of the event that the judge decides that the attacker wins the game.

Consistency games are the most general ones and fit the general description given above. We say that a cryptographic scheme that implements a task T satisfies the property defined by a game G if for all PPT attackers A it holds that $\mathsf{Succ}^G_{\mathsf{A}}$ is a negligible function in λ.

In an *indistinguishability game*, the attacker focuses on a particular action of the cryptographic task. At some point of the interaction with the challenger, the attacker provides two input strings x_0, x_1 for the action where $x_0 = \langle x_0^L, x_0^R \rangle$, $x_1 = \langle x_1^L, x_1^R \rangle$ such that either the left or the right parts of the strings are required to be different by the challenger while the other parts are required to be equal (for example in the witness hiding game for zero-knowledge, $x_0^L = x_1^L$ will be the statement while x_0^R, x_1^R will be two distinct witnesses). In response, the challenger flips a coin b and executes the action that is attacked on input x_b. The interaction provides the output of the action to the attacker who is supposed to provide a guess b^* for b. The judge decides that the attacker wins whenever $b = b^*$. We say that a cryptographic scheme that implements a task T satisfies the property defined by the indistinguishability game G if for all PPT attackers A it holds that the function $|\mathsf{Succ}^G_{\mathsf{A}} - \frac{1}{2}|$ is a negligible function in λ.

The translation for simulation-based games is very similar to the above and is given in Section C.

3.1 Ideal functionalities from consistency games

Suppose that G is a consistency game for a cryptographic task $T = \langle A_1, \ldots, A_k \rangle$ that involves a challenger C, an attacker A and a judge J. Let Σ be any cryptographic scheme that implements the task T. Our goal is to obtain a canonical functionality $\mathcal{F}_G \in \mathscr{F}_T$ such that if a protocol π_Σ UC-realizes any $\mathcal{F} \gtrsim \mathcal{F}_G$ then the cryptographic scheme Σ satisfies the property defined by the game G. Our methodology proceeds in three steps: we first define an environment (and also the corresponding ideal world) based on the game G. Second,

[4]Typically, the functionality of the judge is incorporated as part of the challenger machine; we find it more convenient to specify it as a separate function.

based on this environment, we define a language that corresponds to the event where the attacker wins the game. Third, provided that the language is decidable, we obtain a canonical functionality by incorporating the language decider as part of the $\mathsf{validate}()$ predicate of the canonical functionality. We describe the three steps in more detail below.

Step 1: Defining the environment and simulator. We first present the transformation from the game G for a task T implemented by a scheme Σ to the corresponding environment $\mathcal{Z}_G^{\mathsf{A}}$ and the ideal world adversary $\mathcal{S}_G^{\Sigma}$. We say that the transformation is sound, provided that the judge J decides that the attacker wins the game if and only if the environment $\mathcal{Z}_G^{\mathsf{A}}$ returns 1 in an execution with $\mathcal{F}_T^{\mathrm{dum}}$ and $\mathcal{S}_G^{\Sigma}$. More specifically, it holds that $\Pr[\mathrm{IDEAL}_{\mathcal{F}_T^{\mathrm{dum}},\mathcal{Z}_G^{\mathsf{A}},\mathcal{S}_G^{\Sigma}}(1^\lambda) = 1] = \mathsf{Succ}_{\mathsf{A}}^{G}$.

First, we describe how we derive the environment $\mathcal{Z}_G^{\mathsf{A}}$ based on the game G. $\mathcal{Z}_G^{\mathsf{A}}$ will simulate both the attacker A and the challenger C; whenever C makes an oracle call to some action A_i, the environment $\mathcal{Z}_G^{\mathsf{A}}$ needs to invoke some entity π_T by issuing an $\textsc{Action}_i$ symbol. The program of C will be executed by $\mathcal{Z}_G^{\mathsf{A}}$ respecting the "packaging" of the task's actions in the abstract entity π_T. For example, in the unforgeability game for digital signatures, an oracle call to the key generation operation will result in issuing the symbol $(\textsc{KeyGen}, \langle S, sid\rangle)$ to a party called S, where S is a random name from the namespace for some random sid; subsequent calls by C to the signing oracle for a message m, will result in the symbols $(\textsc{Sign}, \langle S, sid\rangle, m)$ directed to the same party S. If the attacker A needs to play the role of some party of the cryptographic task T, $\mathcal{Z}_G^{\mathsf{A}}$ will need to spawn and corrupt a π_T entity and then simulate it according to the operation of A. In such case, $\mathcal{S}_G^{\Sigma}$ will mediate the corruption operation between $\mathcal{Z}_G^{\mathsf{A}}$ and the ideal functionality.

Second, we define an ideal world adversary $\mathcal{S}_G^{\Sigma}$ that will be paired with $\mathcal{Z}_G^{\mathsf{A}}$. $\mathcal{S}_G^{\Sigma}$ will interact with $\mathcal{Z}_G^{\mathsf{A}}$ to corrupt parties if the environment requests it and it will also provide influence action symbols whenever a leak action symbol occurs following the program of the scheme Σ.

Step 2: Defining the "bad language." This language will correspond to the event that the attacker wins the game. It is denoted by $B_{T,G}^{\mathrm{I/O}} \subseteq \bigcup_{\mathsf{A},\Sigma} L^{\mathrm{I/O}}_{\mathcal{F}_T^{\mathrm{dum}},\mathcal{Z}_G^{\mathsf{A}},\mathcal{S}_G^{\Sigma}}$ and contains those strings for which the environment $\mathcal{Z}_G^{\mathsf{A}}$ returns 1. Note that based on the soundness of the transformation of the game G to the environment $\mathcal{Z}_G^{\mathsf{A}}$ those strings correspond exactly to the event where the attacker A wins the game G against the challenger C. While this language captures the event that the attacker wins the game, it is not sufficient for describing the winning event within more complex executions because the bad sequence of symbols may be interleaved with other actions. We define this extended bad language as exactly those strings of $L^{\mathrm{I/O}}_{\mathcal{F}_T^{\mathrm{dum}}}$ that contain as a subsequence a string of $B_{T,G}^{\mathrm{I/O}}$, and denote it as $B_{T,G}^{\mathrm{ext}}$.

Step 3: Defining the ideal functionality. In order to define the class of canonical functionalities that capture the game G we need first to show that the extended bad language $B_{T,G}^{\mathrm{ext}}$ defined in step 2 is polynomial-time decidable. Then, given the decider D for the language, we define the canonical functionality $\mathcal{F}_G$ that captures the game G by requiring that $w \in B_{T,G}^{\mathrm{ext}}$ if and only if $\mathsf{validate}(w) = 0$; in other words, the function $\mathsf{validate}()$ simulates the decider D, and whenever the decider accepts the functionality halts.

Claim 3.1. *Suppose that a cryptographic scheme Σ implements a cryptographic task T and G is a consistency game for T. Then it holds that if π_Σ UC-realizes some $\mathcal{F} \gtrsim \mathcal{F}_G$, then Σ satisfies the property defined by G.*

We demonstrate the above claim for several fundamental cryptographic tasks in Section 4 and Section D.

3.2 Ideal functionalities from hiding games

Here we focus on indistinguishability games; the translation for simulation-based games is given in Section C. Let G be an indistinguishability game for a cryptographic task T. We show how to define a canonical functionality for the task that implies the indistinguishability property. In this case, our methodology proceeds in two steps: we first define an environment and an ideal world simulator based on the game G. Second, based on the environment's operation we define the canonical functionality by appropriately modifying the $\mathsf{suppress}$ function.

Step 1: Defining the environment and simulator. As in the case of consistency games, we define an environment $\mathcal{Z}_G^{\mathsf{A}}$ and simulator $\mathcal{S}_G^{\Sigma}$ based on the operation of the challenger C, the attacker A and the judge J. The transformation is identical to the one in step 1 of the previous subsection.

Step 2: Defining the canonical functionality. During any execution of the environment $\mathcal{Z}_G^{\mathsf{A}}$ with $\mathcal{S}_G^{\Sigma}$, it holds that it issues a symbol $\textsc{Action}_i$ with input x_b where b is a random bit selected by $\mathcal{Z}_G^{\mathsf{A}}$ and x_0, x_1 were provided by the attacker A (which is simulated by $\mathcal{Z}_G^{\mathsf{A}}$). Assuming that $x_0 = \langle x_0^L, x_0^R \rangle$ and $x_1 = \langle x_1^L, x_1^R \rangle$ and the game G contains the test $x_0^L = x_1^L$ and $x_0^R \neq x_1^R$, we define the suppress function for symbol $\mathsf{a} = (\textsc{Action}_i, \mathbf{P}, x_b)$ where $b \in \{0,1\}$ by $\mathsf{suppress}(\mathsf{a}) = \langle x_b^L, (-)^{|x_b^R|} \rangle$ (recall that $\mathsf{suppress}(\mathsf{a}) \prec x_b$).

Claim 3.2. *Suppose that a cryptographic scheme Σ implements a cryptographic task T and G is an indistinguishability game for T. Then it holds that if π_Σ UC-realizes some $\mathcal{F} \gtrsim \mathcal{F}_G$, then Σ satisfies the hiding property defined by game G.*

4 Applying the Methodology: Commitments

We now apply our methodology to several fundamental cryptographic tasks. Here we treat commitments, whose properties are modeled by both consistency and hiding games; the application to other tasks can be found in Section D.

Following Figure 1, any canonical functionality for commitment, $\mathcal{F}_{\texttt{COM}}$, is defined for two types of roles, the committer C and the verifier V, with two actions, $\textsc{Commit}$ and $\textsc{Open}$. The WF predicate for $\mathcal{F}_{\texttt{COM}}$ requires that a $\textsc{Commit}$ should precede $\textsc{CommitReturn}$ and an $\textsc{Open}$ should precede $\textsc{OpenReturn}$; moreover, $\textsc{Commit}$ should always precede $\textsc{Open}$ and $\textsc{CommitReturn}$ should always precede $\textsc{OpenReturn}$. The default output DO for $\mathcal{F}_{\texttt{COM}}$ is defined as follows: whenever history contains the symbols $(\textsc{Commit}, \langle C, V, sid \rangle, m)$ and $(\textsc{Open}, \langle C, V, sid \rangle)$ the default output is m, otherwise it is empty. Based on the above the dummy functionality $\mathcal{F}_{\texttt{COM}}^{\text{dum}}$ is defined (cf. Definition 2.1). We consider the properties of commitments in turn. For simplicity we consider single-move commitment protocol.

Correctness. This property can be modeled by a consistency game where the adversary chooses an m for which the committer produces a commitment that the verifier fails to accept.

Definition 4.1 (Correctness)**.** *A commitment scheme* $\Sigma(\texttt{COM}) = \langle \texttt{commit}, \texttt{verify} \rangle$ *is* correct *if for all PPT attackers* A, *it holds that* $\Pr[m \leftarrow \mathsf{A}(1^\lambda); (c, \xi) \leftarrow \texttt{commit}(m); \phi \leftarrow \texttt{verify}(c, m, \xi) : \phi = 0] \leq \mathsf{negl}(\lambda)$.

Game G_{corr} is as follows. The challenger C uses as oracles the algorithms $\texttt{commit}()$, $\texttt{verify}()$, and interacts with the attacker A: the attacker A outputs m; the challenger invokes the committer oracle with m and obtains $\langle c, \xi \rangle$; then the challenger queries the verifier oracle with $\langle c, \xi \rangle$ and obtains the response ϕ. The judge J decides that A wins the game if $\phi \neq 1$. We describe the three steps of the transformation outlined in Section 3.1.

Step 1. We construct an environment $\mathcal{Z}_{\text{corr}}^{\mathsf{A}}$ and the corresponding ideal world adversary $\mathcal{S}_{\text{corr}}^{\Sigma}$ based on game G_{corr}. In order to simulate the game, the environment first picks C, V from the namespace at random as well as a random sid. Then, it simulates A on input 1^λ; once A outputs m, $\mathcal{Z}_{\text{corr}}^{\mathsf{A}}$ sends $(\textsc{Commit}, \langle C, V, sid \rangle, m)$ to party C; when $\mathcal{Z}_{\text{corr}}^{\mathsf{A}}$ receives c from the adversary it simply relays it back. When it receives $(\textsc{CommitReturn}, \langle C, V, sid \rangle)$ from party V it sends $(\textsc{Open}, \langle C, V, sid \rangle)$ to party C. Finally, if $\mathcal{Z}_{\text{corr}}^{\mathsf{A}}$ receives $(\textsc{OpenReturn}, \langle C, V, sid \rangle, \langle m, 0 \rangle)$ from party V, it terminates with 1, otherwise, in any other case, with 0.

The ideal-world adversary $\mathcal{S}_{\text{corr}}^{\Sigma}$, when it receives $(\textsc{LeakCommit}, \langle C, V, sid \rangle, m)$ it simulates $\texttt{commit}()$ to obtain $\langle c, \xi \rangle$. It then shows c to the environment $\mathcal{Z}$ and when the environment responds with c, $\mathcal{S}_{\text{corr}}^{\Sigma}$ sends a symbol $(\textsc{InflCommit}, \langle C, V, sid \rangle)$ to the functionality. When it receives $(\textsc{LeakOpen}, \langle C, V, sid \rangle)$, it sends the symbol $(\textsc{InflOpen}, \langle C, V, sid \rangle, \texttt{verify}(c, m, \xi))$ to the functionality.

Step 2. For any correctness attacker A and scheme Σ, the environment $\mathcal{Z}_{\text{corr}}^{\mathsf{A}}$, the adversary $\mathcal{S}_{\text{corr}}^{\Sigma}$, and the dummy canonical commitment functionality together give rise to the language $L^{\text{I/O}}_{\mathcal{F}_{\texttt{COM}}^{\text{dum}}, \mathcal{Z}_{\text{corr}}^{\mathsf{A}}, \mathcal{S}_{\text{corr}}^{\Sigma}}$ (cf. Section 2).

We consider the subset of strings $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}}$ of the union of all the I/O languages quantified over all possible correctness attackers A and schemes Σ that contains exactly those strings that correspond to the case that the environment returns 1. Formally, $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}} \overset{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma} B^{\mathrm{I/O}}_{\mathcal{F}^{\mathrm{dum}}_{\mathtt{COM}},\mathcal{Z}^{\mathsf{A}}_{\mathrm{corr}},\mathcal{S}^{\Sigma}_{\mathrm{corr}}}$. We next prove the following characterization of this language as well as determine its time complexity:

Lemma 4.2. *(1)* $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{Commit}, \langle C, V, sid\rangle, m)(\textsc{CommitReturn}, \langle C, V, sid\rangle) \\ (\textsc{Open}, \langle C, V, sid\rangle)(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m, 0\rangle) \end{array} \right\}$*, and (2)* $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}}$ *is decidable in polynomial time.*

In order to obtain the bad language for the correctness property we extend $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}}$ as follows: $B^{\mathrm{ext}}_{\mathtt{COM},\mathrm{corr}} = \{w \in L^{\mathrm{I/O}}_{\mathcal{F}^{\mathrm{dum}}_{\mathtt{COM}}} \mid \exists w' \in B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}} \text{ such that } w' \preccurlyeq w\}$. Observe that $B_{\mathtt{COM},\mathrm{corr}}$ is also polynomial time decidable.

Step 3. We next define the class of ideal functionalities that corresponds to the correctness property.

Definition 4.3 (Canonical Functionality $\mathcal{F}_{\mathrm{corr}}$)**.** *The functionality* $\mathcal{F}_{\mathrm{corr}} \in \mathscr{F}_{\mathtt{COM}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\mathtt{COM}}$*, where (1)* $\mathsf{suppress}()$ *is the same as in* $\mathcal{F}^{\mathrm{dum}}_{\mathtt{COM}}$*, and (2)* $\mathsf{validate}(w) = 0$ *if and only if* $w \in B^{\mathrm{ext}}_{\mathtt{COM},\mathrm{corr}}$*.*

Theorem 4.4 (Translation Soundness)**.** *If* $\pi_{\Sigma(\mathtt{COM})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{corr}}$*, then* $\Sigma(\mathtt{COM})$ *is correct.*

In fact, the converse of the above theorem also holds hence for correctness the translation is tight.

Theorem 4.5. *If* $\Sigma(\mathtt{COM})$ *is correct, then* $\pi_{\Sigma(\mathtt{COM})}$ *realizes* $\mathcal{F}_{\mathrm{corr}}$*.*

Hiding. We now apply our methodology over the hiding indistinguishability game for commitment schemes.

Definition 4.6 (Hiding)**.** *A commitment scheme* $\Sigma(\mathtt{COM}) = \langle \mathtt{commit}, \mathtt{verify}\rangle$ *is* hiding *if for all PPT attackers* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*, it holds that* $\Pr[(m_0, m_1, st) \leftarrow \mathsf{A}_1(1^\lambda); b \xleftarrow{\mathrm{r}} \{0,1\}; (c, \xi) \leftarrow \mathtt{commit}(m_b); b^* \leftarrow \mathsf{A}_2(st, c) : b^* = b \wedge m_0 \neq m_1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$*.*

The above definition can be modeled as a hiding game G_{hide} for the task $\mathtt{COM}$ as follows. The challenger C is allowed to use algorithms $\mathtt{commit}(), \mathtt{verify}()$ as oracles, and interacts with the attacker $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. First A_1 produces a tuple $\langle m_0, m_1\rangle$, where $m_0 \neq m_1$. In response, the challenger randomly chooses a bit b and queries the $\mathtt{commit}()$ oracle with m_b to obtain $\langle c, \xi\rangle$. Then, C sends c to A_2 to obtain b^* as a guess of b. The judge J decides that A wins the game if $b^* = b$. We next proceed to apply the methodology of Section 3.2.

Step 1. We construct an environment $\mathcal{Z}^{\mathsf{A}}_{\mathrm{hide}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma}_{\mathrm{hide}}$ based on the game G_{hide} described above. In order to simulate the game, the environment first picks C, V from the namespace at random as well as a random sid. Then it requests the corruption of the party V and simulates A_1 on input 1^λ. Once A_1 produces $\langle m_0, m_1\rangle$, $\mathcal{Z}^{\mathsf{A}}_{\mathrm{hide}}$ flips a random coin b, gives to C the symbol $(\textsc{Commit}, \langle C, V, sid\rangle, m_b)$ and waits for the transmission from C to V that contains the commitment c. Then, $\mathcal{Z}^{\mathsf{A}}_{\mathrm{hide}}$ simulates A_2 on input c to obtain b^* and terminates with 1 if and only if $b = b^*$ and $m_0 \neq m_1$. The ideal world adversary $\mathcal{S}^{\Sigma}_{\mathrm{hide}}$, whenever it receives $(\textsc{LeakCommit}, \langle C, V, sid\rangle, m)$ it executes $\mathtt{commit}()$ on m and communicates the output of the protocol to the environment (similarly, it simulates the real world in any other respect).

Step 2. Based on the environment $\mathcal{Z}^{\mathsf{A}}_{\mathrm{hide}}$ we define the functionality class that corresponds to the hiding game:

Definition 4.7 (Canonical Functionality $\mathcal{F}_{\mathrm{hide}}$)**.** *The functionality* $\mathcal{F}_{\mathrm{hide}} \in \mathscr{F}_{\mathtt{COM}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}$*, where (1)* $\mathsf{validate}() = 1$ *always, and (2)* $\mathsf{suppress}(\mathsf{a}) = (-)^{|m|}$ *for* $\mathsf{a} = (\textsc{Commit}, \langle C, V, sid\rangle, m)$*.*

Theorem 4.8 (Translation Soundness)**.** *If* $\pi_{\Sigma(\mathtt{COM})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{hide}}$*, then* $\Sigma(\mathtt{COM})$ *satisfies hiding.*

The converse of the above theorem does not hold as hiding is not sufficiently strong to imply the UC-realization of $\mathcal{F}_{\mathrm{hide}}$. Nevertheless, if the hiding property is strengthened to equivocality (see Section D.1.2), then we are able to show that the converse holds for static adversaries.

Binding. We model the binding property of commitments in terms of extractability as this stronger formulation will enable a tight translation. See Section D.1.1 for the more standard formulation of binding.

Definition 4.9 (Binding in the sense of Extractability)**.** *A commitment scheme* $\Sigma(\mathtt{COM}) = \langle \mathtt{commit}, \mathtt{verify}\rangle$ *is* binding in the sense of extractability *if there exists a PPT* E*, for all PPT attackers* A *such that* $\Pr[(c, m_2, \xi_2) \leftarrow \mathsf{A}(1^\lambda); m_1 \leftarrow \mathsf{E}(c); \phi \leftarrow \mathtt{verify}(c, m_2, \xi_2) : \phi = 1 \wedge m_1 \neq m_2] \leq \mathsf{negl}(\lambda)$.

The above property can be modeled by a consistency game, G_{bind}, as follows. The challenger C uses as oracles the algorithms $\mathtt{commit}(), \mathtt{verify}()$ and the witness extractor E, and interacts with the attacker A: the attacker A outputs $\langle c, m_2, \xi_2\rangle$; the challenger simulates E on input c to obtain response m_1, and then queries the the verifier oracle with the pair $\langle m_2, \xi_2\rangle$ to obtain response ϕ. The judge J decides that A wins the game if $m_1 \neq m_2$ and $\phi = 1$. Next we describe the three steps of the transformation outlined in Section 3.1.

Step 1. We construct an environment $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ based on the game G_{bind} described above. In order to simulate the game, the environment first picks C, V from the namespace at random as well as a random sid. Then, the environment requests the corruption of party C and simulates A on input 1^λ; once A outputs $\langle c, m_2, \xi_2\rangle$, the environment sends c to $\mathcal{S}$ and requests from $\mathcal{S}$ to make a commitment on behalf of the party C to the party V. The environment simulates E on input c and obtains m_1. After receiving a symbol $(\textsc{CommitReturn}, \langle C, V, sid\rangle)$ from party V, the environment sends $\langle m_2, \xi_2\rangle$ to $\mathcal{S}$ and requests from $\mathcal{S}$ to open the commitment of C to V. If $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ receives $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ from V, where m_2 is different from m_1, it terminates with 1; otherwise it terminates with 0.

We next describe the operation of $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$. When $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ receives the corruption request from the environment it relays it to the functionality. When $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ receives c from the environment, it runs the extractor E on input c to compute m_1, and then gives a symbol $(\textsc{Commit}, \langle C, V, sid\rangle, m_1)$ to the functionality on behalf of the corrupted party C; then, $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ sends symbol $(\textsc{InflCommit}, \langle C, V, sid\rangle)$ to the functionality. When $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ receives $\langle m_2, \xi_2\rangle$ from the environment it sends a symbol $(\textsc{InflOpen}, \langle C, V, sid\rangle, \langle m_2, \mathtt{verify}(c, m_2, \xi_2)\rangle)$ to the functionality.

Step 2. For any extractability attacker A, any scheme Σ, any extractor E, the environment $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ and the adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ and the dummy canonical commitment functionality together give rise to the language $L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathtt{COM}}, \mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}, \mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}}$. We consider the subset of strings $B^{\text{I/O}}_{\mathtt{COM},\text{bind}}$ of the union of all the I/O languages quantified over all possible A, E and Σ that contains exactly those strings corresponds to the case that the environment returns 1. Formally, $B^{\text{I/O}}_{\mathtt{COM},\text{bind}} \stackrel{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma,\mathsf{E}} B^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathtt{COM}}, \mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}, \mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}}$ We now show:

Lemma 4.10. *(1)* $B^{\text{I/O}}_{\mathtt{COM},\text{bind}} = \left\{ w \;\middle|\; \begin{array}{c} w = (\textsc{Commit}, \langle C, V, sid\rangle, m)(\textsc{CommitReturn}, \langle C, V, sid\rangle) \\ (\textsc{Open}, \langle C, V, sid\rangle)(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m', 1\rangle) \\ \text{such that } m \neq m' \text{ for some } m, m' \end{array} \right\}$*, and*
(2) $B^{\text{I/O}}_{\mathtt{COM},\text{bind}}$ *is decidable in polynomial time.*

In order to obtain the bad language for the binding property we extend $B^{\text{I/O}}_{\mathtt{COM},\text{bind}}$ as follows: $B^{\text{ext}}_{\mathtt{COM},\text{bind}} = \{w \in L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathtt{COM}}} \mid \exists w' \in B^{\text{I/O}}_{\mathtt{COM},\text{bind}} \text{ such that } w' \preccurlyeq w\}$. Observe that $B_{\mathtt{COM},\text{bind}}$ is also polynomial-time decidable.

Step 3. We next define the class of ideal functionalities that corresponds to the binding property.

Definition 4.11 (Canonical Functionality $\mathcal{F}_{\text{bind}}$)**.** *The functionality* $\mathcal{F}_{\text{bind}} \in \mathscr{F}_{\mathtt{COM}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\mathtt{COM}}$ *where (1)* $\mathsf{suppress}()$ *is the same as in* $\mathcal{F}^{\text{dum}}_{\mathtt{COM}}$*, and (2)* $\mathsf{validate}(w) = 0$ *if and only if* $w \in B^{\text{ext}}_{\mathtt{COM},\text{bind}}$.

Theorem 4.12 (Translation Soundness)**.** *If* $\pi_{\Sigma(\mathtt{COM})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\text{bind}}$ *against static adversaries, then* $\Sigma(\mathtt{COM})$ *is binding in the sense of extractability.*

In fact, the converse of the above theorem also holds hence for binding in the sense of extractability the translation is tight.

Theorem 4.13. *If* $\Sigma(\mathtt{COM})$ *is binding in the sense of extractability, then* $\pi_{\Sigma(\mathtt{COM})}$ *realizes* $\mathcal{F}_{\text{bind}}$.

The canonical ideal commitment functionality. The (canonical) ideal commitment functionality would equal $\mathcal{F}_{\text{corr}} \wedge \mathcal{F}_{\text{bind}} \wedge \mathcal{F}_{\text{hide}}$. In light of the translation soundness theorems above and Proposition 2.5 we obtain:

Corollary 4.14. *If $\pi_{\Sigma(\texttt{COM})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\text{corr}} \wedge \mathcal{F}_{\text{bind}} \wedge \mathcal{F}_{\text{hide}}$, then the commitment scheme $\Sigma(\texttt{COM})$ satisfies correctness, binding, and hiding.*

Functionality $\mathcal{F}_{\text{corr}} \wedge \mathcal{F}_{\text{bind}} \wedge \mathcal{F}_{\text{hide}}$ turns out to be equivalent (in the sense of UC-emulation) to the commitment functionality as it appears in [Can05]. We note that our canonical functionality easily generalizes to the multi-session setting (see Section D.1.3), and in this case one can show that it implies non-malleability (in the sense that any π_Σ realizing $\mathcal{F}_{\text{corr}} \wedge \mathcal{F}_{\text{bind}} \wedge \mathcal{F}_{\text{hide}}$ would imply that $\Sigma(\texttt{COM})$ is non-malleable). Interestingly, we also show that merely realizing $\hat{\mathcal{F}}_{\text{hide}}$ implies the (weaker) non-malleability definition put forth in [PR05]. This demonstrates the power of the methodology to establish (fine-grain) relations between security properties.

Further treatment of commitments appears in Section D, where we also apply the methodology to other cryptographic tasks — signatures, zero-knowledge and oblivious transfer.

References

[Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptology*, 4(2):75–122, 1991.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1992.

[BH04] Michael Backes and Dennis Hofheinz. How to break and repair a universally composable signature functionality. In Kan Zhang and Yuliang Zheng, editors, *ISC 2004*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004. http://eprint.iacr.org/2003/240.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW 2004*, pages 219–235. IEEE Computer Society, 2004. Full version at http://eprint.iacr.org/2003/239/.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Cryptology ePrint Archive, Report 2000/067*, December 2005. Latest version at http://eprint.iacr.org/2000/067/.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC 2002*, pages 494–503. ACM, 2002. Full version at http://www.cs.biu.ac.il/~lindell/PAPERS/uc-comp.ps.

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In Carl Pomerance, editor, *CRYPTO 1987*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer, 1987.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000. Preliminary version appears at STOC 1991.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC 1987*, pages 218–229. ACM, 1987.

[HK07] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In *Cryptology ePrint Archive: Report 2007/118*, 2007. Preliminary version appears in Eurocrypt 2005.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991. Long version at `http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-511.pdf`.

[PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS 2005*, pages 563–572. IEEE Computer Society, 2005. Available at `http://www.eecs.harvard.edu/~alon/PAPERS/conc-nmc/conc-nmc.ps`.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, 2001.

[Rab81] Michael Rabin. How to exchange secrets by oblivious transfer. In *Technical Report TR-81*. Harvard University, 1981.

A The Universal Composability Framework

The UC framework was proposed by Canetti for defining the security and composition of protocols [Can01]. In this framework one first defines an "ideal functionality" of a protocol, and then proves that a particular implementation of this protocol operating in a given computational environment securely realizes this ideal functionality. The basic entities involved are n players $P_1, \ldots, P_n$, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The real execution of a protocol π, run by the players in the presence of $\mathcal{A}$ and an environment machine $\mathcal{Z}$, with input z, is modeled as a sequence of *activations* of the entities. The environment $\mathcal{Z}$ is activated first, generating in particular the inputs to the other players. Then the protocol proceeds by having $\mathcal{A}$ exchange messages with the players and the environment. Finally, the environment outputs one bit, which is the output of the protocol.

The security of the protocols is defined by comparing the real execution of the protocol to an ideal process in which an additional entity, the ideal functionality $\mathcal{F}$, is introduced; essentially, $\mathcal{F}$ is an incorruptible trusted party that is programmed to produce the desired functionality of the given task. The players are replaced by dummy players, who do not communicate with each other; whenever a dummy player is activated, it forwards its input to $\mathcal{F}$. Let $\mathcal{A}$ denote the adversary in this idealized execution. As in the real-life execution, the output of the protocol execution is the one-bit output of $\mathcal{Z}$. Now a protocol π *securely realizes* an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-execution adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π in the real-life execution, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal execution. More precisely, if the two binary distribution ensembles, $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{A}$ and players running protocol π (resp., adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$), are computationally indistinguishable (denoted $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$). For further details on the UC framework refer to [Can05].

B Proofs

(Proof of Theorem 2.2). Consider a task T, and its well-formedness predicate WF_T. We construct a scheme Σ that implements T such that π_Σ realizes the dummy functionality $\mathcal{F}_T^{\text{dum}}$. We first give description for π_T, then we design the scheme Σ; the protocol π_Σ will be obtained by implementing all actions of π_T with the algorithms of Σ. A π_T entity P maintains an array $\mathsf{history}$, initially empty, which is used to record the entity's action symbols. In particular, when P receives a symbol $(\textsc{Action}, \mathbf{P}, x)$ from the environment, it records the symbol into its $\mathsf{history}$, runs the predicate WF_T over $\mathsf{history}$, and if the predicate returns 0, then the input is ignored, and the input will be removed from its $\mathsf{history}$. Whenever required by the action, the π_T entity returns an output symbol $(\textsc{ActionReturn}, \mathbf{P}, y)$, using the WF_T predicate to ensure well-formedness. We next describe the scheme Σ implementing the cryptographic task T. Recall that for each action T specifies a domain and range; given that we are only interested in designing a protocol realizing the dummy functionality we will simply define each action of Σ to map every input of the action domain $D_i^{(\lambda)}$ to an element of the action range $R_i^{(\lambda)}$. This captures the case of a non-interactive action. For interactive actions, say between two parties, Σ provides a two party protocol where the two parties coordinate according to the input-output behavior of the action. This completes the description of Σ that together with π_T defines the protocol π_Σ.

Next, we construct an ideal world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish an execution involving π_Σ and the real world adversary from an execution of $\mathcal{F}_T^{\text{dum}}$ and the ideal world adversary. The construction of $\mathcal{S}$ is as follows: $\mathcal{S}$ will simply perform a faithful simulation of the real world execution with the protocol π_Σ and the real-world adversary. This is possible as the canonical dummy functionality relays all (valid) I/O from the environment without any modifications. We next prove that no environment $\mathcal{Z}$ can distinguish the ideal from the real world for the above simulator $\mathcal{S}$ and in fact the simulation is perfect.

Observe that the only difference between the real world execution and the ideal world execution is the fact that the verification of the well-formedness predicate in the real world is distributed amongst the parties whereas in the ideal world it is handled by the canonical functionality. Observe that if the combined history of all parties in an ideal world execution is well-formed then the local history of each party in the real world will also be well-formed (as the same WF_T predicate is used globally and locally and the predicate is only sensitive in the order of symbols). Note that the reverse direction is not necessarily true; indeed a set of well-formed local histories may not be composed to a global history that is well-formed (and this may provide an opportunity for an adversarial environment to distinguish the real from the ideal world). Nevertheless, this is not the case due to the fact that a Σ scheme, specifically the coordination component of the protocol implementation of interactive actions, will ensure that the composition (according to the real order of events as induced by the adversary) of the local histories of all parties in a real world execution will result in a well-formed global history.

In the case of corrupted parties observe that the composed global history of a real world execution might cease to be well-formed as it may not include the local histories of corrupted parties (which are handled internally by the adversary). This discrepancy, however, will not result in any distinguishing advantage as the simulator $\mathcal{S}$ has the power to insert symbols in the canonical functionality's history that follow the actions of corrupted parties and thus maintain the well-formedness of the functionality's history.

Based on the above we conclude that the ideal world adversary $\mathcal{S}$ is performing a perfect simulation of the ideal world when interacting with $\mathcal{F}_T^{\text{dum}}$ and thus π_Σ is a UC-realization of $\mathcal{F}_T^{\text{dum}}$. □

(Proof of Proposition 2.5). Let π be a protocol that UC-realizes $\mathcal{F}$ and let $\mathcal{F}'$ be any functionality such that $\mathcal{F}' \lesssim \mathcal{F}$ which means that $\mathcal{F} = \mathcal{F}' \wedge \mathcal{F}''$ for some $\mathcal{F}'' \in \mathscr{F}_T$. Let $\mathcal{F}' = \mathcal{F}_T^{\mathsf{suppress}_1,\mathsf{validate}_1}, \mathcal{F}'' = \mathcal{F}_T^{\mathsf{suppress}_2,\mathsf{validate}_2} \in \mathscr{F}_T$. To prove the proposition, it suffices to prove the following statement that any protocol π that UC-realizes $\mathcal{F}$ also UC-realizes $\mathcal{F}'$.

To prove that π UC-realizes $\mathcal{F}'$, we need to show that for any $\mathcal{A}'$ there is an ideal world adversary $\mathcal{S}'$ such that for all $\mathcal{Z}'$, $\text{IDEAL}_{\mathcal{F}',\mathcal{S}',\mathcal{Z}'} \approx \text{REAL}_{\pi,\mathcal{A}',\mathcal{Z}'}$. Notice that based on the condition that protocol π realizes $\mathcal{F}$, for any $\mathcal{A}$ there is an ideal world adversary $\mathcal{S}$ such that for all $\mathcal{Z}$, $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$.

Given a real world adversary $\mathcal{A}'$ for the protocol π, there exists an $\mathcal{S}$ from the premise of the proposition that simulates it in the ideal world interacting with $\mathcal{F}$. We construct an $\mathcal{S}'$ that interacts with $\mathcal{F}'$ as follows: $\mathcal{S}'$ simulates $\mathcal{S}$ in its interface with the functionality $\mathcal{F}'$ with the following modification: each time when $\mathcal{F}'$ has input $\mathsf{a} = (\textsc{Action}, \mathbf{P}, x)$ it gives to the adversary the symbol $(\textsc{LeakAction}, \mathbf{P}, x_1)$ where $x_1 = \mathsf{suppress}_1(\mathsf{a})$; given this symbol, $\mathcal{S}'$ computes $x_2 = \mathsf{suppress}_2(\textsc{Action}, \mathbf{P}, x_1)$ and gives the symbol $(\textsc{LeakAction}, \mathbf{P}, x_2)$ to $\mathcal{S}$. This completes the description of $\mathcal{S}'$.

Given an environment $\mathcal{Z}'$ we will show that $\text{IDEAL}_{\mathcal{F}',\mathcal{S}',\mathcal{Z}'} \approx \text{REAL}_{\pi,\mathcal{A}',\mathcal{Z}'}$. From the premise of the proposition we know that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}'} \approx \text{REAL}_{\pi,\mathcal{A}',\mathcal{Z}'}$, thus it suffices to show $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}'} \approx \text{IDEAL}_{\mathcal{F}',\mathcal{S}',\mathcal{Z}'}$.

To each run of $\mathcal{F}$ with $\mathcal{S}$ and $\mathcal{Z}'$ we can correspond a run of $\mathcal{F}'$ with $\mathcal{S}'$ and $\mathcal{Z}'$; observe that the correspondence will preserve the history of the canonical functionality, i.e., the history of $\mathcal{F}$ in the run with $\mathcal{S}$ and $\mathcal{Z}'$ will be the same in the corresponding run of $\mathcal{F}'$ with $\mathcal{S}'$ and $\mathcal{Z}'$ (the environment is the same in both cases and $\mathcal{S}'$ operates identically to $\mathcal{S}$ in terms of the way it influences the functionality). Thus, given that the event that $\mathsf{validate}_2(\mathsf{history}) = 0$ happens with negligible probability over all runs of $\mathcal{F}$ with $\mathcal{S}$ and $\mathcal{Z}'$ (since this a real world simulation and whenever this event happens the functionality $\mathcal{F}$ returns an error symbol), it follows that it also happens with negligible probability over the runs of $\mathcal{F}'$ with $\mathcal{S}'$ and $\mathcal{Z}'$. Consider the event that $\mathcal{Z}'$ returns 1 over all runs of $\mathcal{F}$ with $\mathcal{S}$ and $\mathcal{Z}'$ and observe that its probability is the same to the event that $\mathcal{Z}'$ returns 1 over all runs of $\mathcal{F}'$ with $\mathcal{S}'$ and $\mathcal{Z}'$ where both events are taken over the conditional space where $\mathsf{validate}_2(\mathsf{history}) = 1$. Given that $\mathsf{validate}_2(\mathsf{history}) = 0$ is a negligible probability event in either space the proof of the proposition follows. □

(Proof of Lemma 4.2). (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\text{I/O}}_{\mathtt{COM},\text{corr}} \subseteq Y$. Let w be any string in $B^{\text{I/O}}_{\mathtt{COM},\text{corr}}$; then it holds that there exist A, Σ such that w equals the history string in the ideal world execution of the environment $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{corr}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\mathtt{COM}}$. Based on the definition of the environment $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$ and the adversary $\mathcal{S}^{\Sigma}_{\text{corr}}$, we know that the symbol $(\textsc{Commit}, \langle C, V, \mathit{sid}\rangle, m)$ from dummy party C will be recorded into history in

the dummy functionality; and then after receiving an InflCommit symbol from the adversary, the symbol $(\textsc{CommitReturn}, \langle C, V, \mathit{sid}\rangle)$ will be recorded in history and be sent to the dummy party V by the functionality. Later a symbol $(\textsc{Open}, \langle C, V, \mathit{sid}\rangle)$ will be recorded into history; and then after an InflOpen symbol, the symbol $(\textsc{OpenReturn}, \langle C, V, \mathit{sid}\rangle, \langle m, 0\rangle)$ will be recorded in history. It follows that the string w belongs to the set Y.

Second we need to show $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}} \supseteq Y$. Let w be any string in Y. We will construct A, Σ such that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\mathrm{corr}}$ with adversary $\mathcal{S}^{\Sigma}_{\mathrm{corr}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathtt{COM}}$ it holds that history $= w$. Given that $w \in Y$ it holds that there exist m such that $w = (\textsc{Commit}, \langle C, V, \mathit{sid}\rangle, m)(\textsc{CommitReturn}, \langle C, V, \mathit{sid}\rangle)(\textsc{Open}, \langle C, V, \mathit{sid}\rangle)(\textsc{OpenReturn}, \langle C, V, \mathit{sid}\rangle, \langle m, 0\rangle)$. Define A output always m. Define `commit` that on input $\langle m, \xi\rangle$ returns c and the `verify` that on input $\langle c, m, \xi\rangle$ returns 0. It follows immediately that the history string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\mathrm{corr}}$ with adversary $\mathcal{S}^{\Sigma}_{\mathrm{corr}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathtt{COM}}$ would equal w.

(2) It is easy to show the language $B^{\mathrm{I/O}}_{\mathtt{COM},\mathrm{corr}}$ is decidable. □

(Proof of Theorem 4.4). By contradiction, assume scheme $\Sigma(\mathtt{COM})$ is not correct. We need to construct an environment $\mathcal{Z}$ to distinguish the two worlds with non-negligible probability. Based on the successful correctness attacker A, we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A}}_{\mathrm{corr}}$ as defined above. Notice that in the real world, A is a successful correctness attacker against $\Sigma(\mathtt{COM})$, so $\mathcal{Z}$ outputs 1 with non-negligible probability. However in the ideal world, $\langle m, 0\rangle$ would cause any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{corr}}$ to halt, so the environment $\mathcal{Z}$ can never output 1. Therefore the constructed $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

(Proof of Theorem 4.5). Given that no attacker A can win the correctness game above, we need to show that there exists a adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. The adversary $\mathcal{S}$ is designed as the generic adversary for commitment task, which means $\mathcal{S}$ will simulate the real world inside. Each time, when $\mathcal{S}$ obtains $(\textsc{LeakCommit}, \langle C, V, \mathit{sid}\rangle, m)$ from $\mathcal{F}_{\mathrm{corr}}$, it simulates party C inside to commit such m into c, and sends the commitment value c to party V; when the simulated V receives the commitment value c from party C, the ideal world adversary $\mathcal{S}$ sends symbol $(\textsc{CommitReturn}, \langle C, V, \mathit{sid}\rangle)$ to $\mathcal{F}_{\mathrm{corr}}$; later when $\mathcal{S}$ obtain the LeakOpen symbol from the functionality, it simulates party C to open the commitment value c to the m to party V; when the simulated V receives the opening from party C, party V produces the verification result $\langle m, \phi\rangle$, and the adversary $\mathcal{S}$ sends $(\textsc{InflOpen}, \langle C, V, \mathit{sid}\rangle, \langle m, \phi\rangle)$ to $\mathcal{F}_{\mathrm{corr}}$. Furthermore $\mathcal{S}$ corrupts the parties following $\mathcal{Z}$'s instruction. Whenever $\mathcal{S}$ receives a commitment value c from a corrupted committer C, $\mathcal{S}$ randomly select m' and play the role of the committer C to send $(\textsc{Commit}, \langle C, V, \mathit{sid}\rangle, m')$ to $\mathcal{F}_{\mathrm{corr}}$, and receives the corresponding LeakCommit symbol from the functionality; later when $\mathcal{S}$ receives from the corrupted committer C the opening for the c, $\mathcal{S}$ simulates party V with such opening value, and if V returns $\langle m, \phi\rangle$, then $\mathcal{S}$ sends $(\textsc{InflOpen}, \langle C, V, \mathit{sid}\rangle, \langle m, \phi\rangle)$ to $\mathcal{F}_{\mathrm{corr}}$.

Assume $\pi_{\Sigma(\mathtt{COM})}$ cannot realize $\mathcal{F}_{\mathrm{corr}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$ inside; and A further simulates the real world for the copy of $\mathcal{Z}$.

We let F denote the event that in a run of $\pi_{\Sigma(\mathtt{COM})}$ with $\mathcal{Z}$, no party is corrupted, and the commitment value c is computed based on the witness m and some coins ξ, but later c cannot be opened to $\langle m, \xi\rangle$, i.e., $\langle c, m, \xi\rangle$ cannot be verified. Observe that if event F does not occur and $\mathcal{F}_{\mathrm{corr}}$ does not halt, the simulated $\mathcal{Z}$ cannot distinguish the two worlds; recall that $\mathcal{F}_{\mathrm{corr}}$ halts only when $m' = m$ and $\phi = 0$; notice that $m' = m$ holds with only negligible probability, which means $\mathcal{F}_{\mathrm{corr}}$ halts with negligible probability. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful correctness attacker. □

(Proof of Theorem 4.8). By contradiction, we assume $\Sigma(\mathtt{COM})$ is not hiding, i.e., there exists a successful hiding attacker A who can guess the hidden bit b with non-negligible probability higher than $1/2$. Now we need to construct an environment that distinguish the two worlds with non-negligible probability. Based on the successful hiding attacker A, we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A}}_{\mathrm{hide}}$ as defined above. Notice that in the real world the value c will

be produced based on m_b according to the commitment protocol and since A is a successful hiding attacker, $\mathcal{Z}$ will output 1 with probability bounded away from $1/2$ by a non-negligible fraction; on the other hand, in the ideal world for any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\text{hide}}$, since any such functionality will suppress the m_b entirely in response to the $(\textsc{Commit}, \langle C, V, sid\rangle, m_b)$ symbol, no matter how the adversary $\mathcal{S}$ is designed (note that $\mathcal{S}$ has adversarial role in this proof), c will be produced independently of b, therefore even an unbounded A will not be able to influence the output based on b. It follows that $\mathcal{Z}$ will output 1 with probability $1/2$. It follows that $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

(Proof of Lemma 4.10). (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\text{I/O}}_{\texttt{COM},\text{bind}} \subseteq Y$. Let w be any string in $B^{\text{I/O}}_{\texttt{COM},\text{bind}}$; then it holds that there exist $\mathsf{A}, \Sigma, \mathsf{E}$ such that w equals the history string in the ideal world execution of the environment $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ with adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\texttt{COM}}$. Based on the definition of the environment $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ and the adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$, we know that the symbol $(\textsc{Commit}, \langle C, V, sid\rangle, m_1)$ will be patched by the adversary into history in the dummy functionality; and then after receiving an $\textsc{InflCommit}$ symbol from the adversary, the symbol $(\textsc{CommitReturn}, \langle C, V, sid\rangle)$ will be recorded in history and be sent to the dummy party V by the functionality. Later a symbol $(\textsc{Open}, \langle C, V, sid\rangle)$ will be patched into history; and then after an $\textsc{InflOpen}$ symbol, the symbol $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ will be recorded in history. It follows that the string w belongs to the set Y.

Second we need to show $B^{\text{I/O}}_{\texttt{COM},\text{bind}} \supseteq Y$. Let w be any string in Y. We will construct $\mathsf{A}, \Sigma, \mathsf{E}$ such that in the ideal world execution of $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ with adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\texttt{COM}}$ it holds that history $=$ w. Given that $w \in Y$ it holds that there exist m_1, m_2 such that $w = (\textsc{Commit}, \langle C, V, sid\rangle, m_1)(\textsc{CommitReturn}, \langle C, V, sid\rangle)(\textsc{Open}, \langle C, V, sid\rangle)(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ with $m_1 \neq m_2$. Let c, ξ_2 be a random string and define A output always $\langle c, m_2, \xi_2\rangle$. Next define E as follows: given c it returns m_1; for any other input it returns a random string. Finally define `commit` that on input $\langle m_2, \xi_2\rangle$ returns c and the `verify` that on input $\langle c, m_2, \xi_2\rangle$ returns 1. It follows immediately that the history string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ with adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\texttt{COM}}$ would equal w.

(2) It is easy to show the language $B^{\text{I/O}}_{\texttt{COM},\text{bind}}$ is decidable. □

(Proof of Theorem 4.12). Given that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds, we need to show that there exists a witness extractor E such that no A can open a valid commitment value in a way that disagrees with the extractor E with non-negligible probability.

We construct an extractor E as follows. Given input c, E simulates $\mathcal{S}$. E gives $\mathcal{S}$ the symbol $(\textsc{Corrupt}, P)$ as well as the instruction to deliver c to the receiver V as the commitment value produced by the corrupted committer C. Whenever E receives the symbol $(\textsc{Commit}, \langle C, V, sid\rangle, m_1)$ produced by $\mathcal{S}$ on behalf of the corrupted C, E sends to $\mathcal{S}$ the symbol $(\textsc{LeakCommit}, \langle C, V, sid\rangle)$ on behalf of the functionality. If E receives the symbol $(\textsc{InflCommit}, \langle C, V, sid\rangle)$ from $\mathcal{S}$, E outputs m_1, otherwise outputs $\perp$. This completes the description of E.

Assume for the sake of contradiction that the scheme $\Sigma(\texttt{COM})$ is not binding in the sense of extractability, i.e., for the extractor E defined above there exists a successful binding attacker A (as in the definition of binding in the sense of extractability) that wins the game with non-negligible probability β. That is $\Pr[(c, m_2, \xi_2) \leftarrow \mathsf{A}(1^\lambda); \phi \leftarrow \texttt{verify}(c, m_2, \xi_2); m_1 \leftarrow \mathsf{E}(c) : m_1 \neq m_2 \wedge \phi = 1] = \beta$.

We now construct an environment $\mathcal{Z}$ that distinguishes the real and the ideal worlds with non-negligible probability thus deriving a contradiction. The construction of $\mathcal{Z}$ is similar to $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\text{bind}}$ in step 1 of the translation. The environment $\mathcal{Z}$ first corrupts party C; then the environment simulates A as follows: when A outputs a triple $\langle c, m_2, \xi_2\rangle$, the environment simulates E on input c to obtain m_1. Then, the environment $\mathcal{Z}$ gives c to the adversary as the commitment value produced by the corrupted party C. Then, after receiving $(\textsc{CommitReturn}, \langle C, V, sid\rangle)$ from party V, $\mathcal{Z}$ gives $\langle m_2, \xi_2\rangle$ to the adversary as the opening for the commitment value c. When it receives $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ from party V, the environment simulates the operation of V on inputs c and $\langle m_2, \xi_2\rangle$ and obtains the output ϕ. If $m_1 \neq m_2$ and $\phi = 1$, the environment terminates with 1; in any other case, it terminates with 0. This completes the description of $\mathcal{Z}$.

First we consider the case that $\mathcal{Z}$ interacts with the real world. Define R_1 as the event that in the real world execution, the tuple produced by A is valid, and we let α_{r1} denote the probability the event occurs, i.e., $\alpha_{r1} = \Pr[R_1] = \Pr[(c, m_2, \xi_2) \leftarrow \mathsf{A}(1^\lambda); \phi \leftarrow \mathtt{verify}(c, m_2, \xi_2); m_1 \leftarrow \mathsf{E}(c) : \phi = 1]$. Define R_2 as the event in the conditional space on R_1 when the extractor E returns a message m_1 such that $m_1 \neq m_2$; let α_{r2} denote the probability of the event, i.e., $\alpha_{r2} = \Pr[R_2]$. It follows that $\beta = \alpha_{r1}\alpha_{r2}$. Define R_3 as the event in the conditional space on R_1 when the environment receives $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$; let α_{r3} denote the probability of the event, i.e., $\alpha_{r3} = \Pr[R_3]$. When R_1 occurs, given that verification is deterministic, the receiver V on input c and $\langle m_2, \xi_2\rangle$, always returns $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ to the environment, which means that the event R_3 occurs with probability 1, i.e., we have $\alpha_{r3} = 1$. It follows that the probability that $\mathcal{Z}$ outputs 1 when it interacts with the real world $\alpha_r = \Pr[R_1]\Pr[R_3 R_2]$. Note that when R_1 occurs, R_3 will occur with probability 1, and at the same time R_2 will occur with probability α_{r2}, so we have $\Pr[R_3 R_2] = 1 \cdot \alpha_{r2} = \alpha_{r2}$. As a result $\alpha_r = \Pr[R_1]\Pr[R_3 R_2] = \alpha_{r1}\alpha_{r2} = \beta$. We have $\alpha_{r1} \geq \beta$ and $\alpha_{r2} \geq \beta$; since β is non-negligible, so are α_{r1}, α_{r2}. Further, we have $\alpha_r = \beta$.

Next we consider the case that $\mathcal{Z}$ interacts with the ideal world. We define I_1 as the event that in the ideal world execution the tuple produced by A is valid and we let $\alpha_{i1} = \Pr[I_1]$. We define I_2 as the event in the conditional space on I_1 when the extractor E outputs m_1 such that $m_1 \neq m_2$; let $\alpha_{i2} = \Pr[I_2]$. Notice that $\alpha_{i1} = \alpha_{r1}$ and $\alpha_{i2} = \alpha_{r2}$ since the events I_1, I_2 and R_1, R_2 are in correspondence. Define I_3 as the event in the conditional space on I_1 when the environment receives $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ and denote $\alpha_{i3} = \Pr[I_3]$.

Claim: The events I_2 and I_3 are complementary. Recall that both I_2 and I_3 are over the conditional space on I_1. Whenever event I_2 occurs we have that E fails to extract the message that will be opened by the corrupted prover; this means that $\mathcal{S}$ either is not providing the symbol $(\textsc{CommitReturn}, \langle C, V, sid\rangle)$ to the functionality, or in case it does it the message provided by $\mathcal{S}$ in the $(\textsc{Commit}, \langle C, V, sid\rangle, m_1)$ symbol is such that $m_1 \neq m_2$. On the other hand, whenever the event I_3 occurs we have that the environment receives an $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$. This means that the ideal world adversary $\mathcal{S}$ given valid (c, m_2, ξ_2) makes the functionality to output $(\textsc{OpenReturn}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ to the environment. This can only happen when $\mathcal{S}$ produces the $(\textsc{InflOpen}, \langle C, V, sid\rangle, \langle m_2, 1\rangle)$ symbol and at the same time the history of the functionality contains the symbols $(\textsc{Commit}, \langle C, V, sid\rangle, m_2)$ and $(\textsc{CommitReturn}, \langle C, V, sid\rangle)$. It is easy to see that I_2, I_3 are complementary.

Next we compute the probability that $\mathcal{Z}$ outputs 1 when it interacts with the ideal world as follows $\alpha_i = \Pr[I_1]\Pr[I_3 I_2]$. When I_1 occurs, I_2 occurs with probability α_{i2}, and at the same time, I_3 occurs with probability α_{i3}; thus $\Pr[I_2 I_3] = \alpha_{i2}\alpha_{i3}$ given that the events are independents. Based on the claim above, $\alpha_{i2} + \alpha_{i3} = 1$. Thus we have $\alpha_i = \Pr[I_1]\Pr[I_2 I_3] = \alpha_{i1}\alpha_{i2}\alpha_{i3} = \alpha_{i1}\alpha_{i2}(1 - \alpha_{i2}) = \alpha_{r1}\alpha_{r2}(1 - \alpha_{r2}) = (1 - \alpha_{r2})\beta$.

The difference in the probability that the environment returns 1 between the two worlds is $\alpha_r - \alpha_i = (1 - (1 - \alpha_{r2}))\beta = \alpha_{r2}\beta \geq \beta^2$. Since β is non-negligible, so is the difference. Therefore $\mathcal{Z}$ can distinguish the ideal and the real worlds with non-negligible probability. $\square$

(Proof of Theorem 4.13). Given that there exists a witness extractor E such that no A can "explain" the same commitment value in two ways (as in definition for binding in the sense of extractability). We need to show that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. We use E as a part of $\mathcal{S}$, i.e., we let $\mathcal{S} = \mathcal{S}^{\Sigma,\mathsf{E}}_{\text{bind}}$ as above. Each time, when $\mathcal{S}$ obtains $(\textsc{LeakCommit}, \langle C, V, sid\rangle, m)$ from $\mathcal{F}_{\text{bind}}$, it simulates party C inside to commit on such m to party V; when the simulated V receives the commitment value from party C, the ideal world adversary $\mathcal{S}$ sends symbol $(\textsc{InflCommit}, \langle C, V, sid\rangle)$ to $\mathcal{F}_{\text{bind}}$; later when $\mathcal{S}$ obtain the $\textsc{LeakOpen}$ symbol from the functionality, it simulates party C to open the commitment value to the m to party V; when the simulated V receives the opening value from party C, party V produces the verification result $\langle m, \phi\rangle$, and the adversary $\mathcal{S}$ sends $(\textsc{InflOpen}, \langle C, V, sid\rangle, \langle m, \phi\rangle)$ to $\mathcal{F}_{\text{bind}}$. Further $\mathcal{S}$ corrupts the parties following $\mathcal{Z}$'s instruction. Whenever $\mathcal{S}$ receives a commitment value c from a corrupted committer C,

$\mathcal{S}$ runs E to extract the underlying m and play the role of the committer C to send $(\textsc{Commit}, \langle C, V, sid\rangle, m)$ to $\mathcal{F}_{\text{bind}}$, and receives the corresponding $\textsc{LeakCommit}$ symbol from the functionality; later when $\mathcal{S}$ receives from the corrupted committer C the opening for the c, $\mathcal{S}$ simulates party V with such opening value, and if V returns $\langle m', \phi\rangle$, then $\mathcal{S}$ sends $(\textsc{InflOpen}, \langle C, V, sid\rangle, \langle m', \phi\rangle)$ to $\mathcal{F}_{\text{bind}}$.

Next we justify the validity of $\mathcal{S}$ based on the validity of E. Assume $\pi_{\Sigma(\texttt{COM})}$ cannot realize $\mathcal{F}_{\text{bind}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ which can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$ inside; and A further simulates the real world for the copy of $\mathcal{Z}$. The attacker A sets $\langle c, m', \xi'\rangle$ outputted by $\mathcal{Z}$ as its output.

We let F denote the event that $\mathcal{Z}$ outputs c and $\langle m', \xi'\rangle$, where c can be extracted into m and $m \neq m'$ and $\langle m', \xi'\rangle$ is a valid opening. Observe that if event F does not occur and $\mathcal{F}_{\text{bind}}$ does not halt, the simulated $\mathcal{Z}$ cannot distinguish the two worlds; recall that $\mathcal{F}_{\text{bind}}$ halt only when $m' \neq m$ and $\phi = 1$; notice that given the extractor E, $m' \neq m$ holds with only negligible probability, which means $\mathcal{F}_{\text{bind}}$ halts with negligible probability. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful binding attacker. $\square$

C Simulation-Based Games

We define a *simulation-based game* similarly to an indistinguishability game in the sense that the adversary will again be attacking a *particular action*. This can be extended to a set of actions but we do not consider such games at present. The simulation-based game has the distinguishing characteristic that the challenger has access to a PPT TM called the simulation machine S that produces outputs that appear to be close to the attacked action output while using only a portion of the input that is given to the action. In the simulation-based game, at some point of the interaction, the attacker will provide some input x to the challenger. In response, the challenger C will flip a coin b, and if $b = 0$ it will execute the action on the whole input x and return the output to A, while, if $b = 1$ it will run the simulation machine of the action with only a portion of x and return the output to A. The game will terminate with the attacker A providing a guess b^* for the bit b. The judge is defined in the same way as in the indistinguishability game. We say that a cryptographic scheme that implements a task T satisfies the property defined by the simulation-based game G if there exists a simulation machine S for all PPT attackers A so that the function $|\mathsf{Succ}^G_{\mathsf{A}} - \frac{1}{2}|$ is a negligible function in λ.

We next define the translation of a simulation-based hiding game. As in the case of indistinguishability our methodology proceeds in two steps: we first define an environment and ideal-world simulator and then a corresponding canonical functionality.

Step 1: Defining the environment and simulator. Suppose S is a simulator for a particular action of the cryptographic task and A is an adversary playing the simulation game G. The environment $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_G$ proceeds as follows: it first flips a coin b, and if $b = 0$ it simulates G together with the adversary A and executes all actions requested by the challenger C. On the other hand, if $b = 1$, $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_G$ operates in the same way but it substitutes the execution of the action under consideration with an execution of the simulator S. As before the environment outputs 1 if and only if the judge determines that the adversary wins the game G. $\mathcal{S}^{\Sigma,\mathsf{S}}_G$ on the other hand operates as follows: given the $\textsc{LeakAction}$ symbol for the action under consideration it simulates S internally and returns the output to the environment.

Step 2: Defining the canonical functionality. $\textsc{Action}_i$ operates on input x; assume that that $x = \langle x^L, x^R\rangle$ and the simulator of the action S action operates on input x^L. In such case, we define the $\mathsf{suppress}$ function for symbol $\mathsf{a} = (\textsc{Action}_i, \mathbf{P}, x)$ by $\mathsf{suppress}(\mathsf{a}) = \langle x^L, (-)^{|x^R|}\rangle$.

Claim C.1. *Suppose that a cryptographic scheme Σ implements a cryptographic task T and G is an simulation-based game for T. Then it holds that if π_Σ UC-realizes some $\mathcal{F} \gtrsim \mathcal{F}_G$ then Σ satisfies the hiding property defined by game G.*

We note that it is possible to define simulation-based hiding games where a different part of the action input x is supposed to be provided to the simulator depending on actions taken prior to the action that is being attacked. For example, if $x = \langle x^L, x^R \rangle$ it can be the case that the simulator is given x^L on some executions and x^R on others (for an example of such a formulation see [HK07] in the case of oblivious transfer). In such a setting we will define suppress to eliminate both the x^L, x^R parts of the action content for a symbol $(\textsc{Action}_i, \mathbf{P}, x)$.

D Applying the Methodology: Other Tasks

D.1 Commitments (cont'd)

D.1.1 Binding

We begin this section by showing an alternative, more standard definition of binding:

Definition D.1 (Binding). *A commitment scheme* $\Sigma(\texttt{COM}) = \langle \texttt{commit}, \texttt{verify} \rangle$ *is* binding *if for all PPT attackers* A *it holds that* $\Pr\left[\begin{array}{l}(c, m_1, \xi_1, m_2, \xi_2) \leftarrow \mathsf{A}(1^\lambda); \phi_1 \leftarrow \texttt{verify}(c, m_1, \xi_1);\\ \phi_2 \leftarrow \texttt{verify}(c, m_2, \xi_2) : \phi_1 = \phi_2 = 1 \wedge m_1 \neq m_2\end{array}\right] \leq \mathsf{negl}(\lambda)$.

The property of extractability given in Definition 4.9 is a strengthening of the binding formulation as given above.

Proposition D.2. *A commitment scheme* Σ *that satisfies binding in the sense of extractability (cf. Definition 4.9) satisfies the binding property of Definition D.1.*

Proof. Given a scheme Σ that is extractable, we need to show that such Σ is binding. By contradiction, assume that Σ is not binding, i.e., there is a binding attacker A_{bind} that wins the binding game with non-negligible probability β. We construct an extractability attacker A_{ext} that wins the extractability game also with non-negligible probability.

A_{ext} simulates A_{bind}; once A_{bind} outputs $\langle c, m_1, \xi_1, m_2, \xi_2 \rangle$, A_{ext} flips a coin b, and outputs a tuple $\langle c, m_b, \xi_b \rangle$. Since A_{bind} is a successful binding attacker, with non-negligible probability β, $\texttt{verify}(c, m_1, \xi_1) = 1$ and $\texttt{verify}(c, m_2, \xi_2) = 1$ and $m_1 \neq m_2$. For any E with input c, it outputs m' such that $m' = m_b$ with probability at most $\frac{1}{2}$. Therefore following the extractability game, for all E, there exists an extractability attacker with probability $\frac{\beta}{2}$, which is also non-negligible. □

D.1.2 Hiding and equivocality

In this section we formulate commitment hiding as a simulation-based game, apply our translation methodology to it and investigate the property's relationship with the commitment property of equivocality.

Definition D.3 (Hiding). *A commitment scheme* $\Sigma(\texttt{COM}) = \langle \texttt{commit}, \texttt{verify} \rangle$ *is* hiding *if there exists a PPT* S *so that for all PPT attackers* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*, it holds*

$$\Pr\left[\begin{array}{l}(m, st) \leftarrow \mathsf{A}_1(1^\lambda); b \xleftarrow{r} \{0,1\};\\ \textbf{if } b = 0 \textbf{ then } (c, \xi) \leftarrow \texttt{commit}(m) \textbf{ else } c \leftarrow \mathsf{S}(1^\lambda);\\ b^* \leftarrow \mathsf{A}_2(st, c) \;:\; b = b^*\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The above definition can be modeled as a hiding game G_{hide} for the task $\texttt{COM}$ as follows. The challenger C is allowed to use algorithms $\texttt{commit}()$, $\texttt{verify}()$ as oracles, and interacts with the attacker $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. First A_1 produces a value m. In response, the challenger C randomly chooses a bit b; if $b = 0$, then C queries the $\texttt{commit}()$ oracle with m to obtain $\langle c, \xi \rangle$. If $b = 1$, then C feeds S with 1^λ to obtain c. Later C sends such c to A_2 to obtain b^* as a guess of b. The judge J decides that A wins the game if $b^* = b$. We next proceed to apply the methodology of Section C.

Step 1. We construct an environment $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\text{hide}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma,\mathsf{S}}_{\text{hide}}$ based on the game G_{hide} described above. In order to simulate the game, the environment first picks C, V from the namespace at random as well as a random sid. Then it requests the corruption of the party V and simulates A_1 on input 1^λ. Once A_1 produces m, $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\text{hide}}$ sends C the symbol $(\text{COMMIT}, \langle C, V, sid\rangle, m)$. The environment now expects c from the adversary. Outside the environment, a coin b is flipped; if $b = 0$, the environment will interact with the real world, party C will produce c based on its input $(\text{COMMIT}, \langle C, V, sid\rangle, m)$. If $b = 1$, the environment will interact with the ideal world, now $\mathcal{S}^{\Sigma}_{\text{hide}}$ will produce such c based on the information it receives from the ideal functionality. Finally the environment simulates A_2 on input c; when A_2 outputs b^*, the environment outputs such b^*.

Step 2. Based on the environment $\mathcal{Z}^{\mathsf{A}}_{\text{hide}}$ we define the functionality class that corresponds to the hiding game:

Definition D.4 (Canonical Functionality $\mathcal{F}_{\text{hide}}$)**.** *The functionality* $\mathcal{F}_{\text{hide}} \in \mathscr{F}_{\texttt{COM}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}$, *where (1)* $\mathsf{validate}() = 1$ *always, and (2)* $\mathsf{suppress}(\mathsf{a}) = (-)^{|m|}$ *for* $\mathsf{a} = (\text{COMMIT}, \langle C, V, sid\rangle, m)$.

Note that the functionality $\mathcal{F}_{\text{hide}}$ above is the same as the one in Definition 4.7 that was produced by the indistinguishability-based formulation of hiding. Interestingly, $\mathcal{F}_{\text{hide}}$ captures a stronger notion of hiding, namely, the equivocality property. Next we give the definition of equivocality, and then we show the equivalence between the security notions of $\mathcal{F}_{\text{hide}}$ and equivocality.

Definition D.5 (Equivocality)**.** *A commitment scheme* $\Sigma(\texttt{COM}) = \langle\texttt{commit}, \texttt{verify}\rangle$ *is* equivocal *if there exists a PPT* $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ *for all PPT attackers* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$, *so that*

$$\Pr\left[\begin{array}{l} (m, st) \leftarrow \mathsf{A}_1(1^\lambda); b \xleftarrow{r} \{0,1\}; \\ \textbf{if } b = 0 \textbf{ then } (c, \xi) \leftarrow \texttt{commit}(m) \\ \;\textbf{else if } b = 1 \textbf{ then } (c, \delta) \leftarrow \mathsf{S}_1(1^\lambda); \\ st \leftarrow \mathsf{A}_2(c); \\ \textbf{if } b = 1 \textbf{ then } \xi \leftarrow \mathsf{S}_2(c, \delta, m); \\ b^* \leftarrow \mathsf{A}_3(st, \xi) \;:\; b = b^* \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Theorem D.6. *If* $\pi_{\Sigma(\texttt{COM})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\text{hide}}$ *against static adversaries, then* $\Sigma(\texttt{COM})$ *is equivocal.*

Proof. Assume that there exists an ideal world adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish the executions in the real world and executions with the ideal process $\mathcal{F}_{\text{hide}}$. We will show that there exists a simulator S such that no A can win the simulation-based game for equivocality. S operates as follows: on input 1^λ it runs $\mathcal{S}$ on input $(\text{LEAKCOMMIT}, \mathbf{P})$ and when $\mathcal{S}$ responds with c it sets δ to be the internal state of $\mathcal{S}$ and returns (c, δ). S on input (c, δ, m) it invokes S with state δ on input $\langle(\text{LEAKOPEN}, \mathbf{P}), m\rangle$; when S outputs the message (m, ξ) as the opening from the committer to the verifier, S terminates and returns ξ. Next we justify the validity of S based on the validity of $\mathcal{S}$.

By contradiction, we assume $\Sigma(\texttt{COM})$ is not equivocal, i.e., for all S, there exists a successful equivocality attacker A that outputs $b^* = b$ with non-negligible probability at least $\beta + 1/2$ where β is a non-negligible function. We construct an environment $\mathcal{Z}$ based on A that distinguishes the two worlds with non-negligible probability; we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\text{hide}}$ as defined above.

If $b = 0$ and $\mathcal{Z}$ interacts with the real world, the pair $\langle c, \xi\rangle$ will be produced honestly by the party P following the $\texttt{commit}()$ protocol; recall that A succeeds with advantage β in predicting b. On the other hand, if $b = 0$ and $\mathcal{Z}$ interacts with the ideal world, the values $\langle c, \xi\rangle$ will be produced by $\mathcal{S}$ as defined above (based on $c \leftarrow \mathsf{S}_1(1^\lambda)$ and $\xi \leftarrow \mathsf{S}_2(c, m)$, respectively). Therefore, $\mathcal{Z}$, conditioning on $b = 0$ it distinguishes the two worlds with non-negligible probability β, a contradiction. □

Theorem D.7. *If* $\Sigma(\texttt{COM})$ *is equivocal, then* $\pi_{\Sigma(\texttt{COM})}$ *realizes* $\mathcal{F}_{\text{hide}}$ *against static adversaries.*

Given that Σ is equivocal there is a simulator S according to the equivocality definition. To show that Σ UC realizes $\mathcal{F}_{\text{hide}}$, we will construct a simulator $\mathcal{S}$ by employing S for the simulation of the interactions between the committer and receiver. The proof of the theorem follows easily.

D.1.3 Multi-session commitments

We now show how to extend the results of Section 4 for the commitment primitive in the multi-session setting where a single functionality serves a population of committers and verifiers.

In the single-session setting $\mathbf{P}$ determines the id's of the two parties involved in the session as well as their roles. In the multi-session setting $\mathbf{P}$ determines the id's of all active parties within the session as well as the id's of the two parties involved in a particular subsession and their roles. The translation methodology given in Section 4 generalizes directly resulting in the multisession commitment functionality $\hat{\mathcal{F}}_{\text{comp}} \wedge \hat{\mathcal{F}}_{\text{bind}} \wedge \hat{\mathcal{F}}_{\text{hide}}$. Moreover, we are able to investigate the relationship of these properties with the non-malleability property that is defined for the multi-session setting. We use the definition of non-malleability that was given in [PR05].

Definition D.8 (Concurrent non-malleability w.r.t. opening [PR05])**.** *A commitment scheme* $\Sigma(\mathtt{COM}) = \langle \mathtt{commit}, \mathtt{verify}\rangle$ *is* concurrently non-malleable with respect to opening *if for all PPT attackers* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$ *there exists a PPT* $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$*, so that for all PPT* $\mathsf{D} = (\mathsf{D}_1, \mathsf{D}_2)$

$$\Pr\left[\begin{array}{l} (\{m_i\}_{i\in[n]}, \gamma) \leftarrow \mathsf{D}_1(1^\lambda); b \xleftarrow{r} \{0,1\}; st := 1^\lambda; \delta := 1^\lambda; \\ \textbf{for } i \in [n] \\ \quad \textbf{if } b = 0 \\ \quad \textbf{then} \\ \qquad (c_i, \xi_i) \leftarrow \mathtt{commit}(m_i); (c_i', st) \leftarrow \mathsf{A}_1(st, c_i); (m_i', \xi_i') \leftarrow \mathsf{A}_2(st, m_i, \xi_i); \\ \quad \textbf{else} \\ \qquad (c_i', \delta) \leftarrow \mathsf{S}_1(\delta); (m_i', \xi_i') \leftarrow \mathsf{S}_2(\delta, m_i); \\ b^* \leftarrow \mathsf{D}_2(\gamma, \{m_i'\}_{i\in[n]}) \ : \ b = b^* \wedge_{i\in[n]} \mathtt{verify}(c_i', m_i', \xi_i') = 1 \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Theorem D.9. *If* π_Σ *realizes* $\hat{\mathcal{F}}_{\text{hide}}$*, then* Σ *satisfies concurrent non-malleability as defined above.*

D.2 Digital Signatures

The basic requirements for digital signatures were first formulated in [GMR88] including completeness, consistency[5], and unforgeability. We can model each property by a consistency game. In this section we show how to translate these traditional notions into the corresponding UC canonical functionalities for the digital signature task.

Following Figure 1, any canonical signature functionality $\mathcal{F}_{\mathtt{SIG}}$ is defined for two types of roles, the signer S and the verifiers V, with three actions, KEYGEN, SIGN, VERIFY. We denote the canonical signature functionality class as $\mathscr{F}_{\mathtt{SIG}}$. Next we treat the three security properties, unforgeability, completeness, and consistency, of signature one by one.

D.2.1 Unforgeability

Definition D.10 (Unforgeability)**.** *A signature scheme* $\Sigma(\mathtt{SIG}) = \langle \mathtt{gen}, \mathtt{sign}, \mathtt{verify}\rangle$ *is* unforgeable *if for all PPT forgers* A*,*

$$\Pr[(vk, sk) \leftarrow \mathtt{gen}(1^\lambda); (m, \sigma) \leftarrow \mathsf{A}^{\mathtt{sign}(vk,sk,\cdot)}(vk); \phi \leftarrow \mathtt{verify}(vk, m, \sigma) \ : \\ \phi = 1 \textit{ and } \mathsf{A} \textit{ never asked } \mathtt{sign}(vk, sk, \cdot) \textit{ to sign } m] \leq \mathsf{negl}(\lambda).$$

The above definition is a reformulation of GMR notion for unforgeability, which is based on a consistency game G_{uf} for task $\mathtt{SIG}$. The challenger C uses algorithms $\mathtt{gen}(), \mathtt{sign}(), \mathtt{verify}()$ as oracles, and interacts with forger A: C queries the $\mathtt{gen}()$ oracle and obtains $\langle sk, vk\rangle$, and then sends such vk to A; each time upon receiving m from the forger A, the challenger C queries the $\mathtt{sign}()$ oracle with m and obtains σ, and then returns such σ to A; upon receiving from A a pair $\langle m', \sigma'\rangle$, C queries the $\mathtt{verify}()$ oracle with $\langle m', \sigma', vk\rangle$ and

[5] Consistency is implied in the GMR specification, as pointed out by Canetti [Can04].

obtains the verification result. The judge J decides that A wins the game if m' has never been queried before and the verification result is 1.

Step 1. Based on the game G_{uf} described above, we can construct an environment $\mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}$ and the corresponding ideal world adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$ as follows. In order to simulate the game, the environment first picks S and V from the namespace at random as well as a random sid. The environment sends $(\textsc{KeyGen}, \langle S, sid\rangle)$ to party S and receives $(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)$ from it; then the environment simulates A internally by feeding vk as the input; when A queries m to its signing oracle, the environment sends $(\textsc{Sign}, \langle S, sid\rangle, m)$ to party S and receives σ, and returns such σ to A as the signature for the queried m; once A outputs a pair $\langle m, \sigma\rangle$, the environment inputs $(\textsc{VerifyReturn}, \langle V, sid\rangle, \langle m, \sigma, vk\rangle)$ to V and receives the verification result. In the case that m has never been queried and the verification result is 1, the environment terminates with 1; otherwise with 0.

We next define the ideal-world adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$. Each time $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$ receives $(\textsc{LeakKeyGen}, \langle S, sid\rangle)$ from the dummy functionality, it runs $(vk, sk) \leftarrow \mathtt{gen}(1^{\lambda})$ and sends $(\textsc{InflKeyGen}, \langle S, sid\rangle, vk)$ to the functionality. Later when $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$ receives $(\textsc{LeakSign}, \langle S, sid\rangle, m)$ from the dummy functionality, it runs $\sigma \leftarrow \mathtt{sign}(vk, sk, m)$, and sends $(\textsc{InflSign}, \langle S, sid\rangle, \sigma)$ to the functionality. When $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$ receives $(\textsc{LeakVerify}, \langle V, sid\rangle, \langle m, \sigma, vk\rangle)$ from the dummy functionality, it runs $\phi \leftarrow \mathtt{verify}(vk, sk, m, \sigma)$, and sends $(\textsc{InflVerify}, \langle V, sid\rangle, \phi)$ to the functionality.

Based on the dummy signature functionality $\mathcal{F}_{\mathtt{SIG}}^{\mathrm{dum}}$, the environment $\mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}$, and the adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$, we define the set of strings in the I/O communication of the functionality to the environment as specified in every possible run of the ideal world execution. Based on this we will obtain the bad language corresponding to the functionality in step 2.

Step 2. For any forger A and scheme Σ, the environment $\mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}$, the adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$, and the dummy canonical signature functionality together give rise to the language $L^{\mathrm{I/O}}_{\mathcal{F}_{\mathtt{SIG}}^{\mathrm{dum}}, \mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}, \mathcal{S}_{\mathrm{uf}}^{\Sigma}}$ (cf. Section 2). We consider the subset of strings $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}}$ of the union of all the I/O languages quantified over all possible forgers A and schemes Σ that contains exactly those strings for which the environment returns 1. Formally,

$$B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}} \overset{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma} L^{\mathrm{I/O}}_{\mathcal{F}_{\mathtt{SIG}}^{\mathrm{dum}}, \mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}, \mathcal{S}_{\mathrm{uf}}^{\Sigma}}$$

We next prove the following characterization of this language as well as determine its time-complexity:

Lemma D.11. *(1)* $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk) \\ \qquad (\textsc{Sign}, \langle S, sid\rangle, m_1)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma_1)\cdots \\ \qquad (\textsc{Sign}, \langle S, sid\rangle, m_k)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma_\ell) \\ \qquad (\textsc{Verify}, \langle V, sid\rangle, \langle m', \sigma', vk\rangle)(\textsc{VerifyReturn}, \langle V, sid\rangle, 1) \\ \textit{such that } m' \notin \{m_1, \ldots, m_\ell\} \end{array} \right\}$,

and (2) $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}}$ *is decidable in polynomial time.*

Proof. (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}} \subseteq Y$. Let w be any string in $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}}$; then it holds that there exist A, Σ such that w equals the history string in the ideal world execution of the environment $\mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}$ with adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$ and the dummy functionality $\mathcal{F}_{\mathtt{SIG}}^{\mathrm{dum}}$. Based on the definition of the environment $\mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}$ and the adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$, we know that the symbols $(\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)(\textsc{Sign}, \langle S, sid\rangle, m_1)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma_1)\cdots$ $(\textsc{Sign}, \langle S, sid\rangle, m_k)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma_\ell)(\textsc{Verify}, \langle V, sid\rangle, \langle m', \sigma', vk\rangle)(\textsc{VerifyReturn}, \langle V, sid\rangle, 1)$ will be recorded into history in the dummy functionality. It follows that the string w belongs to the set Y.

Second we need to show $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}} \supseteq Y$. Let w be any string in Y. We will construct A, Σ such that in the ideal world execution of $\mathcal{Z}_{\mathrm{uf}}^{\mathsf{A}}$ with adversary $\mathcal{S}_{\mathrm{uf}}^{\Sigma}$ and the dummy functionality $\mathcal{F}_{\mathtt{SIG}}^{\mathrm{dum}}$ it holds that history $= w$. Given that $w \in Y$, there exist string $w = (\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)(\textsc{Sign}, \langle S, sid\rangle, m_1)$ $(\textsc{SignReturn}, \langle S, sid\rangle, \sigma_1)\cdots(\textsc{Sign}, \langle S, sid\rangle, m_k)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma_\ell)(\textsc{Verify}, \langle V, sid\rangle, \langle m', \sigma', vk\rangle)$

($\textsc{VerifyReturn}, \langle V, sid\rangle, 1$). Define $\mathtt{gen}$ output $\langle vk, sk\rangle$. Define $\mathtt{sign}$ that upon input m_i returns σ_i for $1 \le i \le \ell$; Define A output $\langle m', \sigma'\rangle$; Define $\mathtt{verify}$ that upon input $\langle m', \sigma', vk\rangle$ returns 1. It follows immediately that the history string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\mathrm{uf}}$ with adversary $\mathcal{S}^{\Sigma}_{\mathrm{uf}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}$ would equal w.

(2) It is easy to show the language $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}}$ is decidable. □

In order to obtain the bad language for the unforgeability property we extend $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}}$ as follows:

$$B^{\mathrm{ext}}_{\mathtt{SIG},\mathrm{uf}} = \left\{ w \in L^{\mathrm{I/O}}_{\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}} \;\middle|\; \exists w' \in B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{uf}} \text{ s.t. } w' \preccurlyeq w \right\}$$

We observe that $B^{\mathrm{ext}}_{\mathtt{SIG},\mathrm{uf}}$ is also decidable in polynomial time.

Step 3. Next we define the class of ideal functionalities that corresponds to the unforgeability property.

Definition D.12 (Canonical Functionality $\mathcal{F}_{\mathrm{uf}}$)**.** *The functionality $\mathcal{F}_{\mathrm{uf}} \in \mathscr{F}_{\mathtt{SIG}}$ equals $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\mathtt{SIG}}$, where (1)* $\mathsf{suppress}()$ *is same as in $\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}$, and (2)* $\mathsf{validate}(w) = 0$ *if and only if $w \in B^{\mathrm{ext}}_{\mathtt{SIG},\mathrm{uf}}$.*

Theorem D.13. *If $\pi_{\Sigma(\mathtt{SIG})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{uf}}$, then $\Sigma(\mathtt{SIG})$ is unforgeable.*

Proof. By contradiction, assume scheme $\Sigma(\mathtt{SIG})$ is not unforgeable. We need to construct an environment $\mathcal{Z}$ to distinguish the two worlds with non-negligible probability. Based on the successful forger A, we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A}}_{\mathrm{uf}}$ as defined above. Notice that in the real world, A is a successful forger against $\Sigma(\mathtt{SIG})$, so $\mathcal{Z}$ outputs 1 with non-negligible probability. However in the ideal world, the case that a valid pair $\langle m', \sigma'\rangle$ where m' is new would cause any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{uf}}$ to halt, so the environment $\mathcal{Z}$ can never output 1. Therefore the constructed $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

Theorem D.14. *If $\Sigma(\mathtt{SIG})$ is unforgeable, then $\pi_{\Sigma(\mathtt{SIG})}$ realizes $\mathcal{F}_{\mathrm{uf}}$.*

Proof. Given that no forger A can win the unforgeability game above, we need to show that there exists a adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. The adversary $\mathcal{S}$ is designed as the generic adversary for signature task.

Assume $\pi_{\Sigma(\mathtt{SIG})}$ cannot realize $\mathcal{F}_{\mathrm{uf}}$, i.e. for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$ inside; and A further simulates the real world for the copy of $\mathcal{Z}$.

We let F denote the event that in a run of $\pi_{\Sigma(\mathtt{SIG})}$ with $\mathcal{Z}$, signer is honest, verification key vk is produced by the signer, m is not signed by the signer, and $\langle vk, m, \sigma\rangle$ is valid. Observe that if event F does not occur, the simulated $\mathcal{Z}$ cannot distinguish the two worlds. However, based on assumption above, $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful forger. □

D.2.2 Completeness

Definition D.15 (Completeness)**.** *A signature scheme $\Sigma(\mathtt{SIG}) = \langle \mathtt{gen}, \mathtt{sign}, \mathtt{verify}\rangle$ is* complete *if for all PPT attackers* A*,*

$$\Pr[m \leftarrow \mathsf{A}(1^\lambda); (vk, sk) \leftarrow \mathtt{gen}(1^\lambda); \sigma \leftarrow \mathtt{sign}(vk, sk, m); \phi \leftarrow \mathtt{verify}(vk, m, \sigma) \;:\; \phi = 0] \le \mathsf{negl}(\lambda).$$

The above definition can be modeled as a consistency game, G_{comp} as follows. The challenger C uses algorithms $\mathtt{gen}(), \mathtt{sign}(), \mathtt{verify}()$ as oracles, and interacts with completeness attacker A: after receiving m produced by A, the challenger C queries the $\mathtt{gen}()$ oracle and obtains sk, vk; then C queries the $\mathtt{sign}()$ oracle with sk, m and obtains σ; later C queries the $\mathtt{verify}()$ oracle with $\langle m, \sigma, vk\rangle$ to obtains the verification result. The judge J decides that A wins the game if the verification result is 0.

Step 1. Based on the game G_{comp} described above, we can construct an environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$. The environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ here is similar to the environment $\mathcal{Z}^{\mathsf{A}}_{\text{uf}}$; the environment first picks S and V from the namespace at random as well as a random sid. The environment simulates A with input 1^{λ} and obtains m; it then sends $(\textsc{KeyGen}, \langle S, sid\rangle)$ to party S and receives $(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)$ from the party S; later the environment sends $(\textsc{Sign}, \langle S, sid\rangle, m)$ to party S and receives σ; the environment inputs $(\textsc{VerifyReturn}, \langle V, sid\rangle, \langle m, \sigma, vk\rangle)$ to V and receives the verification result. If the verification result $\phi = 0$, the environment terminates with 1; otherwise with 0. The adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ is defined similarly to the adversary $\mathcal{S}^{\Sigma}_{\text{uf}}$ in the previous section.

Step 2. For any completeness attacker A and scheme Σ, the environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$, the adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$, and the dummy canonical signature functionality together give rise to the language $L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\texttt{SIG}}, \mathcal{Z}^{\mathsf{A}}_{\text{comp}}, \mathcal{S}^{\Sigma}_{\text{comp}}}$. We consider the subset of strings $B^{\text{I/O}}_{\texttt{SIG},\text{comp}}$ of the union of all the I/O languages quantified over all possible completeness attackers A and schemes Σ that contains exactly those strings for which the environment returns 1. Formally,

$$B^{\text{I/O}}_{\texttt{SIG},\text{comp}} \overset{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma} L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\texttt{SIG}}, \mathcal{Z}^{\mathsf{A}}_{\text{comp}}, \mathcal{S}^{\Sigma}_{\text{comp}}}$$

We next prove the following characterization of this language as well as determine its time-complexity:

Lemma D.16. *(1)* $B^{\text{I/O}}_{\texttt{SIG},\text{comp}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk) \\ \quad (\textsc{Sign}, \langle S, sid\rangle, m)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma) \\ \quad (\textsc{Verify}, \langle V, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V, sid\rangle, 0) \end{array} \right\}$

(2) $B^{\text{I/O}}_{\texttt{SIG},\text{comp}}$ *is decidable in polynomial time.*

Proof. (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\text{I/O}}_{\texttt{SIG},\text{comp}} \subseteq Y$. Let w be any string in $B^{\text{I/O}}_{\texttt{SIG},\text{comp}}$; then it holds that there exist A, Σ such that w equals the history string in the ideal world execution of the environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\texttt{SIG}}$. Based on the definition of the environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ and the adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$, we know that the symbols $(\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)(\textsc{Sign}, \langle S, sid\rangle, m)(\textsc{SignReturn}, \langle S, sid\rangle, \sigma)$ $(\textsc{Verify}, \langle V, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V, sid\rangle, 0)$ will be recorded into history in the dummy functionality. It follows that the string w belongs to the set Y.

Second we need to show $B^{\text{I/O}}_{\texttt{SIG},\text{comp}} \supseteq Y$. Let w be any string in Y. We will construct A, Σ such that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\texttt{SIG}}$ it holds that history $=$ w. Given that $w \in Y$, there exist $w = (\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)(\textsc{Sign}, \langle S, sid\rangle, m)$ $(\textsc{SignReturn}, \langle S, sid\rangle, \sigma)(\textsc{Verify}, \langle V, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V, sid\rangle, 0)$. Define $\texttt{gen}$ output $\langle vk, sk\rangle$. Define $\texttt{sign}$ that upon input m returns σ; Define A output $\langle m, \sigma\rangle$; Define $\texttt{verify}$ that upon input $\langle m, \sigma\rangle$ returns 0. It follows immediately that the history string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\texttt{SIG}}$ would equal w.

(2) It is easy to show the language $B^{\text{I/O}}_{\texttt{SIG},\text{comp}}$ is decidable. $\square$

In order to obtain the bad language for the completeness property we extend $B^{\text{I/O}}_{\texttt{SIG},\text{comp}}$ as follows:

$$B^{\text{ext}}_{\texttt{SIG},\text{comp}} = \left\{ w \in L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\texttt{SIG}}} \;\middle|\; \exists w' \in B^{\text{I/O}}_{\texttt{SIG},\text{comp}} \text{ s.t. } w' \preccurlyeq w \right\}$$

We observe that $B^{\text{ext}}_{\texttt{SIG},\text{comp}}$ is also decidable in polynomial time.

Step 3. We then define the class of ideal functionalities that corresponds to the completeness property.

Definition D.17 (Canonical Functionality $\mathcal{F}_{\text{comp}}$)**.** *The functionality* $\mathcal{F}_{\text{comp}} \in \mathscr{F}_{\texttt{SIG}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\texttt{SIG}}$, *where (1)* $\mathsf{suppress}()$ *is the same as in* $\mathcal{F}^{\text{dum}}_{\texttt{SIG}}$, *and (2)* $\mathsf{validate}(w) = 0$ *if and only if* $w \in B^{\text{ext}}_{\texttt{SIG},\text{comp}}$.

Theorem D.18. *If $\pi_{\Sigma(\texttt{SIG})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\text{comp}}$, then $\Sigma(\texttt{SIG})$ is complete.*

Proof. By contradiction, assume scheme $\Sigma(\texttt{SIG})$ is not complete. We need to construct an environment $\mathcal{Z}$ to distinguish the two worlds with non-negligible probability. Based on the successful completeness attacker A, we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ as defined above. Notice that in the real world, A is a successful completeness attacker against $\Sigma(\texttt{SIG})$, so $\mathcal{Z}$ outputs 1 with non-negligible probability. However in the ideal world, the case that a pair $\langle m, \sigma \rangle$ produced in the signing stage is verified with $\phi = 0$ would cause any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\text{comp}}$ to halt, so the environment $\mathcal{Z}$ can never output 1. Therefore the constructed $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

Theorem D.19. *If $\Sigma(\texttt{SIG})$ is complete, then $\pi_{\Sigma(\texttt{SIG})}$ realizes $\mathcal{F}_{\text{comp}}$.*

Proof. Given that no attacker A can win the completeness game above, we need to show that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. The adversary $\mathcal{S}$ is designed as the generic adversary for the signature task (that performs a simulation of the real-world).

Assume $\pi_{\Sigma(\texttt{SIG})}$ cannot realize $\mathcal{F}_{\text{comp}}$, i.e. for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$ inside; and A further simulates the real world for the copy of $\mathcal{Z}$. The adversary A will output the plaintext m that corresponds to the following event F:

F is defined as the event that in a run of $\pi_{\Sigma(\texttt{SIG})}$ with $\mathcal{Z}$, an honest signer, the verification key vk is produced by the signer, m is signed by the signer into σ based on the vk, and $\langle vk, m, \sigma \rangle$ verifies to 0. Observe that if the event F does not occur, the simulated $\mathcal{Z}$ cannot distinguish the two worlds. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means that the event F must occur with non-negligible probability, i.e., A is a successful completeness attacker. □

D.2.3 Consistency

Definition D.20 (Consistency). *A signature scheme $\Sigma(\texttt{SIG}) = \langle \texttt{gen}, \texttt{sign}, \texttt{verify} \rangle$ is* consistent *if for all PPT attackers A,*

$$\Pr[(vk, m, \sigma) \leftarrow \mathsf{A}(1^\lambda); \phi_1 \leftarrow \texttt{verify}(vk, m, \sigma); \phi_2 \leftarrow \texttt{verify}(vk, m, \sigma) \; : \; \phi_1 \neq \phi_2] \leq \mathsf{negl}(\lambda).$$

The above definition can also be modeled by a consistency game, G_{cons}, as follows. The challenger C uses algorithms $\texttt{gen}(), \texttt{sign}(), \texttt{verify}()$ as oracles, and interacts with the consistency attacker A: C simulates A on input 1^λ to obtain $\langle vk, m, \sigma \rangle$ and then calls the the $\texttt{verify}()$ oracle with $\langle m, \sigma, vk \rangle$ twice and obtains the verification results ϕ_1 and ϕ_2 respectively. The judge J decides that A wins the game if the two verification results are different, i.e., $\phi_1 \neq \phi_2$.

Step 1. Based on the game G_{cons} described above, we can construct an environment $\mathcal{Z}^{\mathsf{A}}_{\text{cons}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma}_{\text{cons}}$ as follows. The environment first picks S and two V's from the namespace at random as well as a random sid. Then the environment simulates A to obtain $\langle vk, m, \sigma \rangle$ and gives the symbols $(\textsc{Verify}, \langle V_1, sid \rangle, \langle m, \sigma, vk \rangle)$ and $(\textsc{Verify}, \langle V_2, sid \rangle, \langle m, \sigma, vk \rangle)$ to obtain the symbols $(\textsc{VerifyReturn}, \langle V_1, sid \rangle, \phi_1)$ and $(\textsc{VerifyReturn}, \langle V_2, sid \rangle, \phi_2)$. In the case that $\phi_1 \neq \phi_2$, the environment terminates with 1 otherwise with 0. $\mathcal{S}^{\Sigma}_{\text{cons}}$ is defined similarly to $\mathcal{S}^{\Sigma}_{\text{uf}}$.

Step 2. For any consistency attacker A and scheme Σ, the environment $\mathcal{Z}^{\mathsf{A}}_{\text{cons}}$, the ideal adversary $\mathcal{S}^{\Sigma}_{\text{cons}}$, and the dummy canonical signature functionality together give rise to the language $L^{\mathrm{I/O}}_{\mathcal{F}^{\text{dum}}_{\texttt{SIG}}, \mathcal{Z}^{\mathsf{A}}_{\text{cons}}, \mathcal{S}^{\Sigma}_{\text{cons}}}$. We consider the subset of strings $B^{\mathrm{I/O}}_{\texttt{SIG},\text{cons}}$ of the union of all the $\mathrm{I/O}$ languages quantified over all possible consistency attackers A and schemes Σ that contains exactly those strings for which the environment returns 1. Formally,

$$B^{\mathrm{I/O}}_{\texttt{SIG},\text{cons}} \stackrel{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma} L^{\mathrm{I/O}}_{\mathcal{F}^{\text{dum}}_{\texttt{SIG}}, \mathcal{Z}^{\mathsf{A}}_{\text{cons}}, \mathcal{S}^{\Sigma}_{\text{cons}}}$$

We next prove the following characterization of this language as well as determine its time-complexity:

Lemma D.21. *(1)* $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{Verify}, \langle V_1, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_1, sid\rangle, \phi_1) \\ \quad (\textsc{Verify}, \langle V_2, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_2, sid\rangle, \phi_2) \\ \textit{such that } \phi_1 \neq \phi_2 \end{array} \right\}$, *and (2)* $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}}$ *is decidable in polynomial time.*

Proof. (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}} \subseteq Y$. Let w be any string in $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}}$; then it holds that there exist A, Σ such that w equals the $\mathsf{history}$ string in the ideal world execution of the environment $\mathcal{Z}^{\mathsf{A}}_{\mathrm{cons}}$ with adversary $\mathcal{S}^{\Sigma}_{\mathrm{cons}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}$. Based on the definition of the environment $\mathcal{Z}^{\mathsf{A}}_{\mathrm{cons}}$ and the adversary $\mathcal{S}^{\Sigma}_{\mathrm{cons}}$, we know that the symbols $(\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)(\textsc{Verify}, \langle V_1, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_1, sid\rangle, \phi_1)$ $(\textsc{Verify}, \langle V_2, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_2, sid\rangle, \phi_2)$ where $\phi_1 \neq \phi_2$ will be recorded into $\mathsf{history}$ in the dummy functionality. It follows that the string w belongs to the set Y.

Second we need to show $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}} \supseteq Y$. Let w be any string in Y. We will construct A, Σ such that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\mathrm{cons}}$ with adversary $\mathcal{S}^{\Sigma}_{\mathrm{cons}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}$ it holds that $\mathsf{history} = w$. Given that $w \in Y$, there exist string $w = (\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk)$ $(\textsc{Verify}, \langle V_1, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_1, sid\rangle, \phi_1)(\textsc{Verify}, \langle V_2, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn},$ $\langle V_2, sid\rangle, \phi_2)$ with $\phi_1 \neq \phi_2$. Define $\mathtt{gen}$ output $\langle vk, sk\rangle$. Define A output $\langle m, \sigma\rangle$; Define $\mathtt{verify}$ that upon input $\langle m, \sigma, vk\rangle$ returns ϕ_1 for the first time and ϕ_2 for the second time. It follows immediately that the $\mathsf{history}$ string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\mathrm{cons}}$ with adversary $\mathcal{S}^{\Sigma}_{\mathrm{cons}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}$ would equal w.

(2) It is easy to show the language $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}}$ is decidable. □

In order to obtain the bad language for the consistency property we extend $B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}}$ as follows:

$$B^{\mathrm{ext}}_{\mathtt{SIG},\mathrm{cons}} = \left\{ w \in L^{\mathrm{I/O}}_{\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}} \;\middle|\; \exists w' \in B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}} \text{ s.t. } w' \preccurlyeq w \right\}$$

We observe that $B^{\mathrm{ext}}_{\mathtt{SIG},\mathrm{cons}}$ is also decidable in polynomial time.

Step 3. We proceed next to define the canonical functionality that corresponds to the consistency property.

Definition D.22 (Canonical Functionality $\mathcal{F}_{\mathrm{cons}}$)**.** *The functionality* $\mathcal{F}_{\mathrm{cons}} \in \mathscr{F}_{\mathtt{SIG}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\mathtt{SIG}}$, *where (1)* $\mathsf{suppress}()$ *is same as in* $\mathcal{F}^{\mathrm{dum}}_{\mathtt{SIG}}$, *and (2)* $\mathsf{validate}(w) = 0$ *if and only if* $w \in B^{\mathrm{ext}}_{\mathtt{SIG},\mathrm{cons}}$.

Theorem D.23. *If* $\pi_{\Sigma(\mathtt{SIG})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{cons}}$, *then* $\Sigma(\mathtt{SIG})$ *is consistent.*

Proof. By contradiction, assume scheme $\Sigma(\mathtt{SIG})$ is not consistent. We need to construct an environment $\mathcal{Z}$ to distinguish the two worlds with non-negligible probability. Based on the successful consistency attacker A, we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A}}_{\mathrm{cons}}$ as defined above. Notice that in the real world, A is a successful consistency attacker against $\Sigma(\mathtt{SIG})$, so $\mathcal{Z}$ outputs 1 with non-negligible probability. However in the ideal world, the case that $\phi_1 \neq \phi_2$ would cause any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{cons}}$ to halt, so the environment $\mathcal{Z}$ can never output 1. Therefore the constructed $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

Theorem D.24. *If* $\Sigma(\mathtt{SIG})$ *is consistent, then* $\pi_{\Sigma(\mathtt{SIG})}$ *realizes* $\mathcal{F}_{\mathrm{cons}}$.

Proof. Given that no attacker A can win the consistency game above, we need to show that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. The adversary $\mathcal{S}$ is designed as the generic ideal world adversary (that performs a simulation of the real-world).

Assume $\pi_{\Sigma(\mathtt{SIG})}$ cannot realize $\mathcal{F}_{\mathrm{cons}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can distinguish the two worlds with non-negligible probability. A operates by simulating a copy of $\mathcal{Z}$ in the real world; it returns m, σ, vk based on an event F as defined below.

We let F denote the event that in a run of $\pi_{\Sigma(\mathtt{SIG})}$ with $\mathcal{Z}$, the same tuple $\langle vk, m, \sigma\rangle$ is verified with different results in two verifications. Observe that if event F does not occur, the simulated $\mathcal{Z}$ cannot distinguish the two worlds. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful consistency attacker. $\square$

In this section we used a different game-based formulation compared to the one in [Can04]. The reason is that the consistency formulation given there and reformulated below falls short of capturing the intended properties for the digital signature task in the UC setting. We illustrate these issues below.

Definition D.25 (Weak Consistency). *A signature scheme* $\Sigma(\mathtt{SIG}) = \langle \mathtt{gen}, \mathtt{sign}, \mathtt{verify} \rangle$ *is* weakly consistent *if for all PPT attackers* A,

$$\Pr\left[\begin{array}{c}(vk, sk) \leftarrow \mathtt{gen}(1^\lambda); (m, \sigma) \leftarrow \mathsf{A}^{\mathtt{sign}(vk,sk,\cdot)}(vk); \\ \phi_1 \leftarrow \mathtt{verify}(vk, m, \sigma); \phi_2 \leftarrow \mathtt{verify}(vk, m, \sigma)\ :\ \phi_1 \neq \phi_2\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark D.26. The weak consistency definition above is taken from [Can04]. We can construct a counterexample Σ' below which satisfies completeness, unforgeability and the weak consistency, but the corresponding $\pi_{\Sigma'}$ does not realize $\mathcal{F}_{\mathtt{SIG}}$ in [Can04] (or the $\mathcal{F}_{\mathtt{SIG}}$ that is produced from our translation methodology in this section).

Let Σ be a scheme satisfies completeness, unforgeability and weak consistency. We modify such Σ into Σ': (1) prepend a bit b to the verification key; if $b = 0$ then the verification procedure remains the same; if $b = 1$ then the verification procedure accepts its input message-signature pair with probability $1/2$; (2) the key generation algorithm returns a verification key starting with bit 0. Notice that Σ' still satisfies the three properties, completeness, unforgeability and weak consistency, since the honest key generation will never return a verification key starting with bit 1. According to Theorem 2 in [Can04], the corresponding $\pi_{\Sigma'}$ can realize $\mathcal{F}_{\mathtt{SIG}}$. However this is not true. When the signer is corrupted at the beginning, a verification key vk' with starting bit 1 can be chosen and then two verification requests with the same input $\langle m, \sigma, vk' \rangle$ will return in different verification results with non-negligible probability, i.e., $1/2$ in this case.

D.2.4 The canonical ideal signature functionality

The (canonical) ideal signature functionality would equal $\mathcal{F}_{\text{uf}} \wedge \mathcal{F}_{\text{comp}} \wedge \mathcal{F}_{\text{cons}}$. In light of Proposition 2.5 we obtain the following:

Corollary D.27. *If* $\pi_{\Sigma(\mathtt{SIG})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\text{uf}} \wedge \mathcal{F}_{\text{comp}} \wedge \mathcal{F}_{\text{cons}}$, *then the signature scheme* $\Sigma(\mathtt{SIG})$ *satisfies unforgeability, completeness, and consistency.*

Remark D.28. It is easy to verify that the canonical functionality $\mathcal{F}_{\text{uf}} \wedge \mathcal{F}_{\text{comp}} \wedge \mathcal{F}_{\text{cons}}$ is UC-equivalent to the digital signature ideal functionality of [Can04] and thus UC-realizable. Recall that a problematic variant of $\mathcal{F}_{\mathtt{SIG}}$ appeared first in [Can01]; a main shortcoming of that first rendering of $\mathcal{F}_{\mathtt{SIG}}$ was its failure to capture the consistency property something that was also pointed out in [BH04]; this latter work did not capture consistency fully either as pointed out in [Can04], which performed a thorough investigation between the correspondence of the game-based security formulation of the Goldwasser et al. [GMR88] notion for digital signatures and the $\mathcal{F}_{\mathtt{SIG}}$ ideal functionality. In [Can04] a correspondence theorem was shown that established that any digital signature scheme secure in the [GMR88]-sense would result in a UC-secure signature protocol.

However, as we now show with the help of our methodology this correspondence does not stand. In fact, when one applies our translation methodology to the three game-based definitions that are put forth in [Can04] to capture the [GMR88] notion of security, the resulting functionality is not the $\mathcal{F}_{\mathtt{SIG}}$ functionality as defined above. This is due to the fact that the consistency game as defined in [Can04] (cf. page 12, Definition 1) assumes an honest key generation. More specifically, if our consistency game translation is applied to the consistency game as given in that work it results in a bad language that is of the following form (cf. the bad language of Lemma D.21):

$$(B^{\mathrm{I/O}}_{\mathtt{SIG},\mathrm{cons}})' = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{KeyGen}, \langle S, sid\rangle)(\textsc{KeyGenReturn}, \langle S, sid\rangle, vk) \\ \quad (\textsc{Verify}, \langle V_1, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_1, sid\rangle, \phi_1) \\ \quad (\textsc{Verify}, \langle V_2, sid\rangle, \langle m, \sigma, vk\rangle)(\textsc{VerifyReturn}, \langle V_2, sid\rangle, \phi_2) \\ \text{such that } \phi_1 \neq \phi_2 \end{array} \right\}$$

It follows that the corresponding $\mathcal{F}'_{\mathrm{cons}}$ canonical functionality would have a validate predicate that checks for verification inconsistency *only in the case* that a KeyGen symbol has been recorded in the history of the functionality. This is too restrictive as it precludes corrupted signers that may never register a KeyGen symbol with the functionality (and in fact this is exactly the issue pointed out in [Can04] regarding the previous work of [BH04]). It is easy to see that the resulting (weaker) canonical functionality $\mathcal{F}'_{\mathtt{SIG}} = \mathcal{F}_{\mathrm{uf}} \wedge \mathcal{F}_{\mathrm{comp}} \wedge \mathcal{F}'_{\mathrm{cons}}$ resides at a lower point compared to $\mathcal{F}_{\mathtt{SIG}}$ in the $\mathscr{F}_{\mathtt{SIG}}$ lattice of canonical functionalities. Furthermore, it is possible to design a digital signature scheme Σ so that its corresponding π_Σ UC-realizes $\mathcal{F}'_{\mathtt{SIG}}$ but fails to realize $\mathcal{F}_{\mathtt{SIG}}$, as shown in Remark D.26. It is easy to see that this scheme passes the game based formulation of [Can04] and based on our methodology it will UC-realize $\mathcal{F}'_{\mathtt{SIG}}$. Nevertheless, $\mathcal{F}_{\mathtt{SIG}}$ will not be realized by this digital signature (the environment can produce two public-keys prepended with "0" and then issue two verification requests — this is sufficient to distinguish the real from the ideal world in the $\mathcal{F}_{\mathtt{SIG}}$ setting). As a result the correspondence between the game-based notions and the UC formulation given in [Can04] (cf. page 12, Theorem 2) is not correct. The appropriate formulation of the consistency game is the one we present in Definition D.20 and this provides the exact game-based correspondence to the $\mathcal{F}_{\mathtt{SIG}}$ canonical functionality.

D.3 Zero-Knowledge Proofs

The concept of zero-knowledge proofs was introduced by Goldwasser, Micali and Rackoff [GMR89]. Bellare and Goldreich [BG92] extended the soundness property to knowledge extraction. Here we focus on zero-knowledge proofs that are public-coin protocols for the verifier, i.e., the verifier has a public random tape.

Following Figure 1, the canonical functionality for zero-knowledge, $\mathcal{F}_{\mathsf{ZK}}$, is defined for two types of roles, the prover P and the verifier V, with a single action Prove. We denote the zero-knowledge proof functionality class as $\mathscr{F}_{\mathsf{ZK}}$. The WF predicate for $\mathcal{F}_{\mathsf{ZK}}$, requires that a Prove should precede ProveReturn. The default output DO returns $\langle x, \phi\rangle$ whenever $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, w\rangle)$ is in the history, where $\phi = 1$ if and only if $\langle x, m\rangle$ belongs to the relation that parameterizes the zero-knowledge task, and $\phi = 0$ otherwise. Based on the above the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathsf{ZK}}$ is defined (cf. Definition 2.1).

D.3.1 Completeness

For simplicity we provide the ZK properties for a single protocol (i.e., essentially the NIZK setting); nevertheless the results of this section can be extended to protocols with many moves in straightforward way.

Definition D.29 (Completeness)**.** *A zero-knowledge proof scheme* $\Sigma(\mathrm{ZK}) = \langle \mathtt{prove}, \mathtt{verify}\rangle$ *is* complete *if for all PPT attackers* A,

$$\Pr[(x, m) \leftarrow \mathsf{A}(1^\lambda); \varpi \leftarrow \mathtt{prove}(x, m); \phi \leftarrow \mathtt{verify}(x, \varpi) : (x, m) \in R \wedge \phi \neq 1] \leq \mathsf{negl}(\lambda).$$

The above property can be modeled by a consistency game, G_{comp}, as follows. The challenger C uses as oracles the algorithms $\mathtt{prove}(), \mathtt{verify}()$, and interacts with the attacker A: the attacker A outputs a pair $(x, m) \in R$; the challenger queries the proving oracle with the pair $\langle x, m\rangle$ and obtains the response ϖ; the challenger then queries the verification oracle with the pair $\langle x, \varpi\rangle$ and obtains the response ϕ. The judge J decides that A wins the game if $(x, m) \in R$ and $\phi \neq 1$. Next we describe the three steps of the transformation outlined in Section 3.1.

Step 1. We construct an environment $\mathcal{Z}^{\mathsf{A}}_{\mathrm{comp}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma}_{\mathrm{comp}}$ based on the game G_{comp} described above. In order to simulate the game, the environment $\mathcal{Z}^{\mathsf{A}}_{\mathrm{comp}}$ first picks P, V

from the namespace at random as well as a random sid. Then simulates A on input 1^λ; once A outputs $\langle x, m\rangle$, and if $(x, m) \in R$, then the environment sends $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ to party P; the environment then receives ϖ as the proof transcripts through the adversary. Finally if $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ receives a symbol $(\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, 0\rangle)$ from V, the environment terminates with 1; otherwise terminates with 0.

The ideal-world adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ translates the communications from $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ into the appropriate symbols that the canonical zero-knowledge functionality understands. When $\mathcal{S}^{\Sigma}_{\text{comp}}$ receives the pair $\langle x, \varpi\rangle$, if $\texttt{verify}(x, \varpi) = 1$, it sends a symbol $(\textsc{InflProve}, \langle P, V, sid\rangle, \langle x, 1\rangle)$; else if $\texttt{verify}(x, \varpi) = 0$, a symbol $(\textsc{InflProve}, \langle P, V, sid\rangle, \langle x, 0\rangle)$ to the functionality.

Step 2. For any completeness attacker A and scheme Σ, the environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$, the adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$, and the dummy canonical zero-knowledge functionality $\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}$ together give rise to the language $L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}, \mathcal{Z}^{\mathsf{A}}_{\text{comp}}, \mathcal{S}^{\Sigma}_{\text{comp}}}$ (cf. Section 2). We consider the subset of strings $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}}$ of the union of all the I/O languages quantified over all possible completeness attackers A and schemes Σ that contains exactly those strings corresponds to the case that the environment returns 1. Formally,

$$B^{\text{I/O}}_{\mathsf{ZK},\text{comp}} \overset{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma} L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}, \mathcal{Z}^{\mathsf{A}}_{\text{comp}}, \mathcal{S}^{\Sigma}_{\text{comp}}}$$

We next prove the following characterization of this language as well as determine its time-complexity:

Lemma D.30. *(1)* $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle) \\ \quad (\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, 0\rangle) \\ \textit{such that } (x, m) \in R \end{array} \right\}$*, and (2)* $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}}$ *is decidable in polynomial time.*

Proof. (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}} \subseteq Y$. Let w be any string in $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}}$; then it holds that there exist A, Σ such that w equals the history string in the ideal world execution of the environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}$. Based on the definition of the environment $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ and the adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$, we know that the symbol $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ from the dummy party P will be recorded by the adversary into history in the dummy functionality; and then after receiving an $\textsc{InflProve}$ symbol from the adversary, the symbol $(\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, 0\rangle)$ will be recorded in history and be sent to the dummy party V by the functionality. It follows that the string w belongs to the set Y.

Second we need to show $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}} \supseteq Y$. Let w be any string in Y. We will construct A, Σ such that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}$ it holds that history $= w$. Given that $w \in Y$ it holds that there exist x, m such that $w = (\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ $(\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, 0\rangle)$ with $(x, m) \in R$. Define A output $\langle x, m\rangle$. Let ϖ be a random string and define $\texttt{prove}$ that on input $\langle x, m\rangle$ returns ϖ and the $\texttt{verify}$ that on input $\langle x, \varpi\rangle$ returns 0. It follows immediately that the history string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ with adversary $\mathcal{S}^{\Sigma}_{\text{comp}}$ and the dummy functionality $\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}$ would equal w.

(2) It is easy to show the language $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}}$ is decidable. □

In order to obtain the bad language for the completeness property we extend $B^{\text{I/O}}_{\mathsf{ZK},\text{comp}}$ as follows:

$$B^{\text{ext}}_{\mathsf{ZK},\text{comp}} = \left\{ w \in L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}} \;\middle|\; \exists w' \in B^{\text{I/O}}_{\mathsf{ZK},\text{comp}} \text{ such that } w' \preccurlyeq w \right\}$$

We observe that $B^{\text{ext}}_{\mathsf{ZK},\text{comp}}$ is also decidable in polynomial time.

Step 3. Next we define the class of ideal functionalities that corresponds to the completeness property.

Definition D.31 (Canonical Functionality $\mathcal{F}_{\text{comp}}$)**.** *The functionality* $\mathcal{F}_{\text{comp}} \in \mathscr{F}_{\mathsf{ZK}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\mathsf{ZK}}$ *where (1)* $\mathsf{suppress}()$ *is same as in* $\mathcal{F}^{\text{dum}}_{\mathsf{ZK}}$*, and (2)* $\mathsf{validate}(w) = 0$ *if and only if* $w \in B^{\text{ext}}_{\mathsf{ZK},\text{comp}}$.

Theorem D.32. *If $\pi_{\Sigma(\texttt{ZK})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\text{comp}}$ against static adversaries, then $\Sigma(\texttt{ZK})$ is complete.*

Proof. By contradiction, assume scheme $\Sigma(\texttt{ZK})$ is not complete. We need to construct an environment $\mathcal{Z}$ to distinguish the two worlds with non-negligible probability. Based on the successful completeness attacker A, we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A}}_{\text{comp}}$ as defined above. Notice that in the real world, A is a successful completeness attacker against $\Sigma(\texttt{ZK})$, so $\mathcal{Z}$ outputs 1 with non-negligible probability. However in the ideal world, the case that $(x, m) \in R$ and $\phi = 0$ would cause any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\text{comp}}$ to halt, so the environment $\mathcal{Z}$ can never output 1. Therefore the constructed $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

Theorem D.33. *If $\Sigma(\texttt{ZK})$ is complete, then $\pi_{\Sigma(\texttt{ZK})}$ realizes $\mathcal{F}_{\text{comp}}$.*

Proof. Given that no attacker A can win the completeness game above, we need to show that there exists an adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. The adversary $\mathcal{S}$ is designed as the generic adversary for zero-knowledge task, which means $\mathcal{S}$ will simulate the real world. Each time, when $\mathcal{S}$ obtains $(\text{LeakProve}, \langle P, V, sid\rangle, \langle x, m\rangle)$ from $\mathcal{F}_{\text{comp}}$, it simulates party P to make a proof ϖ based on such $\langle x, m\rangle$, and sends $\langle x, \varpi\rangle$ to party V; when the simulated V receives the pair $\langle x, \varpi\rangle$ from party P, party V produces the verification result ϕ, and the ideal world adversary $\mathcal{S}$ sends symbol $(\text{ProveReturn}, \langle P, V, sid\rangle, \langle x, \phi\rangle)$ to $\mathcal{F}_{\text{comp}}$. Further $\mathcal{S}$ will corrupt any party following the instruction of $\mathcal{Z}$. Whenever $\mathcal{S}$ receives a pair $\langle x, \varpi\rangle$ from a corrupted prover P, $\mathcal{S}$ sends $(\text{Prove}, \langle P, V, sid\rangle, \langle x, ?\rangle)$ to $\mathcal{F}_{\text{comp}}$ where "?" is a special symbol that is not part of the input alphabet for the environment, and receives the corresponding LeakProve symbol from the functionality; later when $\mathcal{S}$ receives from the corrupted prover P the pair $\langle x, \varpi\rangle$, $\mathcal{S}$ simulates party V with such pair, and if V returns $\langle x, \phi\rangle$, then $\mathcal{S}$ sends $(\text{ProveReturn}, \langle P, V, sid\rangle, \langle x, \phi\rangle)$ to $\mathcal{F}_{\text{comp}}$.

Assume $\pi_{\Sigma(\texttt{ZK})}$ cannot realize $\mathcal{F}_{\text{comp}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$; A further simulates the real world for the copy of $\mathcal{Z}$.

We let F denote the event that in a run of $\pi_{\Sigma(\texttt{ZK})}$ with $\mathcal{Z}$, no party is corrupted, and the proof ϖ is based on the witness m for the statement x where $(x, m) \in R$, but the verifier is not convinced, i.e., ϖ cannot be verified. Observe that if event F does not occur and the functionality $\mathcal{F}_{\text{comp}}$ does not halt, the simulated $\mathcal{Z}$ cannot distinguish the two worlds; recall that $\mathcal{F}_{\text{comp}}$ halts only when $(x, m') \in R$ and $\phi = 0$; notice that $(x, ?) \in R$ does not hold, which means $\mathcal{F}_{\text{comp}}$ will not halt. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful completeness attacker. □

D.3.2 Soundness

In this subsection, we investigate the consistency property when the prover is corrupted. We start with soundness in the sense of language membership.

Definition D.34 (Soundness)**.** *A zero-knowledge proof scheme $\Sigma(\text{ZK}) = \langle \texttt{prove}, \texttt{verify}\rangle$ is* sound *if for all PPT attackers* A *it holds that*

$$\Pr[(x, \varpi) \leftarrow \mathsf{A}(1^\lambda); \phi \leftarrow \texttt{verify}(x, \varpi) : x \notin L_R \wedge \phi = 1] \leq \mathsf{negl}(\lambda).$$

However, when we apply our methodology directly on the soundness game, the obtained environment $\mathcal{Z}_{\text{sound}}$ should test an NP-predicate to determine if $x \in L_R$, and the bad language

$$B^{\text{I/O}}_{\texttt{ZK},\text{sound}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\text{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle) \\ \qquad (\text{ProveReturn}, \langle P, V, sid\rangle, \langle x, 1\rangle) \\ \text{such that } x \notin L_R \end{array} \right\}$$

would be undecidable in polynomial time. To handle this issue, we apply our methodology to a stronger soundness game, i.e., a knowledge-extraction game.

Definition D.35 (Soundness in the sense of Knowledge Extraction). *A zero-knowledge proof scheme* $\Sigma(\mathrm{ZK}) = \langle \texttt{prove}, \texttt{verify} \rangle$ *is* sound in the sense of knowledge extraction *for the* NP *relation* R *if there exists a PPT* E *such that for all PPT attackers* A *it holds that*

$$\Pr[(x, \varpi) \leftarrow \mathsf{A}(1^\lambda); \phi \leftarrow \texttt{verify}(x, \varpi); m \leftarrow \mathsf{E}(x, \varpi) : (x, m) \notin R \wedge \phi = 1] \leq \mathsf{negl}(\lambda).$$

The above property can be modeled by a consistency game, G_{sound}, as follows. The challenger C uses as oracles the algorithms $\texttt{prove}()$, $\texttt{verify}()$ and the knowledge extractor E, and interacts with the attacker A: the attacker A outputs a pair $\langle x, \varpi \rangle$; the challenger queries the extractor with such pair and obtains the response m; the challenger also queries the verification oracle with the pair and obtains the response ϕ. The judge J decides that A wins the game if $(x, m) \notin R$ and $\phi = 1$. Next we describe the three steps of the transformation outlined in Section 3.1.

Step 1. We construct an environment $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$ and the corresponding ideal world adversary $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ based on the game G_{sound} described above. In order to simulate the game, the environment first picks P, V from the namespace at random as well as a random sid. Then, the environment requests the corruption of party P and simulates A on input 1^λ; once A outputs $\langle x, \varpi \rangle$, the environment gives $\langle x, \varpi \rangle$ to $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ and requests from $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ to make a proof ϖ for statement x from the party P to party V; at the same time, the environment simulates E on input $\langle x, \varpi \rangle$ to obtain an output m. If $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$ receives a symbol $(\textsc{ProveReturn}, \langle P, V, sid \rangle, \langle x, 1 \rangle)$ from V, and $(x, m) \notin R$, it terminates with 1; otherwise it terminates with 0.

The ideal-world adversary $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ communicates with the environment $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$ and converts this interaction to the appropriate symbols the functionality understands. In particular, $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ will relay the corruption request of $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$ to the functionality. When $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ receives the pair $\langle x, \varpi \rangle$, it simulates E on input $\langle x, \varpi \rangle$ to obtain an output m; then it provides a symbol $(\textsc{Prove}, \langle P, V, sid \rangle, \langle x, m \rangle)$ to the functionality on behalf of P; further if $\texttt{verify}(x, \varpi) = 1$, it also sends a symbol $(\textsc{InflProve}, \langle P, V, sid \rangle, \langle x, 1 \rangle)$ to the functionality, else if $\texttt{verify}(x, \varpi) = 0$, it sends a symbol $(\textsc{InflProve}, \langle P, V, sid \rangle, \langle x, 0 \rangle)$ to the functionality.

Step 2. For any soundness attacker A, scheme Σ and extractor E, the environment $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$, the adversary $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$, and the dummy canonical zero-knowledge functionality together give rise to the language $L^{\mathrm{I/O}}_{\mathcal{F}_{\mathsf{ZK}}^{\mathrm{dum}}, \mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}, \mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}}$ (cf. Section 2). We consider the subset of strings $B^{\mathrm{I/O}}_{\mathsf{ZK},\text{sound}}$ of the union of the I/O languages quantified over all possible soundness attackers A, extractors E and schemes Σ that contains exactly those strings that correspond to the case where the environment returns 1. Formally,

$$B^{\mathrm{I/O}}_{\mathsf{ZK},\text{sound}} \overset{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma,\mathsf{E}} B^{\mathrm{I/O}}_{\mathcal{F}_{\mathsf{ZK}}^{\mathrm{dum}}, \mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}, \mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}}$$

We prove the following facts about this language:

Lemma D.36. *(1)* $B^{\mathrm{I/O}}_{\mathsf{ZK},\text{sound}} = \left\{ w \;\middle|\; \begin{array}{l} w = (\textsc{Prove}, \langle P, V, sid \rangle, \langle x, m \rangle) \\ \qquad (\textsc{ProveReturn}, \langle P, V, sid \rangle, \langle x, 1 \rangle) \\ \textit{such that } (x, m) \notin R \end{array} \right\}$*, and (2)* $B^{\mathrm{I/O}}_{\mathsf{ZK},\text{sound}}$ *is decidable in polynomial time.*

Proof. (1) Denote by Y the language in the right hand side of the lemma's statement (1). First we need to show $B^{\mathrm{I/O}}_{\mathsf{ZK},\text{sound}} \subseteq Y$. Let w be any string in $B^{\mathrm{I/O}}_{\mathsf{ZK},\text{sound}}$; then it holds that there exist $\mathsf{A}, \Sigma, \mathsf{E}$ such that w equals the history string in the ideal world execution of the environment $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$ with adversary $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$ and the dummy functionality $\mathcal{F}_{\mathsf{ZK}}^{\mathrm{dum}}$. Based on the definition of the environment $\mathcal{Z}_{\text{sound}}^{\mathsf{A},\mathsf{E}}$ and the adversary $\mathcal{S}_{\text{sound}}^{\Sigma,\mathsf{E}}$, we know that the symbol $(\textsc{Prove}, \langle P, V, sid \rangle, \langle x, m \rangle)$ will be patched by the adversary into history in the dummy functionality; and then after receiving an $\textsc{InflProve}$ symbol from the adversary, the symbol $(\textsc{ProveReturn}, \langle P, V, sid \rangle, \langle x, 1 \rangle)$ will be recorded in history and be sent to the dummy party V by the functionality. It follows that the string w belongs to the set Y.

Second we need to show $B^{\mathrm{I/O}}_{\mathsf{ZK},\mathrm{sound}} \supseteq Y$. Let w be any string in Y. We will construct $\mathsf{A}, \Sigma, \mathsf{E}$ such that in the ideal world execution of $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\mathrm{sound}}$ with adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\mathrm{sound}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathsf{ZK}}$ it holds that $\mathsf{history} = w$. Given that $w \in Y$ it holds that there exist x, m such that $w = (\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ $(\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ with $(x, m) \notin R$. Let ϖ be a random string and define A output always $\langle x, \varpi\rangle$. Next define E as follows: given $\langle x, \varpi\rangle$ it returns m; for any other input it returns a random string. Finally define $\mathtt{prove}$ that on input $\langle x, m\rangle$ returns ϖ and the $\mathtt{verify}$ that on input $\langle x, \varpi\rangle$ returns 1. It follows immediately that the $\mathsf{history}$ string that in the ideal world execution of $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\mathrm{sound}}$ with adversary $\mathcal{S}^{\Sigma,\mathsf{E}}_{\mathrm{sound}}$ and the dummy functionality $\mathcal{F}^{\mathrm{dum}}_{\mathsf{ZK}}$ would equal w.

(2) Still we need to show the language $B^{\mathrm{I/O}}_{\mathsf{ZK},\mathrm{sound}}$. This follows easily based on the decidability of R. $\square$

In order to obtain the bad language for the soundness property we extend $B^{\mathrm{I/O}}_{\mathsf{ZK},\mathrm{sound}}$ as follows:

$$B^{\mathrm{ext}}_{\mathsf{ZK},\mathrm{sound}} = \left\{ w \in L^{\mathrm{I/O}}_{\mathcal{F}^{\mathrm{dum}}_{\mathsf{ZK}}} \;\middle|\; \exists w' \in B^{\mathrm{I/O}}_{\mathsf{ZK},\mathrm{sound}} \text{ such that } w' \preccurlyeq w \right\}$$

Observe that $B^{\mathrm{ext}}_{\mathsf{ZK},\mathrm{sound}}$ is also polynomial-time decidable.

Step 3. We next define the class of ideal functionalities that corresponds to the knowledge-extraction property.

Definition D.37 (Canonical Functionality $\mathcal{F}_{\mathrm{sound}}$)**.** *The functionality $\mathcal{F}_{\mathrm{sound}} \in \mathscr{F}_{\mathsf{ZK}}$ equals $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}_{\mathsf{ZK}}$ where (1) $\mathsf{suppress}()$ is same as in $\mathcal{F}^{\mathrm{dum}}_{\mathsf{ZK}}$, and (2) $\mathsf{validate}(w) = 0$ if and only if $w \in B^{\mathrm{ext}}_{\mathsf{ZK},\mathrm{sound}}$.*

Theorem D.38. *If $\pi_{\Sigma(\mathsf{ZK})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{sound}}$ against static adversaries, then $\Sigma(\mathsf{ZK})$ is sound in the sense of knowledge extraction.*

Proof. Given that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds, we need to show that there exists a knowledge extractor E such that no A can convince the verifier without the witness being recovered by E (cf., the definition for soundness in the sense of knowledge extraction above).

We construct an extractor E as follows. Given input $\langle x, \varpi\rangle$, E simulates $\mathcal{S}$. E gives $\mathcal{S}$ the symbol $(\textsc{Corrupt}, P)$ as well as the instruction to deliver the pair $\langle x, \varpi\rangle$ to the verifier V on behalf of the corrupted prover P. Whenever E receives from $\mathcal{S}$ the symbol $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ (i.e., the symbol that in an ideal world execution would be given by $\mathcal{S}$ to the functionality on behalf of the corrupted prover P), it sends symbol $(\textsc{LeakProve}, \langle P, V, sid\rangle, x)$ to $\mathcal{S}$ on behalf of the ideal functionality. If E receives the symbol $(\textsc{InflProve}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ from $\mathcal{S}$, it outputs m, otherwise $\perp$. This completes the description of E.

Assume for the sake of contradiction that the scheme $\Sigma(\mathsf{ZK})$ is not sound in the sense of knowledge extraction w.r.t. the extractor E given above, which implies that there exists a successful soundness attacker A (as in the definition of soundness in the sense of knowledge extraction) that wins the game with non-negligible probability β. That is $\Pr[(x, \varpi) \leftarrow \mathsf{A}(1^\lambda); \phi \leftarrow \mathtt{verify}(x, \varpi); m \leftarrow \mathsf{E}(x, \varpi) : (x, m) \notin R \wedge \phi = 1] = \beta$.

We now construct an environment $\mathcal{Z}$ that distinguishes the ideal and the real worlds with non-negligible probability thus refuting the premise of the theorem and deriving a contradiction. The construction of $\mathcal{Z}$ is similar to $\mathcal{Z}^{\mathsf{A},\mathsf{E}}_{\mathrm{sound}}$ as defined above. The environment $\mathcal{Z}$ first corrupts party P; then the environment simulates A as follows: when A outputs a pair $\langle x, \varpi\rangle$, the environment simulates E on input $\langle x, \varpi\rangle$ to obtain m. Then, the environment $\mathcal{Z}$ gives the pair $\langle x, \varpi\rangle$ to the adversary as the statement and the proof produced by the corrupted prover P with the instruction to deliver it to the honest verifier V. When $\mathcal{Z}$ receives $(\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ from party V, the environment simulates party V on input $\langle x, \varpi\rangle$ and obtains ϕ (note that the environment is merely repeating the computation of V, i.e., it simply checks if $\langle x, \varpi\rangle$ is valid taking advantage of the public-coin nature of the protocol). If $(x, m) \notin R$ and $\phi = 1$, the environment terminates with 1; in any other case, it terminates with 0.

First we consider the case that $\mathcal{Z}$ interacts with the real world. Define R_1 as the event that in the real world execution, the pair produced by A is valid, and we let α_{r1} denote the probability the event occurs, i.e., $\alpha_{r1} = \Pr[R_1] = \Pr[(x, \varpi) \leftarrow \mathsf{A}(1^\lambda); \phi \leftarrow \mathtt{verify}(x, \varpi); m \leftarrow \mathsf{E}(x, \varpi) : \phi = 1]$. Define R_2 as the event of the conditional space on R_1 when the extractor E outputs m such that $(x,m) \notin R$; let α_{r2} denote the probability of the event, i.e., $\alpha_{r2} = \Pr[R_2]$. It follows that $\beta = \alpha_{r1}\alpha_{r2}$. Define R_3 as the event in the conditional space on R_1 when the environment receives $(\text{PROVERETURN}, \langle P, V, sid\rangle, \langle x, 1\rangle)$; let α_{r3} denote the probability of this event, i.e., $\alpha_{r3} = \Pr[R_3]$. Note that when R_1 occurs, since the verification is deterministic, the verifier V given input a valid pair $\langle x, \varpi\rangle$ always returns $(\text{PROVERETURN}, \langle P, V, sid\rangle, \langle x, 1\rangle)$, which means the event R_3 occurs with probability 1, i.e., we have $\alpha_{r3} = 1$. It follows that the probability that $\mathcal{Z}$ outputs 1 when it interacts with the real world is $\alpha_r = \Pr[R_1]\Pr[R_3 R_2]$. Note that given that R_1 occurs, the event R_3 will occur with probability 1, and at the same time the event R_2 will occur with probability α_{r2}, and we have $\Pr[R_3 R_2] = 1 \cdot \alpha_{r2} = \alpha_{r2}$. So, $\alpha_r = \Pr[R_1]\Pr[R_3 R_2] = \alpha_{r1}\alpha_{r2} = \beta$. We have $\alpha_{r1} \geq \beta$ and $\alpha_{r2} \geq \beta$; since β is non-negligible, so are α_{r1}, α_{r2}. Further, we have $\alpha_r = \beta$.

Next we consider the case that $\mathcal{Z}$ interacts with the ideal world. We define I_1 as the event that in the ideal world execution the pair produced by A is valid and we let $\alpha_{i1} = \Pr[I_1]$. We define I_2 as the event of the conditional space on I_1 when the extractor E outputs m such that $(x,m) \notin R$ and we let $\alpha_{i2} = \Pr[I_2]$. Notice that $\alpha_{i1} = \alpha_{r1}$ and $\alpha_{i2} = \alpha_{r2}$ since the events I_1, I_2 and R_1, R_2 are in correspondence. Next, define I_3 as the event in the conditional space on I_1 when the environment receives $(\text{PROVERETURN}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ and denote $\alpha_{i3} = \Pr[I_3]$.

Claim: The events I_2 and I_3 are complementary. Recall that both I_2 and I_3 are over the conditional space on I_1. Whenever I_2 occurs we have that E given a valid pair $\langle x, \varpi\rangle$ outputs m s.t. $(x,m) \notin R$. Given that E bases its output on the simulation of $\mathcal{S}$ this implies that $\mathcal{S}$ does not influence the functionality with a $(\text{INFLPROVE}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ symbol, or in case it does the witness provided by $\mathcal{S}$ in the $(\text{PROVE}, \langle P, V\rangle, \langle x, m\rangle)$ symbol is not valid witness. On the other hand, whenever the event I_3 occurs we have that the environment receives $(\text{PROVERETURN}, \langle P, V, sid\rangle, \langle x, 1\rangle)$. This means that the ideal world adversary $\mathcal{S}$ given the valid pair $\langle x, \varpi\rangle$ makes the ideal functionality to output $(\text{PROVERETURN}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ to the environment. This can only happen when $\mathcal{S}$ produces the $(\text{INFLPROVE}, \langle P, V, sid\rangle, \langle x, 1\rangle)$ symbol and at the same time it provided a valid witness to the functionality in the symbol $(\text{PROVE}, \langle P, V\rangle, \langle x, m\rangle)$. It is easy to see that I_2, I_3 are complementary.

Next we compute the probability that $\mathcal{Z}$ outputs 1 when it interacts with the ideal world as follows $\alpha_i = \Pr[I_1]\Pr[I_3 I_2]$. When I_1 occurs, I_2 occurs with probability α_{i2}, and at the same time, I_3 occurs with probability α_{i3}; so $\Pr[I_2 I_3] = \alpha_{i2}\alpha_{i3}$ given that the two events are independent. Based on the claim above $\alpha_{i2} + \alpha_{i3} = 1$. Thus we have $\alpha_i = \Pr[I_1]\Pr[I_2 I_3] = \alpha_{i1}\alpha_{i2}\alpha_{i3} = \alpha_{i1}\alpha_{i2}(1-\alpha_{i2}) = \alpha_{r1}\alpha_{r2}(1-\alpha_{r2}) = (1-\alpha_{r2})\beta$.

The difference in the probability that the environment returns 1 between the two worlds is $\alpha_r - \alpha_i = (1-(1-\alpha_{r2}))\beta = \alpha_{r2}\beta \geq \beta^2$. Since β is non-negligible, so is the difference. Therefore $\mathcal{Z}$ can distinguish the ideal and the real worlds with non-negligible probability. $\square$

Theorem D.39. *If $\Sigma(\mathtt{ZK})$ is sound in the sense of knowledge-extraction, then $\pi_{\Sigma(\mathtt{ZK})}$ realizes $\mathcal{F}_{\text{sound}}$.*

Proof. By assumption there exists a knowledge extractor E such that no A can win the game given in the knowledge-extraction definition above. We need to show that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. Given such E we construct an $\mathcal{S}$ that simulates E; we let $\mathcal{S} = \mathcal{S}^{\Sigma,\mathsf{E}}_{\text{sound}}$ as above. Each time, when $\mathcal{S}$ obtains $(\text{LEAKPROVE}, \langle P, V, sid\rangle, \langle x, m\rangle)$ from $\mathcal{F}_{\text{sound}}$, it simulates party P to make a proof ϖ based on such $\langle x, m\rangle$, and sends $\langle x, \varpi\rangle$ to party V; when the simulated V receives the pair $\langle x, \varpi\rangle$ from party P, party V produces the verification result ϕ, and the ideal world adversary $\mathcal{S}$ sends symbol $(\text{INFLPROVE}, \langle P, V, sid\rangle, \langle x, \phi\rangle)$ to $\mathcal{F}_{\text{sound}}$. Further $\mathcal{S}$ corrupts parties following the instructions of $\mathcal{Z}$. Whenever $\mathcal{S}$ receives a pair $\langle x, \varpi\rangle$ from a corrupted prover P, $\mathcal{S}$ runs the extractor E to obtain m' and play the role of the prover P to send $(\text{PROVE}, \langle P, V, sid\rangle, \langle x, m'\rangle)$ to $\mathcal{F}_{\text{sound}}$, and receives the corresponding LEAKPROVE symbol from the functionality; later when $\mathcal{S}$ receives from the corrupted prover P the pair $\langle x, \varpi\rangle$, $\mathcal{S}$ simulates party V with such pair, and if V returns $\langle x, \phi\rangle$, then $\mathcal{S}$ sends $(\text{INFLPROVE}, \langle P, V, sid\rangle, \langle x, \phi\rangle)$ to $\mathcal{F}_{\text{sound}}$.

Next we justify the validity of $\mathcal{S}$ based on the validity of E. Assume $\pi_{\Sigma(\mathtt{ZK})}$ cannot realize $\mathcal{F}_{\text{sound}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can distinguish the two worlds with non-negligible probability.

We construct A by simulating a copy of $\mathcal{Z}$. When $\mathcal{Z}$ sends out $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ to party P, A computes $\varpi \leftarrow \texttt{prove}(x, m)$ and sends $\langle x, \varpi\rangle$ through the real world dummy adversary to the environment. Later if $\mathcal{Z}$ corrupts the party P, then A sends m and the random coins used for computing ϖ through the dummy adversary to the environment. When $\mathcal{Z}$ sends $\langle x, \varpi\rangle$ through the dummy adversary to party V, A computes $\phi \leftarrow \texttt{verify}(x, \varpi)$, and returns the environment the symbol $(\textsc{ProveReturn}, \langle P, V, sid\rangle, \langle x, \phi\rangle)$. Note that A sets $\langle x, \varpi\rangle$ as its output.

We let F denote the event that $\mathcal{Z}$ outputs $\langle x, \varpi\rangle$, where ϖ is a proof which can be verified for statement x, and m can be extracted such that $(x, m) \notin R$. Observe that if event F does not occur and $\mathcal{F}_{\text{sound}}$ does not halt, the simulated $\mathcal{Z}$ cannot distinguish the two worlds; recall that $\mathcal{F}_{\text{sound}}$ halts only when $(x, m) \notin R$ and $\phi = 1$; notice that given the extractor E, $(x, m) \notin R$ holds with only negligible probability, which means $\mathcal{F}_{\text{sound}}$ halts with negligible probability. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful knowledge-extractability attacker. □

D.3.3 Zero-knowledge

In this section we demonstrate the translation of the simulation-based hiding game for the zero-knowledge property to its corresponding canonical functionality.

Definition D.40 (Zero-knowledge)**.** *A scheme* $\Sigma(\mathtt{ZK}) = \langle \texttt{prove}, \texttt{verify}\rangle$ *is* zero-knowledge *if there exists a PPT* S*, such that for all PPT attackers* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *it holds that*

$$\Pr\left[\begin{array}{l} (x, m, st) \leftarrow \mathsf{A}_1(1^\lambda); b \xleftarrow{r} \{0,1\}; \\ \textbf{if } b = 0 \textbf{ then } \varpi \leftarrow \texttt{prove}(x, m) \textbf{ else } \varpi \leftarrow \mathsf{S}(x); \\ b^* \leftarrow \mathsf{A}_2(st, \varpi) \ : \ b = b^* \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The above definition can be modeled as a hiding game G_{zk} for the task $\mathtt{ZK}$ as follows. The challenger C uses the algorithms $\texttt{prove}()$, $\texttt{verify}()$ and the simulator S as oracles, and interacts with the attacker $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. First A_1 produces a pair $\langle x, m\rangle$ for the challenger C. Then C flips a coin b and if $b = 0$ then C queries the $\texttt{prove}()$ oracle with $\langle x, m\rangle$ to obtain ϖ. If $b = 1$, C provides to S the value x and obtains ϖ. In either case, C sends ϖ to A_2 to obtain a bit b^* which is the attacker's guess of b. The judge J decides that A wins the game if $b^* = b$. We next proceed to apply the methodology of Section C.

Step 1. We construct an environment $\mathcal{Z}_{\text{zk}}^{\mathsf{A},\mathsf{S}}$ and the corresponding ideal world adversary $\mathcal{S}_{\text{zk}}^{\Sigma,\mathsf{S}}$ based on the game G_{zk} described above and a scheme Σ and a simulator S. The environment $\mathcal{Z}_{\text{zk}}^{\mathsf{A},\mathsf{S}}$ first picks P, V from the namespace at random as well as a random sid. Then it requests the corruption of party V and simulates A_1 on input 1^λ. Once A_1 produces $\langle x, m, st\rangle$, $\mathcal{Z}_{\text{zk}}^{\mathsf{A},\mathsf{S}}$ flips a coin b, and if $b = 0$ it sends to party P the symbol $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$. On the other hand, if $b = 1$, it runs internally the simulator S on input x. In either case the environment will obtain a value ϖ (if $b = 0$ the value ϖ will be obtained from the adversary whereas if $b = 1$ the value ϖ will be computed internally by S). Then, $\mathcal{Z}_{\text{zk}}^{\mathsf{A},\mathsf{S}}$ will run A_2 on input $\langle st, \varpi\rangle$ to obtain b^* and will return 1 if and only if $b = b^*$. The ideal world adversary $\mathcal{S}_{\text{zk}}^{\Sigma,\mathsf{S}}$, whenever it receives a symbol $(\textsc{LeakProve}, \langle P, V, sid\rangle, x)$, it will simulate S and send ϖ to the environment.

Step 2. Based on the environment $\mathcal{Z}_{\text{zk}}^{\mathsf{A},\mathsf{S}}$ we define the functionality class that corresponds to the hiding game:

Definition D.41 (Canonical Functionality $\mathcal{F}_{\text{zk}}$)**.** *The functionality* $\mathcal{F}_{\text{zk}} \in \mathscr{F}_{\mathtt{ZK}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}$*, where (1)* $\mathsf{validate}() = 1$ *always, and (2)* $\mathsf{suppress}(\mathsf{a}) = (-)^{|x|+|m|}$ *for* $\mathsf{a} = (\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$.

Theorem D.42. *If* $\pi_{\Sigma(\mathtt{ZK})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\text{zk}}$ *against static adversaries, then* $\Sigma(\mathtt{ZK})$ *is zero-knowledge.*

Proof. Given that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. We need to show that there exists a simulation machine S such that no A can win the hiding game. We can construct S by simulating the adversary $\mathcal{S}$ inside; notice that for any $\mathcal{F} \gtrsim \mathcal{F}_{\text{zk}}$, the adversary $\mathcal{S}$ obtains such x from the default output of the canonical functionality; therefore S needs to obtain from C the statement x. Next we justify the validity of the simulation machine S based on the validity of the adversary $\mathcal{S}$.

By contradiction, assume scheme $\Sigma(\texttt{ZK})$ is not zero-knowledge, i.e., for all S, there exists a successful zero-knowledge attacker A to output $b^* = b$ with non-negligible probability above $1/2$, say $\beta + 1/2$. We need to construct an environment $\mathcal{Z}$ based on A to distinguish the two worlds with non-negligible probability; we use $\mathcal{Z} = \mathcal{Z}_{\mathrm{zk}}^{\mathsf{A},\mathsf{S}}$ as defined above.

If $b = 0$ and $\mathcal{Z}$ interacts with the real world, the ϖ will be produced by party P using $\texttt{prove}()$ algorithm; now the A is a successful attacker and produces $b^* = b$ with non-negligible probability above one half. However if $b = 0$ and $\mathcal{Z}$ interacts with the ideal world, the ϖ will be produced by $\mathcal{S}$; now even an unbounded A can only produces $b^* = b$ with negligible probability above one half. Therefore, $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. □

Theorem D.43. *If $\Sigma(\texttt{ZK})$ is zero-knowledge, then $\pi_{\Sigma(\texttt{ZK})}$ realizes $\mathcal{F}_{\mathrm{zk}}$ against static adversaries.*

Proof. Given that there exists a simulation machine S such that no A can win the game given in the zero-knowledge property definition above. We need to show that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. We construct $\mathcal{S}$ by simulating S inside; we let $\mathcal{S} = \mathcal{S}_{\mathrm{zk}}^{\Sigma,\mathsf{S}}$ as above. Next we justify the validity of $\mathcal{S}$ based on the validity of S.

Assume $\pi_{\Sigma(\texttt{ZK})}$ cannot realize $\mathcal{F}_{\mathrm{zk}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$ inside; and A further simulates the real world for the copy of $\mathcal{Z}$. When $\mathcal{Z}$ sends out $(\textsc{Prove}, \langle P, V, sid\rangle, \langle x, m\rangle)$ to party P, the attacker A outputs $\langle x, m\rangle$; when A receives ϖ, the pair $\langle x, \varpi\rangle$ will be sent to $\mathcal{Z}$; when $\mathcal{Z}$ outputs a bit ϕ, the attacker A sets $b^* = \phi$ as its output.

Notice that if ϖ is produced by $\varpi \leftarrow \texttt{prove}(x, m)$, then the internally simulated $\mathcal{Z}$ is interacting with the real world, and if ϖ is produced by S, based on the construction of $\mathcal{S}$, the $\mathcal{Z}$ is interacting with the ideal world. Based on the assumption, $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, so A can win the game with non-negligible probability. □

D.3.4 The canonical ideal ZK functionality

The (canonical) ideal ZK functionality would equal $\mathcal{F}_{\mathrm{comp}} \wedge \mathcal{F}_{\mathrm{sound}} \wedge \mathcal{F}_{\mathrm{zk}}$. In light of Proposition 2.5 we obtain the following:

Corollary D.44. *If $\pi_{\Sigma(\texttt{ZK})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{comp}} \wedge \mathcal{F}_{\mathrm{sound}} \wedge \mathcal{F}_{\mathrm{zk}}$, then the zero-knowledge scheme $\Sigma(\texttt{ZK})$ satisfies completeness, knowledge-extraction and zero-knowledge property.*

Functionality $\mathcal{F}_{\mathrm{comp}} \wedge \mathcal{F}_{\mathrm{sound}} \wedge \mathcal{F}_{\mathrm{zk}}$ turns out to be equivalent (in the sense of UC-emulation) to the zero-knowledge functionality as it appears in [Can05]. Similarly to the case of the commitment task, our canonical ZK functionality can easily be generalized to the multi-session setting.

D.4 Oblivious Transfer

Oblivious transfer was introduced by [Rab81]. Even, Goldreich and Lempel [EGL85] defined the "1-out-of-2" oblivious transfer (which we consider in this section); Crepeau [Cré87] showed that the two variants are equivalent.

Following Figure 1, any canonical functionality for oblivious transfer in single session, $\mathcal{F}_{\texttt{OT}}$, is defined for two types of roles, the sender S and the receiver R, with a single action $\textsc{Transfer}$. The WF predicate for $\mathcal{F}_{\texttt{OT}}$, requires that a $\textsc{Transfer}$ should precede $\textsc{TransferReturn}$. The default output DO for $\mathcal{F}_{\texttt{OT}}$ is defined as follows: whenever, history contains the symbols $(\textsc{Transfer}, \langle\langle S, R, sid\rangle, S\rangle, \langle m_0, m_1\rangle)$ and $(\textsc{Transfer}, \langle\langle S, R, sid\rangle, R\rangle, i)$, the default output is $\&(m_i)$ which is the pointer to m_i; otherwise it is empty. Based on the above the dummy functionality $\mathcal{F}_{\texttt{OT}}^{\mathrm{dum}}$ is defined (cf. Definition 2.1).

D.4.1 Correctness

For simplicity here we consider two-move OT-protocols; nevertheless, the results of this section can be extended to protocols with many moves in a straightforward way.

Definition D.45 (Correctness). *An oblivious transfer scheme* $\Sigma(\mathtt{OT}) = \langle \mathtt{select}, \mathtt{transfer}, \mathtt{return}\rangle$ *is* correct *if for all PPT attackers* A*, one of the following holds*

$$\Pr\left[\begin{array}{l}(m_0, m_1, i) \leftarrow \mathsf{A}(1^\lambda); \varpi_R \leftarrow \mathtt{select}(i);\\ \varpi_S \leftarrow \mathtt{transfer}(m_0, m_1, \varpi_R);\\ m' \leftarrow \mathtt{return}(\varpi_S) : m' \neq m_i\end{array}\right] \leq \mathsf{negl}(\lambda).$$

The above property can be modeled by a consistency game, G_{corr}, as follows. The challenger C uses as oracles the algorithms $\mathtt{select}(), \mathtt{transfer}(), \mathtt{return}()$, and interacts with the attacker A: the attacker A outputs $\langle m_0, m_1, i\rangle$; the challenger queries the $\mathtt{select}()$ oracle with i and obtains the response ϖ_R;the challenger queries the $\mathtt{transfer}()$ oracle with the pair $\langle m_0, m_1\rangle$ with ϖ_R and obtains the response ϖ_S; the challenger then queries the $\mathtt{return}()$ oracle again with ϖ_S and obtains the response m'. The judge J decides that A wins the game if $m' \neq m_i$. Next we describe the three steps of the transformation outlined in Section 3.1.

Step 1. We construct an environment $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma}_{\text{corr}}$ based on the game G_{corr} described above. In order to simulate the game, the environment $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$ first picks S, R from the namespace at random as well as a random sid. Then the environment simulates A on input 1^λ; once A outputs $\langle m_0, m_1, i\rangle$, the environment sends $(\textsc{Transfer}, \langle\langle S, R, sid\rangle, R\rangle, i)$ to party R; the environment sends $(\textsc{Transfer}, \langle\langle S, R, sid\rangle, S\rangle, \langle m_0, m_1\rangle)$ to party S; the environment then receives ϖ_S as the proof transcripts through the adversary. Finally $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$ receives a symbol $(\textsc{TransferReturn}, \langle\langle S, R, sid\rangle, R\rangle, m')$ from R, where $m' \neq m_i$, the environment terminates with 1; otherwise terminates with 0.

The ideal-world adversary $\mathcal{S}^{\Sigma}_{\text{corr}}$ translates the communications from $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$ into the appropriate symbols that the canonical oblivious transfer functionality understands.

Step 2. For any correctness attacker A and scheme Σ, the environment $\mathcal{Z}^{\mathsf{A}}_{\text{corr}}$, the adversary $\mathcal{S}^{\Sigma}_{\text{corr}}$, and the dummy canonical oblivious transfer functionality $\mathcal{F}^{\text{dum}}_{\mathtt{OT}}$ together give rise to the language $L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathtt{OT}}, \mathcal{Z}^{\mathsf{A}}_{\text{corr}}, \mathcal{S}^{\Sigma}_{\text{corr}}}$ (cf. Section 2). We consider the subset of strings $B^{\text{I/O}}_{\mathtt{OT},\text{corr}}$ of the union of all the I/O languages quantified over all possible correctness attackers A and schemes Σ that contains exactly those strings corresponds to the case that the environment returns 1. Formally,

$$B^{\text{I/O}}_{\mathtt{OT},\text{corr}} \overset{\mathsf{def}}{=} \bigcup_{\mathsf{A},\Sigma} L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathtt{OT}}, \mathcal{Z}^{\mathsf{A}}_{\text{corr}}, \mathcal{S}^{\Sigma}_{\text{corr}}}$$

We next prove the following characterization of this language as well as determine its time-complexity:

Lemma D.46. *(1)* $B^{\text{I/O}}_{\mathtt{OT},\text{corr}} = \left\{ w \;\middle|\; \begin{array}{l} w = \mathsf{abc}\ \textit{or}\ \mathsf{bac} \\ \textit{where}\ \mathsf{a} = (\textsc{Transfer}, \langle\langle S, R, sid\rangle, S\rangle, \langle m_0, m_1\rangle), \\ \qquad \mathsf{b} = (\textsc{Transfer}, \langle\langle S, R, sid\rangle, R\rangle, i), \\ \qquad \mathsf{c} = (\textsc{TransferReturn}, \langle\langle S, R, sid\rangle, R\rangle, m'), \\ \textit{such that}\ m' \neq m_i \end{array}\right\}$*, and (2)* $B^{\text{I/O}}_{\mathtt{OT},\text{corr}}$ *is decidable in polynomial time.*

In order to obtain the bad language for the correctness property we extend $B^{\text{I/O}}_{\mathtt{OT},\text{corr}}$ as follows:

$$B^{\text{ext}}_{\mathtt{OT},\text{corr}} = \left\{ w \in L^{\text{I/O}}_{\mathcal{F}^{\text{dum}}_{\mathtt{OT}}} \;\middle|\; \exists w' \in B^{\text{I/O}}_{\mathtt{OT},\text{corr}} \text{ such that } w' \preccurlyeq w \right\}$$

We observe that $B^{\text{ext}}_{\mathtt{OT},\text{corr}}$ is also decidable in polynomial time.

Step 3. Next we define the class of ideal functionalities that corresponds to the correctness property.

Definition D.47 (Canonical Functionality $\mathcal{F}_{\text{corr}}$)**.** *The functionality $\mathcal{F}_{\text{corr}} \in \mathscr{F}_{\texttt{OT}}$ equals $\mathcal{F}_{\texttt{OT}}^{\textsf{suppress},\textsf{validate}}$, where (1) $\textsf{suppress}()$ is the same as in $\mathcal{F}_{\texttt{OT}}^{\text{dum}}$, and (2) $\textsf{validate}(w) = 0$ if and only if $w \in B_{\texttt{OT},\text{corr}}^{\text{ext}}$.*

Theorem D.48. *If $\pi_{\Sigma(\texttt{OT})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\text{corr}}$, then $\Sigma(\texttt{OT})$ is correct.*

Proof. By contradiction, assume scheme $\Sigma(\texttt{OT})$ is not correct. We need to construct an environment $\mathcal{Z}$ to distinguish the two worlds with non-negligible probability. Based on the successful correctness attacker A, we use $\mathcal{Z} = \mathcal{Z}_{\text{corr}}^{\mathsf{A}}$ as defined above. Notice that in the real world, A is a successful correctness attacker against $\Sigma(\texttt{OT})$, so $\mathcal{Z}$ outputs 1 with non-negligible probability. However in the ideal world, $m' \neq m_i$ would cause any canonical functionality $\mathcal{F} \gtrsim \mathcal{F}_{\text{corr}}$ to halt, so the environment $\mathcal{Z}$ can never output 1. Therefore the constructed $\mathcal{Z}$ distinguishes the two worlds with non-negligible probability. □

Theorem D.49. *If $\Sigma(\texttt{OT})$ is correct, then $\pi_{\Sigma(\texttt{OT})}$ realizes $\mathcal{F}_{\text{corr}}$.*

Proof. Given that no attacker A can win the correctness game above, we need to show that there exists an ideal adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. The adversary $\mathcal{S}$ is designed as the generic adversary for OT task, which means $\mathcal{S}$ will simulate the real world. Each time, when $\mathcal{S}$ obtains $(\text{LEAKTRANSFER}, \langle\langle S, R, sid\rangle, R\rangle, i)$ from $\mathcal{F}_{\text{corr}}$, it simulates party R internally to send out its choice i by using message ϖ_R, and sends ϖ_R to party S; when $\mathcal{S}$ obtains $(\text{LEAKTRANSFER}, \langle\langle S, R, sid\rangle, S\rangle, \langle m_0, m_1\rangle)$ from $\mathcal{F}_{\text{corr}}$, it simulates party S internally to transfer m_0, m_1 by using message ϖ_S, and sends ϖ_S to party R; when the simulated party R produces its output m', $\mathcal{S}$ sends the symbol $(\text{INFLTRANSFER}, \langle\langle S, R, sid\rangle, R\rangle, m')$ to $\mathcal{F}_{\text{corr}}$. Furthermore $\mathcal{S}$ corrupts the parties following $\mathcal{Z}$'s instruction. Whenever $\mathcal{S}$ receives ϖ_S from a corrupted sender S, $\mathcal{S}$ runs the simulated party R to compute the output m', and then randomly select $\hat{m}$. Based on R's choice $i \in \{0,1\}$, $\mathcal{S}$ sets $m_i = m'$ and $m_{1-i} = \hat{m}$, and then plays the role of the corrupted sender S to send symbol $(\text{TRANSFER}, \langle\langle S, R, sid\rangle, S\rangle, \langle m_0, m_1\rangle)$ to $\mathcal{F}_{\text{corr}}$; $\mathcal{S}$ later sends symbol $(\text{INFLTRANSFER}, \langle\langle S, R, sid\rangle, R\rangle, m')$ to $\mathcal{F}_{\text{corr}}$.

Assume $\pi_{\Sigma(\texttt{OT})}$ cannot realize $\mathcal{F}_{\text{corr}}$, i.e., for all $\mathcal{S}$ there exists an environment $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. We construct A by simulating a copy of $\mathcal{Z}$ internally; and A further simulates the real world for the copy of $\mathcal{Z}$.

We let F denote the event that in a run of $\pi_{\Sigma(\texttt{OT})}$ with $\mathcal{Z}$, no party is corrupted, and the value m' receiver obtained is not the one the receiver selected from the sender's contribution. Observe that if event F does not occur and $\mathcal{F}_{\text{corr}}$ does not halt, the simulated $\mathcal{Z}$ cannot distinguish the two worlds; recall that $\mathcal{F}_{\text{corr}}$ halts only when $m' = m_i$; based on our simulator $\mathcal{S}$ this is always guaranteed except negligible probability, which means $\mathcal{F}_{\text{corr}}$ halts with negligible probability. However $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability, which means event F must occur with non-negligible probability, i.e., A is a successful correctness attacker. □

D.4.2 Sender's security

In this section we demonstrate the translation of the simulation-based hiding game for the security of the sender to its corresponding canonical functionality.

Definition D.50 (Sender's Security)**.** *A scheme $\Sigma(\texttt{OT}) = \langle \texttt{select}, \texttt{transfer}, \texttt{return}\rangle$ satisfies* sender security *if there exist PPT machines S and E, such that for all PPT attackers $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ it holds that*

$$\Pr\left[\begin{array}{l} (m_0, m_1, \varpi_R, st) \leftarrow \mathsf{A}_1(1^\lambda); b \xleftarrow{\mathtt{r}} \{0,1\}; i \leftarrow \mathsf{E}(\varpi_R); \\ \textbf{if } b = 0 \textbf{ then } \varpi_S \leftarrow \texttt{transfer}(m_0, m_1, \varpi_R) \\ \quad \textbf{else if } b = 1 \textbf{ then } \varpi_S \leftarrow \mathsf{S}(i, m_i, \varpi_R); \\ b^* \leftarrow \mathsf{A}_2(st, \varpi_S) \ : \ b = b^* \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The above definition can be modeled as a hiding game G_{ssec} for the task $\texttt{OT}$ as follows. The challenger C uses the algorithms $\texttt{select}(), \texttt{transfer}(), \texttt{return}()$, the simulator S, and the extractor E as oracles, and interacts with the attacker $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. First A_1 produces a tuple $\langle m_0, m_1, \varpi_R \rangle$ for the challenger C. Then C flips a coin b, and if $b = 0$ then C queries the $\texttt{transfer}()$ oracle with $\langle m_0, m_1, \varpi_R \rangle$ to obtain ϖ_S. Else if $b = 1$, C queries extractor E with ϖ_R and obtains i; and if $i = 0$, then C provides to S the value $\langle m_0, \varpi_R \rangle$ and obtains ϖ_S; else if $i = 1$, then C provides to S the value $\langle m_1, \varpi_R \rangle$ and obtains ϖ_S. In all cases, C sends ϖ_S to A_2 to obtain a bit b^* which is the attacker's guess of b. The judge J decides that A wins the game if $b^* = b$. We next proceed to apply the methodology of Section C.

Step 1. We construct an environment $\mathcal{Z}_{\text{ssec}}^{\mathsf{A},\mathsf{S},\mathsf{E}}$ and the corresponding ideal world adversary $\mathcal{S}_{\text{ssec}}^{\Sigma,\mathsf{S},\mathsf{E}}$ based on the game G_{ssec} described above and a scheme Σ, a simulation machine S, and an extractor E. The environment first picks S, R from the namespace at random as well as a random sid. Then it requests the corruption of party R and simulates A_1 on input 1^λ. Once A_1 produces $\langle m_0, m_1, \varpi_R, st \rangle$, the environment $\mathcal{Z}_{\text{ssec}}^{\mathsf{A},\mathsf{S},\mathsf{E}}$ flips a coin b, and if $b = 0$ it sends to party S the symbol $(\textsc{Transfer}, \langle \langle S, R, sid \rangle, S \rangle, \langle m_0, m_1 \rangle)$. On the other hand, if $b = 1$, it runs internally the extractor E on ϖ_R and obtains i; then the environment runs internally the simulator S on input $\langle m_i, \varpi_R \rangle$. In either case the environment will obtain a value ϖ_S (if $b = 0$ the value ϖ_S will be obtained from the adversary whereas if $b = 1$ the value ϖ_S will be computed internally by S). Then, $\mathcal{Z}_{\text{ssec}}^{\mathsf{A},\mathsf{S},\mathsf{E}}$ will run A_2 on input $\langle st, \varpi_S \rangle$ to obtain b^* and will return 1 if and only if $b = b^*$. The ideal world adversary $\mathcal{S}_{\text{ssec}}^{\Sigma,\mathsf{S},\mathsf{E}}$, whenever it receives a symbol $(\textsc{LeakTransfer}, \langle \langle S, R, sid \rangle, S \rangle)$, it will simulate S and send ϖ_S to the environment.

Step 2. Based on the environment $\mathcal{Z}_{\text{ssec}}^{\mathsf{A},\mathsf{S},\mathsf{E}}$ we define the functionality class that corresponds to the mixed game:

Definition D.51 (Canonical Functionality $\mathcal{F}_{\text{ssec}}$)**.** *The functionality $\mathcal{F}_{\text{ssec}} \in \mathscr{F}_{\texttt{OT}}$ equals $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}$, where (1) $\mathsf{validate}() = 1$ always, and (2) $\mathsf{suppress}(\mathsf{a}) = (-)^{|m_0|+|m_1|}$, for* $\mathsf{a} = (\textsc{Transfer}, \langle \langle S, R, sid \rangle, S \rangle, \langle m_0, m_1 \rangle)$.

Theorem D.52. *If $\pi_{\Sigma(\texttt{OT})}$ realizes some $\mathcal{F} \gtrsim \mathcal{F}_{\text{ssec}}$ against static adversaries, then $\Sigma(\texttt{OT})$ satisfies sender's security.*

Proof. Given that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. We need to show that there exists a simulation machine S such that no A can win the hiding game. We can construct S by simulating the adversary $\mathcal{S}$; we justify the validity of the simulation machine S based on the validity of the adversary $\mathcal{S}$.

By contradiction, assume scheme $\Sigma(\texttt{OT})$ does not satisfy sender security, i.e., for all S, there exists a successful attacker A to output $b^* = b$ with non-negligible probability above $1/2$, say $\beta + 1/2$. We need to construct an environment $\mathcal{Z}$ based on A to distinguish the two worlds with non-negligible probability; we use $\mathcal{Z} = \mathcal{Z}_{\text{ssec}}^{\mathsf{A},\mathsf{S},\mathsf{E}}$ as defined above.

If $b = 0$ and $\mathcal{Z}$ interacts with the real world, after receiving ϖ_R, the ϖ_S will be produced by party S using $\texttt{transfer}()$ algorithm over m_0, m_1, ϖ_R; now the A is a successful attacker and produces $b^* = b$ with non-negligible probability above one half. However if $b = 1$ and $\mathcal{Z}$ interacts with the ideal world, obtain i by running E over ϖ_R, then the ϖ_S will be produced by $\mathcal{S}$ over m_i, ϖ_R; now even an unbounded A can only produces $b^* = b$ with negligible probability above one half. Therefore, $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. □

D.4.3 Receiver's security

Definition D.53 (Receiver's Security)**.** *A scheme* $\Sigma(\texttt{OT}) = \langle \texttt{select}, \texttt{transfer}, \texttt{return} \rangle$ *satisfies* receiver security *if there exist a PPT machine* S*, such that for all PPT attackers* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ *it holds that*

$$\Pr\left[\begin{array}{l} (i, st) \leftarrow \mathsf{A}_1(1^\lambda); b \xleftarrow{r} \{0,1\}; \\ \textbf{if } b = 0 \textbf{ then } \varpi_R \leftarrow \texttt{select}(i) \\ \quad \textbf{else if } b = 1 \textbf{ then } \varpi_R \leftarrow \mathsf{S}(1^\lambda); \\ b^* \leftarrow \mathsf{A}_2(st, \varpi_R) \ : \ b = b^* \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The above definition can be modeled as a hiding game G_{rsec} for the task $\mathtt{OT}$ as follows. The challenger C uses the algorithms $\mathtt{select}(), \mathtt{transfer}(), \mathtt{return}()$, the simulator S as oracles, and interacts with the attacker $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. First A_1 produces $\langle i, st\rangle$ for the challenger C. Then C flips a coin b and if $b = 0$ then C queries the $\mathtt{select}()$ oracle with i to obtain ϖ_R. If $b = 1$, C queries S without such i and obtains ϖ_R. In both cases, C sends ϖ_R to A_2 to obtain a bit b^* which is the attacker's guess of b. The judge J decides that A wins the game if $b^* = b$. We next proceed to apply the methodology of Section C.

Step 1. We construct an environment $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\mathrm{rsec}}$ and the corresponding ideal world adversary $\mathcal{S}^{\Sigma,\mathsf{S}}_{\mathrm{rsec}}$ based on the game G_{rsec} described above and a scheme Σ, a simulation machine S. The environment first picks S, R from the namespace at random as well as a random sid. Then it requests the corruption of party R and simulates A_1 on input 1^λ. Once A_1 produces $\langle i, st\rangle$, the environment $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\mathrm{rsec}}$ flips a coin b, and if $b = 0$ it sends to party R the symbol $(\textsc{Transfer}, \langle\langle S, R, sid\rangle, R\rangle, i)$. On the other hand, if $b = 1$, the environment runs internally the simulator S. In either case the environment will obtain a value ϖ_R (if $b = 0$ the value ϖ_R will be obtained from the adversary whereas if $b = 1$ the value ϖ_R will be computed internally by S). Then, $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\mathrm{rsec}}$ will run A_2 on input $\langle st, \varpi_R\rangle$ to obtain b^* and will return 1 if and only if $b = b^*$. The ideal world adversary $\mathcal{S}^{\Sigma,\mathsf{S}}_{\mathrm{rsec}}$, whenever it receives a symbol $(\textsc{LeakTransfer}, \langle\langle S, R, sid\rangle, R\rangle)$, it will simulate S and send ϖ_R to the environment.

Step 2. Based on the environment $\mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\mathrm{rsec}}$ we define the functionality class that corresponds to the hiding game:

Definition D.54 (Canonical Functionality $\mathcal{F}_{\mathrm{rsec}}$)**.** *The functionality* $\mathcal{F}_{\mathrm{rsec}} \in \mathscr{F}_{\mathtt{OT}}$ *equals* $\mathcal{F}^{\mathsf{suppress},\mathsf{validate}}$*, where (1)* $\mathsf{validate}() = 1$ *always, and (2)* $\mathsf{suppress}(\mathsf{a}) = -$*, for* $\mathsf{a} = (\textsc{Transfer}, \langle\langle S, R, sid\rangle, R\rangle, i)$.

Theorem D.55. *If* $\pi_{\Sigma(\mathtt{OT})}$ *realizes some* $\mathcal{F} \gtrsim \mathcal{F}_{\mathrm{rsec}}$ *against static adversaries, then* $\Sigma(\mathtt{OT})$ *satisfies receiver's security.*

Proof. Given that there exists an ideal world adversary $\mathcal{S}$ such that no $\mathcal{Z}$ can distinguish the two worlds. We need to show that there exists a simulation machine S such that no A can win the hiding game. We can construct S by simulating the adversary $\mathcal{S}$; we justify the validity of the simulation machine S based on the validity of the adversary $\mathcal{S}$.

By contradiction, assume scheme $\Sigma(\mathtt{OT})$ does not satisfy receiver's security, i.e., for all S, there exists a successful attacker A to output $b^* = b$ with non-negligible probability above $1/2$. We need to construct an environment $\mathcal{Z}$ based on A to distinguish the two worlds with non-negligible probability; we use $\mathcal{Z} = \mathcal{Z}^{\mathsf{A},\mathsf{S}}_{\mathrm{rsec}}$ as defined above.

If $b = 0$ and $\mathcal{Z}$ interacts with the real world, the ϖ_R will be produced by party R using $\mathtt{select}()$ algorithm; now the A is a successful attacker and produces $b^* = b$ with non-negligible probability above one half. However if $b = 0$ and $\mathcal{Z}$ interacts with the ideal world, the ϖ_R will be produced by $\mathcal{S}$; now even an unbounded A can only produces $b^* = b$ with negligible probability above one half. Therefore, $\mathcal{Z}$ can distinguish the two worlds with non-negligible probability. □

Remark D.56. Note that the resulting canonical ideal functionality for oblivious transfer $\mathcal{F}_{\mathtt{OT}} = \mathcal{F}_{\mathrm{ssec}} \wedge \mathcal{F}_{\mathrm{rsec}} \wedge \mathcal{F}_{\mathrm{corr}}$ is different from the corresponding functionalities given in [CLOS02, Can05] (that themselves have important differences). In particular, [Can05] introduced the notion of "delayed output" to the adversary as a mechanism to enable the ideal world adversary to delay the output of a certain action any amount time necessary to make the view of the environment indistinguishable to the real world's. This is important as failing to provide such capability to the adversary may enable an impossibility result due to the existence of environments that can tell the real world from the ideal by simply observing the failure of the simulator to "synchronize" with the protocol flow in the real world. While the introduction of delayed output in [Can05] successfully serves functionalities such as zero-knowledge and commitments (that turn out to be identical to our corresponding canonical versions) that is not the case for oblivious transfer. This is due to the fact that the basic action in oblivious transfer requires the input contribution from both the sender and the receiver prior to producing output. This asks for a more finely grained interaction between the ideal functionality and the ideal world adversary. In our setting this is captured by the LEAKTRANSFER symbols that are sent to the adversary whenever a TRANSFER symbol is submitted by either the sender or the receiver (and note that none of these symbols produce output to the receiver). In contrast, in the oblivious transfer functionality of [Can05] such notifications are handled with two delayed outputs something that forces the ideal functionality to wait when the receiver's input is submitted in case the sender has not submitted his input yet (cf. [Can05], Figure 25, page 108). This may induce an impossibility result for protocols where the receiver is supposed to send the first message in the oblivious transfer protocol: in such case the environment can distinguish the real from the ideal world by activating the receiver without activating the sender and observing the network communication. This would not affect our canonical formulation of the oblivious transfer functionality that notifies using the LEAKTRANSFER symbols the ideal world adversary whenever either party provides input.