Equivocal Blind Signatures and Adaptive UC-Security

Aggelos Kiayias * Hong-Sheng Zhou*

September 4, 2007

Abstract

We study the design of adaptively secure blind signatures in the universal composability (UC) setting. First, we introduce a new property for blind signature schemes that is suitable for arguing security against adaptive adversaries: an *equivocal blind signature* is a blind signature where there exists a simulator that has the power of making arbitrary signing transcripts to correspond to any given message signature pair. Second, we present a general construction methodology for building adaptively secure blind signatures: the starting point is a 2-move "lite blind signature", a lightweight 2-party signature protocol that we formalize and implement both generically as well as number theoretically; formalizing a primitive as "lite" means that the adversary is required to show all private tapes of adversarially controlled parties; this enables us to conveniently separate zero-knowledge (ZK) related security requirements from the remaining security properties in the primitive's design methodology. Next, we focus on the ZK requirements for building blind signatures. To this effect, we formalize two special ZK ideal functionalities, single-verifier-ZK (SVZK) and single-prover-ZK (SPZK) and we investigate the requirements for realizing them in a commit-and-prove fashion as building blocks for adaptively secure UC blind signatures. SVZK can be realized without relying on a multi-session UC commitment; as a result, we can realize SVZK using "mixed-commitments" while employing only a constant size common reference string (CRS); note that such commitments require a linear CRS in the multi-session setting. Regarding SPZK we find the rather surprising result that realizing it only for static adversaries is sufficient to obtain adaptive security for UC blind signatures. This demonstrates that in a blind signature, the signer can employ a commit-and-prove protocol that is based merely on an extractable commitment and even though this will lack "state-reconstruction" it will still be sufficient for non-erasure adaptive security.

We instantiate all the building blocks of our design methodology both generically based on the blind signature construction of Fischlin as well as concretely based on the 2SDH assumption of Okamoto, thus demonstrating the feasibility and practicality of our approach.

*University of Connecticut, Computer Science and Engineering, Storrs, CT, USA, {aggelos,hszhou}@cse.uconn.edu.

1 Introduction

A blind signature is a cryptographic primitive that was proposed by Chaum [Cha82]; it is a digital signature scheme where the signing algorithm is split into a two-party protocol between a user (or client) and a signer (or server). The signing protocol's functionality is that the user can obtain a signature on a message that she selects in a blind fashion, i.e., without the signer being able to extract some useful information about the message from the protocol interaction. At the same time the existential unforgeability property of digital signatures should hold, i.e., after the successful termination of a number of n corrupted user instantiations, an adversary should be incapable of generating signatures for $(n + 1)$ distinct messages.

A blind signature is a very useful privacy primitive that has many applications in the design of electronic-cash schemes, the design of electronic voting schemes as well as in the design of anonymous credential systems. Since the initial introduction of the primitive, a number of constructions have been proposed [Dam88, OO89, Oka92, PS96, JLO97, PS97, Poi98, PS00, AO00, Abe01, AO01, BNPS03, Bol03, CKW04, KZ06, Oka06, Fis06, HKKL07, CNS07] based on different intractability assumptions and security models with various communication and time complexities. The first formal treatment of the primitive in a stand-alone model and assuming random oracles (RO) was given by Pointcheval and Stern in [PS96].

Blind signatures is in fact one of the few complex cryptographic primitives (beyond digital signatures, public-key encryption, and key-exchange) that have been implemented in real world Internet settings (e.g., in the Votopia [Kim04] voting system) and thus the investigation of more realistic attack models for blind signatures is of pressing importance. Juels, Luby and Ostrovsky [JLO97] presented a formal treatment of blind signatures that included the possibility for an adversary to launch attacks that use arbitrary concurrent interleaving of either user or signer protocols. Still, the design of schemes that satisfied such stronger modeling proved somewhat elusive. In fact, Lindell [Lin03] showed that unbounded concurrent security for blind signatures is impossible under a simulation-based security definition without any setup assumption; more recently in [HKKL07], the generic feasibility of blind signatures without setup assumptions was shown but using a game-based security formulation.

With respect to practical provably secure schemes (which is the focus of the present work), assuming random oracles or some setup assumption, various efficient constructions were proposed: for example, [BNPS03, Bol03] presented efficient two-move constructions in the RO model, while [KZ06, Oka06] presented efficient constant-round constructions without random oracles employing a common reference string (CRS) model (i.e., when a trusted setup function initializes all parties' inputs) that withstand concurrent attacks. While achieving security under concurrent attacks is an important property for the design of useful blind signatures, a blind signature scheme may still be insecure for a certain deployment. Game-based security definitions [PS96, JLO97, CKW04, KZ06, Oka06, HKKL07, CNS07] capture properties that are intuitively desirable. But the successive extensions of definitions in the literature and the differences between the various models in fact exemplify the following: on the one hand capturing all desirable properties of a complex cryptographic primitive such as a blind signature is a difficult task, while on the other, even if such properties are attained, a "provably secure" blind signature may still be insecure if deployed within a larger system. For this reason, it is important to consider the realization of blind signatures under a general simulation-based security formulation such as the one provided in the Universal Composability (UC) framework of Canetti [Can01] that enables us to formulate cryptographic primitives so that they remain secure under arbitrary deployments and interleavings of protocol instantiations.

In the UC setting, against static adversaries, it was shown how to construct blind signatures in the CRS model [Fis06] with two moves of interaction. Though the construction in [Fis06] is round-optimal, it is unknown whether it can admit concrete practical instantiations. In addition, security is argued only against

static adversaries; and while it should be feasible to extend the construction of [Fis06] in the adaptive setting this can only exacerbate the difficulty of concretely realizing the basic design. Note that using the secure two party computation compiler of [CLOS02] one can derive adaptively secure blind signatures but this approach is also generic and does not suggest any concrete design.

1.1 Our Results

In this work we study the design of blind signatures in the UC framework against adaptive adversaries. We focus on maintaining a "practice-oriented" approach that entails the following: (i) a constant number of rounds, (ii) a choice of session scope that is consistent with how a blind signature would be implemented in practice, in particular a multitude of clients and one signer should be supported within a single session, (iii) a trusted setup string that is of constant length in the number of parties within a session, (iv) avoiding, if possible, cryptographic primitives that are "per-bit", such as bit-commitment, where one has to spend a communication length of $\Omega(l)$ where l is a security parameter per bit of private input.

Our results are as follows:

Equivocal blind signatures. We introduce a new property for blind signatures, called equivocality that is suitable for arguing security against adaptive adversaries. In an equivocal blind signature there exists a simulator that has the power to construct the internal state of a client including all random tapes so that any simulated communication transcript can be mapped to any given valid message-signature pair. This capability should hold true even after a signature corresponding to the simulated transcript has been released to the adversary. Equivocality can be seen as a strengthening of the notion of blindness as typically defined in game-based security formulations of blind signatures: in an equivocal blind signature, signing transcripts can be simulated in an independent fashion to the message-signature pair they correspond to.

General methodology for building UC blind signatures. We present a general methodology for designing adaptively secure UC blind signatures. Our starting point is the notion of an *equivocal lite blind signature*: The idea behind "lite" blind signatures is that security properties should hold under the condition that the adversary deposits the private tapes of the parties he controls. This "open-all-private-tapes" approach simplifies the blind signature definitions substantially and allows one to separate security properties that relate to zero-knowledge compared to other necessary properties for blind signatures. Note that this is *not* an honest-but-curious type of adversarial formulation as the adversary is not required to be honestly simulating corrupted parties; in particular, the adversary may deviate from honest protocol specifications as long as he can present private tapes that match his communication transcripts.

We then demonstrate two instantiations of an equivocal lite blind signature, one that is based on generic cryptographic primitives that is inspired by the blind signature construction of [Fis06] and one based on the design and the 2SDH assumption of [Oka06].

Study of the ZK requirements for UC blind signatures. Having demonstrated equivocal lite blind-signature as a feasible starting building block, we then focus on the formulation of the appropriate ZK-functionalities that are required for building blind signatures in the adaptive adversary setting. Interestingly, the user and the signer have different ZK "needs" in a blind signature. In particular the corresponding ZK-functionalities turn out to be simplifications of the standard multi-session ZK functionality $\mathcal{F}_{\text{MZK}}$ that restrict the multi-sessions to occur either from many provers to a single verifier (we call this $\mathcal{F}_{\text{SVZK}}$) or from a single prover to many verifiers (we call this $\mathcal{F}_{\text{SPZK}}$). Note that this stems from our blind signature *session scope* that involves a multitude of users interacting with a single signer (this is consistent with the notion that a blind-signature signer is a server within a larger system and is expected that the number of such servers would be very small compared to a much larger population of users and verifiers).

First, regarding $\mathcal{F}_{\text{SVZK}}$, the ZK protocol that users need to execute, we show that it can be realized in a commit-and-prove fashion using a commitment scheme that, as it is restricted in single-verifier setting, it does not require built-in non-malleability (while such property would be essential for general multi-session UC commitments). We thus proceed to realize $\mathcal{F}_{\text{SVZK}}$ using mixed commitments [DN02, Nie03] with only a constant length common reference string (as opposed to linear in the number of parties that is required in the multi-session setting). Second, regarding $\mathcal{F}_{\text{SPZK}}$, the ZK protocol that signer needs to execute towards the users, we find the rather surprising result that it needs only be realized against static adversaries for the resulting blind signature scheme to satisfy adaptive security! This enables a much more efficient realization design for $\mathcal{F}_{\text{SPZK}}$ as we can implement it using merely an extractable commitment and a Sigma protocol (alternatively, using an Ω-protocol [GMY06]). The intuition behind this result is that in a blind signature the signer is not interested in hiding his input in the same way that the user is: this can be seen by the fact that the verification-key itself leaks a lot of information about the signing-key to the adversary/environment, thus, using a full-fledged zero-knowledge instantiation is an overkill from the signer's point of view; this phenomenon was studied in the context of zero-knowledge in [KZ07b]. We note that our $\mathcal{F}_{\text{SPZK}}$ functionality can be seen as a special instance of client-server computation as considered in [PS05] (where the relaxed non-malleability requirement of such protocols was also noted); interestingly $\mathcal{F}_{\text{SVZK}}$ falls outside that framework (despite its client-server nature).

Notations: $a \xleftarrow{\mathtt{r}} \mathtt{RND}$ denotes randomly selecting a in its domain; $\mathsf{negl}()$ denotes negligible function; $\mathsf{poly}()$ denotes polynomial function.

2 Equivocal Lite Blind Signatures

2.1 Our Basic Building Block: Equivocal Lite Blind Signatures

A signature generation protocol is a tuple $\langle \mathtt{CRSgen}, \mathtt{gen}, \mathsf{lbs}_1, \mathsf{lbs}_2, \mathsf{lbs}_3, \mathtt{verify} \rangle$ where $\mathtt{CRSgen}$ is a common reference string generation algorithm, $\mathtt{gen}$ is a key-pair generation algorithm, lbs_i, $i = 1, 2, 3$, comprise a two-move signature generation protocol between the user U and the signer S as described in Figure 1 and $\mathtt{verify}$ is a signature verification algorithm. A *lite blind signature* is a signature generation protocol that satisfies completeness as well as two security properties, *lite-unforgeability* and *lite-blindness*, defined below (consistency is another property [Can05] for signatures that will be trivially satisfied in our design and thus we omit it in this version).

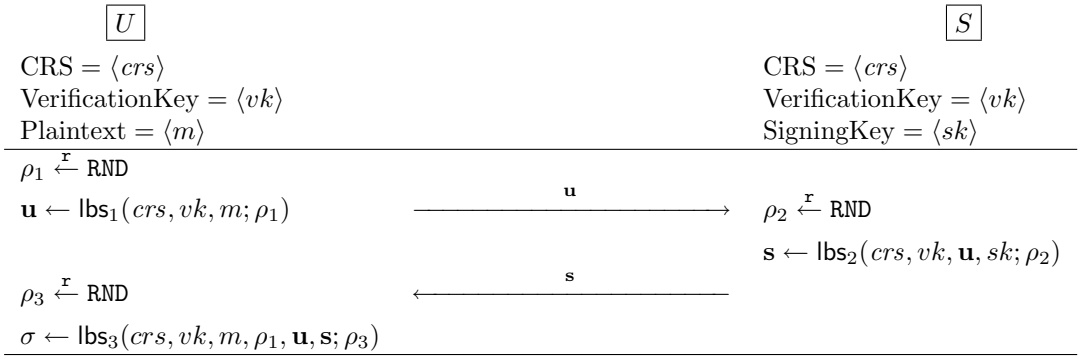

Figure 1: Outline of a two-move signature generation protocol.

Definition 2.1 (Completeness)**.** A signature generation protocol as in Figure 1 is complete if for all $(crs, \tau) \leftarrow \mathtt{CRSgen}(1^\lambda)$, for all $(vk, sk) \leftarrow \mathtt{gen}(crs)$, for all $\rho_1, \rho_2, \rho_3 \xleftarrow{r} \mathtt{RND}$, whenever $\mathbf{u} \leftarrow \mathsf{lbs}_1(crs, vk, m; \rho_1)$, $\mathbf{s} \leftarrow \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2)$, and $\sigma \leftarrow \mathsf{lbs}_3(crs, vk, m, \rho_1, \mathbf{u}, \mathbf{s}; \rho_3)$, then $\mathtt{verify}(crs, vk, m, \sigma) = 1$.

Note that the above completeness is very easy to be generalized to probabilistic case as in [Can05].

Lite-unforgeability that we define below suggests informally that if we "collapse" the $\mathsf{lbs}_1, \mathsf{lbs}_2$ procedures into a single algorithm this will result to a procedure that combined with lbs_3 will be equivalent to the signing algorithm of an unforgeable digital signature $\mathtt{sign}$ in the sense of [GMR88]. We note that lite-unforgeability is much weaker compared to regular unforgeability of blind signatures (as defined e.g., in [Oka06, HKKL07]; refer to Section A.5 in the appendix for this definition) since it requires from the adversary to open the internal tapes of each user instance (as opposed to hiding such internals in the usual unforgeability definition for blind signatures); note that this is not an honest-but-curious modeling as the adversary is not restricted to flip coins honestly.

Definition 2.2 (Lite-unforgeability)**.** A signature generation protocol as in Figure 1 is lite-unforgeable if for all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and for any $L = \mathsf{poly}(\lambda)$, we have $\mathsf{Adv}^{\mathcal{A},L}_{\mathrm{luf}}(\lambda) \leq \mathsf{negl}(\lambda)$, where $\mathsf{Adv}^{\mathcal{A},L}_{\mathrm{luf}}(\lambda) \stackrel{\mathrm{def}}{=} \Pr[\mathbf{Exp}^{\mathrm{LUF}}_{\mathcal{A},L}(\lambda) = 1]$ and the experiment $\mathbf{Exp}^{\mathrm{LUF}}_{\mathcal{A},L}(\lambda)$ is defined below:

Experiment $\mathbf{Exp}^{\mathrm{LUF}}_{\mathcal{A},L}(\lambda)$
  $(crs, \tau) \leftarrow \mathtt{CRSgen}(1^\lambda)$; $(vk, sk) \leftarrow \mathtt{gen}(crs)$; $state := \emptyset$; $k := 0$;
  while $k < L$
    $(m_k, \rho_{1,k}, state) \leftarrow \mathcal{A}_1(state, crs, vk)$;
    $\mathbf{s}_k \leftarrow \mathsf{lbs}_2(crs, vk, \mathsf{lbs}_1(crs, vk, m_k; \rho_{1,k}), sk; \rho_{2,k})$; $\rho_{2,k} \xleftarrow{r} \mathtt{RND}$;
    $state \leftarrow state || \mathbf{s}_k$; $k \leftarrow k + 1$;
  $(m_1, \sigma_1, \ldots, m_\ell, \sigma_\ell) \leftarrow \mathcal{A}_2(state)$;
  if $\ell > L$, and $\mathtt{verify}(crs, vk, m_i, \sigma_i) = 1$ for all $1 \leq i \leq \ell$, and $m_i \neq m_j$ for all $1 \leq i \neq j \leq \ell$
    then return 1 else return 0.

We remark that lite-unforgeability is only required in the standard sense (as in [GMR88]) and not in its strong flavor where the adversary can win the game even if he forges a signature for a message he has already seen [ADR02]. It is straightforward to extend the formulation of lite-unforgeability of blind signatures to capture this stronger flavor of digital signature unforgeability.

Similarly we can formulate blindness (as defined, e.g. in [CNS07]; refer to Section A.5 in the appendix) in the "lite" setting by requiring the adversary to open the private tape of the signer for each user interaction. Given that blindness is subsumed by our equivocality property (defined below), we will not explore this direction further here (the reader may refer to the online full version [KZ07a] for more details). For simplicity we define equivocality only for two-move protocols following the skeleton of Figure 1. Informally an equivocal blind signature scheme is accompanied by a simulator procedure $\mathcal{I}$ which can produce signature generation transcripts without using the user input m and furthermore it can "explain" the transcripts to any adversarially selected m even after the signature σ for m has been generated. The property of equivocal blind signatures parallels the property of equivocal commitments [Bea96] or zero-knowledge with state reconstruction, cf. [GOS06]. We define the property formally below (cf. Figure 2). In the definition, we use the relation $\mathsf{KeyPair}$ defined as $(vk, sk) \in \mathsf{KeyPair}$ if and only if $(vk, sk) \leftarrow \mathtt{gen}(crs)$. Note that we require $(vk, sk), (vk, sk') \in \mathsf{KeyPair}$ to imply $sk = sk'$ (otherwise a blind signature may be susceptible to an attack due to [HK07]).

Definition 2.3 (Equivocality)**.** We say that a signature generation protocol is equivocal if there exists an interactive machine $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2)$, such that for all PPT $\mathcal{A}$, we have $\mathsf{Adv}^{\mathcal{A}}_{\mathrm{eq}}(\lambda) \leq \mathsf{negl}(\lambda)$,

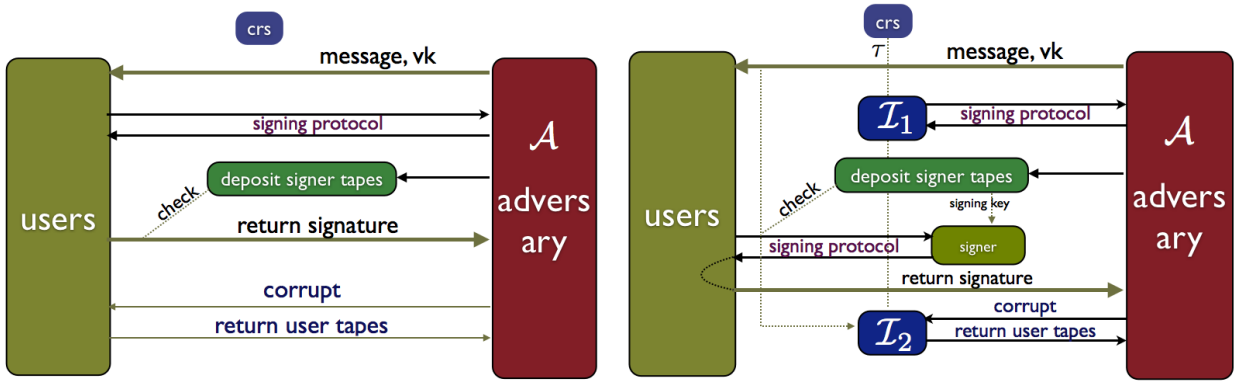

Figure 2: The two worlds an equivocality adversary is asked to distinguish in Definition 2.3. In the left-hand the adversary is interacting with a set of users whereas in the right-hand side the users are interacting with an honest signer instantiation whereas the adversary is interacting with the simulator $\mathcal{I}$.

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{eq}}(\lambda) \stackrel{\mathsf{def}}{=} \left| \begin{array}{r} \Pr[(crs,\tau) \leftarrow \mathtt{CRSgen}(1^\lambda) : \mathcal{A}^{Users(crs,\cdot)}(crs) = 1] \\ -\Pr[(crs,\tau) \leftarrow \mathtt{CRSgen}(1^\lambda) : \mathcal{A}^{\mathcal{I}(crs,\tau,\cdot)}(crs) = 1] \end{array} \right|,$$

where oracle $Users(crs,\cdot)$ operates as:

- Upon receiving message (i,m,vk) from $\mathcal{A}$, select $\rho_1 \xleftarrow{r} \mathtt{RND}$ and compute $\mathbf{u} \leftarrow \mathsf{lbs}_1(crs,vk,m;\rho_1)$, record $\langle i,m,vk,\mathbf{u},\rho_1\rangle$ into $history_i$, and return message $(i,\mathbf{u})$ to $\mathcal{A}$.
- Upon receiving message $(i,\mathbf{s},\rho_2,sk)$ from $\mathcal{A}$, if there exists a record $\langle i,m,vk,\mathbf{u},\rho_1\rangle$ in $history_i$ and both $(vk,sk) \in \mathsf{KeyPair}$ and $\mathbf{s} = \mathsf{lbs}_2(crs,vk,\mathbf{u},sk;\rho_2)$ hold, then select $\rho_3 \xleftarrow{r} \mathtt{RND}$, compute $\sigma \leftarrow \mathsf{lbs}_3(crs,vk,m,\rho_1,\mathbf{u},\mathbf{s};\rho_3)$, update $\langle i,m,vk,\mathbf{u},\rho_1\rangle$ in $history_i$ into $\langle i,m,vk,\mathbf{u},\sigma,\rho_1,\rho_3\rangle$, and return to $\mathcal{A}$ the pair (i,σ); otherwise return to $\mathcal{A}$ the pair $(i,\perp)$.
- Upon receiving message (i,open), return to $\mathcal{A}$ the pair $(i,history_i)$.

and oracle $\mathcal{I}(crs,\tau,\cdot)$ operates as:

- Upon receiving message (i,m,vk) from $\mathcal{A}$, run $(\mathbf{u},aux) \leftarrow \mathcal{I}_1(crs,\tau,vk)$, record $\langle i,m,vk,\mathbf{u},aux\rangle$ into $temp$, and return message $(i,\mathbf{u})$ to $\mathcal{A}$.
- Upon receiving message $(i,\mathbf{s},\rho_2,sk)$ from $\mathcal{A}$, if there exists a record $\langle i,m,vk,\mathbf{u},aux\rangle$ in $temp$ and both $(vk,sk) \in \mathsf{KeyPair}$ and $\mathbf{s} = \mathsf{lbs}_2(crs,vk,\mathbf{u},sk;\rho_2)$ hold, then select $\gamma \xleftarrow{r} \mathtt{RND}$, compute $\sigma \leftarrow \mathtt{sign}(crs,vk,sk,m,\gamma)$ (where $\mathtt{sign}$ is the "collapse" of lbs_i for $i=1,2,3$), update $\langle i,m,vk,\mathbf{u}\rangle$ in $temp$ into $\langle i,m,vk,\mathbf{u},aux;\mathbf{s},sk,\rho_2;\sigma,\gamma\rangle$, and return the pair (i,σ) to $\mathcal{A}$; otherwise return to $\mathcal{A}$ the pair $(i,\perp)$.
- Upon receiving message (i,open), if there exists a record $\langle i,m,vk,\mathbf{u},aux\rangle$ in $temp$ then run $\rho_1 \leftarrow \mathcal{I}_2(i,temp)$, record $\langle i,m,vk,\mathbf{u},\rho_1\rangle$ into $history_i$, and return to $\mathcal{A}$ the pair $(i,history_i)$; if there exists a record $\langle i,m,vk,\mathbf{u},aux;\mathbf{s},sk,\rho_2;\sigma,\gamma\rangle$ in $temp$, then run $(\rho_1,\rho_3) \leftarrow \mathcal{I}_2(i,temp)$, record $\langle i,m,vk,\mathbf{u},\sigma,\rho_1,\rho_3\rangle$ into $history_i$, and return $\mathcal{A}$ with message $(i,history_i)$.

We call a signature generation protocol that satisfies completeness, lite-unforgeability as well as the equivocality property an *equivocal lite blind signature scheme*.

2.2 Constructions

In this subsection, we present two equivocal lite-blind signature constructions. The first construction is generic and is based on the blind signature design of [Fis06] whereas the second is a concrete number theoretic construction that is based on [Oka06]. In the full version of this work [KZ07a] we present additional constructions.

Generic equivocal lite blind signature. Our first construction is based on [Fis06]; the main difference here is that we need the equivocality property (the original design employed two encryption steps for the user that are non-equivocal); in our setting, it is sufficient to have just one equivocal commitment (that is not extractable) in the first stage and then employ an extractable commitment in the second (that is not equivocal). Refer to the signature generation protocol in Figure 3: the $\mathtt{CRSgen}$ algorithm produces $crs = \langle pk_{\text{eqc}}, pk_{\text{exc}}, crs_{\text{nizk}}\rangle$; $\mathtt{EQC}$ is a commitment scheme with committing key pk_{eqc} and $\mathtt{EQCcom}$ is its committing algorithm; $\mathtt{EXC}$ is a commitment scheme with committing key pk_{exc} and $\mathtt{EXCcom}$ is its committing algorithm; $\mathtt{NIZK}$ is an NIZK argument scheme with CRS crs_{nizk} where $\mathtt{NIZKprove}$ is the proof generation algorithm and $\mathtt{NIZKverify}$ is the proof verification algorithm. The $\mathtt{gen}$ algorithm produces a key-pair $\langle vk, sk\rangle$ for a signature scheme $\mathtt{SIG}$ where $\mathtt{SIGsign}$ is the signature generation algorithm and $\mathtt{SIGverify}$ is the corresponding verification algorithm. The language $\mathcal{L}_R \stackrel{\text{def}}{=} \{x | (x,w) \in R\}$ where $R \stackrel{\text{def}}{=} \{(crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \rho_1, \rho_3) \,|\, \mathbf{u} = \mathtt{EQCcom}(pk_{\text{eqc}}, m; \rho_1) \wedge \mathtt{SIGverify}(vk, \mathbf{u}, \mathbf{s}) = 1 \wedge E = \mathtt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)\}$. The $\mathtt{verify}$ algorithm given a message m and signature σ operates as follow: parse σ into E and ϖ, and check that $\mathtt{NIZKverify}((crs, vk, E, m), \varpi) \stackrel{?}{=} 1$.

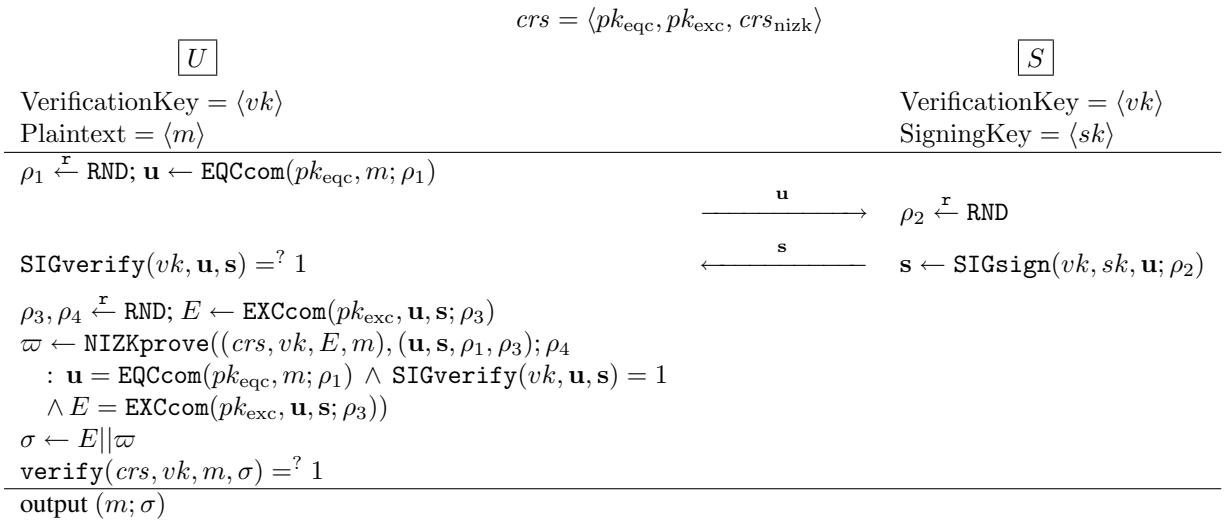

Figure 3: A generic signature generation protocol.

Theorem 2.4. *The two-move signature generation protocol in Figure 3 is an equivocal lite blind signature as follows: it satisfies lite-unforgeability provided that (i)* $\mathtt{SIG}$ *is EU-CMA secure, (ii)* $\mathtt{EQC}$ *is binding, (iii)* $\mathtt{EXC}$ *is extractable, and (iv)* $\mathtt{NIZK}$ *satisfies soundness; and it satisfies equivocality provided that (i)* $\mathtt{EQC}$ *is equivocal, (ii)* $\mathtt{EXC}$ *is hiding, and (iii)* $\mathtt{NIZK}$ *is non-erasure zero-knowledge.*

Concrete number theoretic equivocal lite blind signature. In Figure 4 we present a lite blind signature $\langle \mathtt{CRSgen}, \mathtt{gen}, \mathsf{lbs}_1, \mathsf{lbs}_2, \mathsf{lbs}_3, \mathtt{verify}\rangle$ that uses the 2SDH assumption and is based on Okamoto's blind

signature scheme [Oka06]; the contribution here is our proof (cf. Theorem 2.5 below) that this design is in fact equivocal (instead of merely blind as shown in [Oka06]). In this scheme the `CRSgen` algorithm produces $crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$, where $\hat{\mathsf{e}} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, $\mathbb{G}_1, \mathbb{G}_2$ are groups of order p (refer to Section A.3 for the notations), the `gen` algorithm produces a key-pair $vk = \langle X \rangle$, $sk = \langle x \rangle$ such that $X = g_2^x$, and the `verify` algorithm given a message m and signature $\sigma = \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle$, responds as follows: check that $m, \beta \in \mathbb{Z}_p$, $\varsigma, V_1 \in \mathbb{G}_1$, $\alpha, V_2 \in \mathbb{G}_2$, $\varsigma \neq 1$, $\alpha \neq 1$ and $\hat{\mathsf{e}}(\varsigma, \alpha) = \hat{\mathsf{e}}(g_1, g_2^m u_2 v_2^\beta)$, $\hat{\mathsf{e}}(V_1, \alpha) = \hat{\mathsf{e}}(\psi(X), X) \cdot \hat{\mathsf{e}}(g_1, V_2)$.

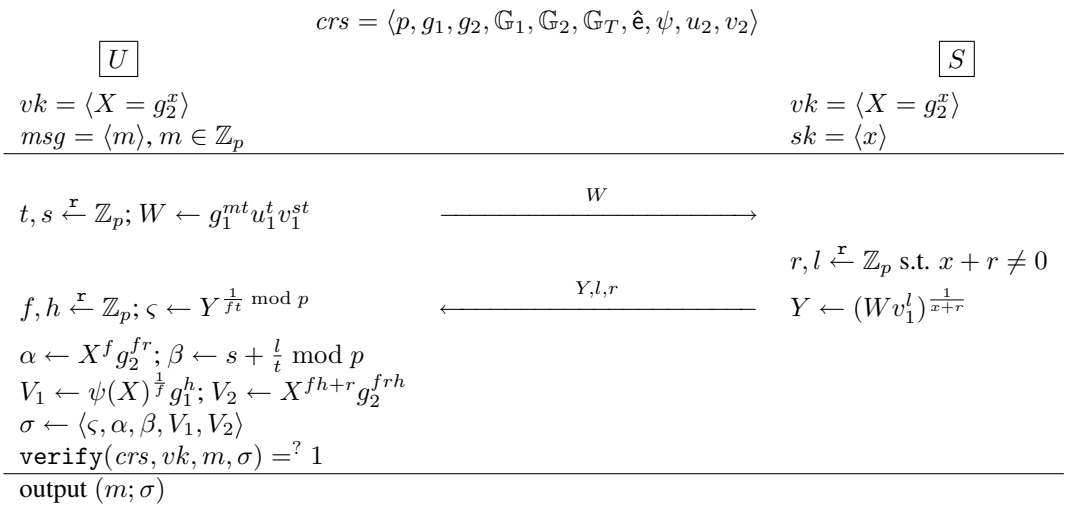

Figure 4: Signature generation protocol based on Okamoto digital signature [Oka06].

Theorem 2.5. *The two-move protocol of Figure 4 is an equivocal lite blind signature as follows: it satisfies lite-unforgeability under the 2SDH assumption; and it satisfies equivocality unconditionally.*

3 Design Methodology for Adaptively Secure UC Blind Signatures

In this section we present our design methodology for constructing UC-blind signatures secure against adaptive adversaries, i.e., the protocol obtained by our method can UC-realize the blind signature functionality $\mathcal{F}_{\text{BSIG}}$ (defined in Figure 5). A previous formalization of the blind signature primitive in the UC setting was given by [Fis06]. In our definition of the ideal functionality we give an explicit description of how corruption is handled; as we deal with adaptive adversaries, being explicit in the way that the ideal functionality interacts with the adversary during corruption makes the security model crisper. One other difference is that our $\mathcal{F}_{\text{BSIG}}$ does not require strong unforgeability from the underlying signing mechanism; this makes the presentation more general as strong unforgeability is not necessary for many applications of the blind signature primitive. Further, protocols realizing the functionality of [Fis06] requires a single "global trapdoor" that enables the functionality to produce a signature for a given message that will be valid for any given public-key; while this can be handy in the security proof, it is not a mandatory requirement for a UC-blind signature (which may allow for a different trapdoor to be used by the functionality in each signature generation); we reflect this in our ideal functionality by allowing the adversary in the corrupted signer setting to "patch" the ideal functionality with a different signing key for each user. In a blind signature session we allow for a single signer (whose identity is hard-coded into the session identifier sid) and a multitude

of users. Each party in a session is executing a "blind-signature program" that is specified in details in the appendix Section B.

Our design is modular and delineates the components required for designing UC blind signatures in the adaptive security setting. We present our methodology in two steps. First, we employ a lite blind signature scheme and we operate in a hybrid world where the following ideal functionalities exist: $\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}$. Here $\mathcal{F}_{\text{CRS}}$ will be an appropriate common reference string functionality; on the other hand, $\mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}$ will be two *different* zero-knowledge functionalities that are variations of the standard multi-session ZK functionality. This reflects the fact that the ZK "needs" of the user and the signer are different in a blind signature. (1) $\mathcal{F}_{\text{SVZK}}^{R_U}$ is the "single-verifier zero-knowledge functionality for the relation R_U" where the user will be the prover and, (2) $\mathcal{F}_{\text{SPZK}}^{R_S}$ is the "single-prover zero-knowledge functionality for the relation R_S" where the signer will be the prover. Due to lack of space both these functionalities are specified in the appendix in Figure 13 and Figure 14 respectively. The two functionalities differ from the multi-session zero-knowledge ideal functionality $\mathcal{F}_{\text{MZK}}$ (e.g., see $\hat{\mathcal{F}}_{\text{ZK}}$ in figure 7, page 49, in [CLOS02]) in the following manner: $\mathcal{F}_{\text{SVZK}}$ assumes that there is only a single verifier that potentially many provers wish to prove to it a certain type of statements; on the other hand, $\mathcal{F}_{\text{SPZK}}$ assumes that only a single prover exists that potentially wishes to convince many verifiers regarding a certain type of statement. Our setting is different from previous UC-formulations of ZK where multiple provers wish to convince multiple verifiers at the same time; while we could use such stronger primitives in our design, recall that we are interested in the simplest possible primitives that can instantiate our methodology as these highlight minimum sufficient requirements for blind signature design in the UC setting.

3.1 Construction in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}, \mathcal{F}_{\text{SPZK}})$-Hybrid World

In this section we describe our blind signature construction in the hybrid world. In Figure 6, we describe a UC blind signature protocol in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid world that is based on an equivocal lite blind signature protocol. The relations parameterized with the ZK functionalities are $R_U = \{((crs, vk, \mathbf{u}), (m, \rho_1)) \mid \mathbf{u} = \mathsf{lbs}_1(crs, vk, m; \rho_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2)) \mid \mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \wedge (vk, sk) \in \mathsf{KeyPair}\}$. We prove the following theorem:

Theorem 3.1. *Given a signature generation protocol that is an equivocal lite blind signature, the protocol $\pi_{\Sigma(\text{BSIG})}$ in Figure 6 securely realizes $\mathcal{F}_{\text{BSIG}}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid model.*

3.2 Realizing $\mathcal{F}_{\text{SVZK}}$ and $\mathcal{F}_{\text{SPZK}}$

In this subsection we focus on the requirements for the UC-realization of the two ZK functionalities $\mathcal{F}_{\text{SVZK}}$ and $\mathcal{F}_{\text{SPZK}}$. We note that they can be instantiated generically based on non-interactive zero-knowledge as in [CLOS02] or [GOS06]. Nevertheless, by focusing on the exact requirements needed for the blind signature setting we manage to get more simplified concrete constructions; note that we will opt for minimizing the overall communication length as opposed to round complexity.

Realizing $\mathcal{F}_{\text{SVZK}}^{R_U}$. The functionality $\mathcal{F}_{\text{SVZK}}^{R_U}$ will be realized against adaptive adversaries. We proceed as follows: first given $(x, w) \in R_U$, we will have the prover commit the witness w into value C; given that we have a single verifier in our setting (the signer), we find it convenient to base the commitment scheme that we employ on the mixed commitment primitive of [DN02, Nie03]; using such a commitment one then can employ a non-erasure Sigma protocol to show the consistency of the witness between the commitment C and the statement x by performing a proof of language membership; finally to defend against a dishonest

Functionality $\mathcal{F}_{\text{BSIG}}$

Key generation: Upon receiving (KEYGEN, sid) from party S, verify that $sid = (S, sid')$ for some sid'. If not, ignore the input. Else, forward (KEYGEN, sid) to the adversary $\mathcal{S}$.

Upon receiving $(\text{ALGORITHMS}, sid, \mathsf{sig}, \mathsf{ver})$ from the adversary $\mathcal{S}$, record $\langle \spadesuit, \mathsf{sig}, \mathsf{ver} \rangle$ in $history(S)$, and output $(\text{VERIFICATIONALG}, sid, \mathsf{ver})$ to party S, where sig is a signing algorithm, and ver is a verification algorithm.

Signature generation: Upon receiving $(\text{SIGN}, sid, m, \mathsf{ver}')$ from party $U \neq S$, where $sid = (S, sid')$, record $\langle m, \mathsf{ver}' \rangle$ in $history(U)$, and send $(\text{SIGN}, sid, U, \mathsf{ver}')$ to the adversary $\mathcal{S}$.

Upon receiving $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerComplete})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, if $\mathsf{ver}' = \mathsf{ver}$ then output $(\text{SIGNSTATUS}, sid, U, \mathsf{completed})$ to party S, and record $\langle U, \mathsf{completed} \rangle$ in $history(S)$. Else if $\mathsf{ver}' \neq \mathsf{ver}$ then halt.

Upon receiving $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerError})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, output $(\text{SIGNSTATUS}, sid, U, \mathsf{incompleted})$ to party S, and record $\langle U, \perp \rangle$ in $history(S)$.

Upon receiving $(\text{SIGNATURE}, sid, U, \mathsf{UserComplete})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature,

- if S is not corrupted and $\langle U, \mathsf{completed} \rangle$ has been recorded in $history(S)$, and $\mathsf{ver}' = \mathsf{ver}$, then compute $\sigma \leftarrow \mathsf{sig}(m, rnd)$ with the required random coins rnd, where sig is the one recorded with $\spadesuit$ in $history(S)$, and do the following: if $\mathsf{ver}(m, \sigma) = 1$ output $(\text{SIGNATURE}, sid, \sigma)$ to party U, and update $history(U)$ into $\langle m, \sigma, rnd, \mathsf{done} \rangle$; else if $\mathsf{ver}(m, \sigma) \neq 1$, halt.
- else if S is corrupted, then compute $\sigma \leftarrow \mathsf{sig}(m, rnd)$ with the required random coins rnd, where sig is the one recorded with U in $history(S)$, and do the following: if $\mathsf{ver}'(m, \sigma) = 1$, output $(\text{SIGNATURE}, sid, \sigma)$ to party U, update $history(U)$ into $\langle m, \sigma, rnd \rangle$; else if $\mathsf{ver}'(m, \sigma) \neq 1$, halt.
- else, halt.

Upon receiving $(\text{SIGNATURE}, sid, U, \mathsf{UserError})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, output $(\text{SIGNATURE}, sid, \perp)$ to party U and update $history(U)$ into $\langle m \rangle$.

Signature verification: Upon receiving $(\text{VERIFY}, sid, m, \sigma, \mathsf{ver}')$ from party V, where $sid = (S, sid')$, do: if $\mathsf{ver}' = \mathsf{ver}$, the signer S is not corrupted, $\mathsf{ver}(m, \sigma) = 1$, and there is no U such that m is recorded with done in $history(U)$, then halt. Else, output $(\text{VERIFIED}, sid, \mathsf{ver}'(m, \sigma))$ to party V.

Corruption: Upon receiving $(\text{CORRUPT}, sid, J)$ from $\mathcal{S}$, mark J with "corrupted" status, and return $(\text{CORRUPTED}, sid, history(J))$ to $\mathcal{S}$. Here J can be party U or party S. Furthermore:

- after receiving $(\text{CORRUPT}, sid, U)$, if S is not corrupted and no $(\text{PATCH}, sid, U, ...)$ message has been received before,
  - if there is no $\langle U, \mathsf{completed} \rangle$ recorded in $history(S)$, upon receiving $(\text{PATCH}, sid, U, \overline{m}, \overline{\mathsf{ver}})$ from the adversary $\mathcal{S}$, then update $history(U)$ into $\langle \overline{m}, \overline{\mathsf{ver}} \rangle$; once a subsequent $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerComplete})$ is received from $\mathcal{S}$, if $\mathsf{ver} = \overline{\mathsf{ver}}$ then update $history(S)$ into $\langle \overline{m}, \mathsf{done} \rangle$, else halt.
  - else if there is $\langle U, \mathsf{completed} \rangle$ recorded in $history(S)$ and $\langle m, \mathsf{ver}' \rangle$ in $history(U)$, then update $history(U)$ into $\langle m, \mathsf{done} \rangle$.
- after receiving $(\text{CORRUPT}, sid, S)$, upon receiving $(\text{PATCH}, sid, S, \overline{\mathsf{sig}}, U)$ from the adversary $\mathcal{S}$, then obtain ver' from $history(U)$; if $\langle \widetilde{U}, \widetilde{\mathsf{sig}}, \mathsf{ver}' \rangle$ has been recorded in $history(S)$ for some party $\widetilde{U}$ where $\widetilde{\mathsf{sig}} \neq \overline{\mathsf{sig}}$, then halt; else record $\langle U, \overline{\mathsf{sig}}, \mathsf{ver}' \rangle$ in $history(S)$.

Figure 5: Blind signature functionality $\mathcal{F}_{\text{BSIG}}$.

Protocol $\pi_{\Sigma(\text{BSIG})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-Hybrid Model

CRS generation: $crs \leftarrow \mathtt{CRSgen}(1^\lambda)$ where λ is the security parameter.

Key generation: When party S is invoked with input (KEYGEN, sid) by $\mathcal{Z}$, it verifies that $sid = (S, sid')$ for some sid'; If not, it ignores the input; Otherwise, it runs $(vk, sk) \leftarrow \mathtt{gen}(crs)$, lets the verification algorithm $\mathsf{ver} := \mathtt{verify}(crs, vk, \cdot, \cdot)$, and sends output $(\text{VERIFICATIONALG}, sid, \mathsf{ver})$ to $\mathcal{Z}$.

Signature generation: On input $(\text{SIGN}, sid, m, \mathsf{ver}')$ by $\mathcal{Z}$ where $sid = (S, sid')$, the party U obtains vk' from ver', selects random ρ_1 and computes $\mathbf{u} \leftarrow \mathsf{lbs}_1(crs, vk', m; \rho_1)$ and sends $(\text{PROVESVZK}, sid, U, (crs, vk', \mathbf{u}), (m, \rho_1))$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$.

Upon receiving $(\text{VERIFIEDSVZK}, sid, U, (crs', vk', \mathbf{u}))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, the party S verifies $crs' = crs$ and $vk' = vk$. If not, then the party S outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{incompleted})$ to $\mathcal{Z}$. Else the party S selects random ρ_2 and computes $\mathbf{s} \leftarrow \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2)$ and sends $(\text{PROVESPZK}, sid, U, (crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2))$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, and outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{completed})$ to $\mathcal{Z}$.

Upon receiving $(\text{VERIFIEDSVZK}, sid, U, \perp)$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, the party S outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{incompleted})$ to $\mathcal{Z}$.

Upon receiving $(\text{VERIFIEDSPZK}, sid, U, (crs', vk'', \mathbf{u}', \mathbf{s}))$ from $\mathcal{F}_{\text{SPZK}}^{R_S}$, the party U verifies that $crs' = crs$ and $vk'' = vk'$ and $\mathbf{u}' = \mathbf{u}$. If not, then party U outputs $(\text{SIGNATURE}, sid, \perp)$ to $\mathcal{Z}$. Else, the party U selects random ρ_3 and computes $\sigma \leftarrow \mathsf{lbs}_3(crs, vk', m, \rho_1, \mathbf{u}, \mathbf{s}; \rho_3)$, and outputs $(\text{SIGNATURE}, sid, \sigma)$ to $\mathcal{Z}$.

Upon receiving $(\text{VERIFIEDSPZK}, sid, U, \perp)$ from $\mathcal{F}_{\text{SPZK}}^{R_S}$, the party U outputs $(\text{SIGNATURE}, sid, \perp)$ to $\mathcal{Z}$.

Signature verification: When party V is invoked with input $(\text{VERIFY}, sid, m, \sigma, \mathsf{ver}')$ by $\mathcal{Z}$ where $sid = (S, sid')$, it outputs $(\text{VERIFIED}, sid, \mathsf{ver}'(m, \sigma))$ to $\mathcal{Z}$.

Corruption: When party J is invoked with incoming $(\text{CORRUPT}, sid, J)$ by $\mathcal{Z}$, it sends outgoing $(\text{CORRUPTED}, sid, history(J))$ to $\mathcal{Z}$. Here J can be party U or party S.

Figure 6: Blind signature protocol $\pi_{\Sigma(\text{BSIG})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid model based on a lite-blind signature scheme $\langle \mathtt{CRSgen}, \mathtt{gen}, \mathsf{lbs}_1, \mathsf{lbs}_2, \mathsf{lbs}_3, \mathtt{verify} \rangle$. Here functionalities $\mathcal{F}_{\text{SVZK}}^{R_U}$ and $\mathcal{F}_{\text{SPZK}}^{R_S}$ are parameterized with relations $R_U = \{((crs, vk, \mathbf{u}), (m, \rho_1)) \mid \mathbf{u} = \mathsf{lbs}_1(crs, m; \rho_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2)) \mid \mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \wedge (vk, sk) \in \mathsf{KeyPair}\}$, respectively.

verifier, the Sigma protocol will have to be strengthened so that it can be simulated without knowing the witness; this e.g., can be based on Damgård's trick [Dam00].

Based on the above we obtain an efficient number-theoretic instantiation of the functionality that is secure under the Decisional Composite Residuosity assumption of Paillier [Pai99] (the construction can be found in the appendix, Figure 16). The underlying mixed-commitment is based on Damgård-Jurik encryption [DJ01]; it could be also based on other encryption schemes as well.

Remark 3.2. Depending on the properties of the statement x that is proven, we remark that it is possible to simplify the implementation of $\mathcal{F}_{\text{SVZK}}$. In particular, if x includes a commitment of the witness w that is equivocal based on the given crs, then the relevant commitment that is made to w by the prover in the very first communication flow of the protocol (as e.g., in Figure 15 in the appendix) is unnecessary. In this case, that commitment may be dropped entirely from the realization as in the security proof x itself can be used instead. Note that taking advantage of such modification is done only for the sake of efficiency of the overall protocol (and in fact we will employ it in our concrete protocol instantiation in Section 3.3).

Realizing $\mathcal{F}_{\text{SPZK}}^{R_S}$. Regarding $\mathcal{F}_{\text{SPZK}}^{R_S}$ we find that, rather surprisingly, our task for attaining an adaptive secure UC blind signature is simpler since security against a static adversary suffices. The reason is that in the UC blind signature security proof, the simulator knows the signing secret which means the witness for $\mathcal{F}_{\text{SPZK}}^{R_S}$ is known by the simulator, and thus no equivocation of dishonestly simulated transcripts is ever necessary! This behavior was explored by the authors in the context of zero-knowledge in [KZ07b]; in the framework of that paper, we can say a blind signature protocol falls into the class of protocols where a leaking version of $\mathcal{F}_{\text{SPZK}}^{R_S}$ is sufficient for security and thus $\mathcal{F}_{\text{SPZK}}^{R_S}$ need be realized only against static adversaries.

Similarly to the realization of $\mathcal{F}_{\text{SVZK}}^{R_U}$, for $(x, w) \in R_S$, we have the prover commit to the witness w into the value C, but here we only need employ an extractable commitment considering we only need to realize $\mathcal{F}_{\text{SPZK}}^{R_S}$ against static adversaries; then we develop a Sigma protocol to show the consistency between the commitment C and the statement x by performing a proof of language membership; the first two steps together can be viewed as an Ω-protocol in [GMY06]; further we need to wrap up such Ω-protocol by applying e.g., Damgård's trick to defend against dishonest verifiers. We present an instantiation of $\mathcal{F}_{\text{SPZK}}$ against static adversaries that reflects the above points in Figure 18 in the appendix, where we use an extractable commitment $\texttt{EXC}$ and a Sigma protocol for the consistency of the relation R_S together with the extractable commitment. Note that here the underlying Sigma protocol is not necessary to be non-erasure (as it was the case in our realization of $\mathcal{F}_{\text{SVZK}}$).

3.3 Concrete Construction

In this section, we demonstrate how it is possible to derive an efficient UC blind signature instantiation based on Theorem 3.1 and the realization of its hybrid world with the related ZK-functionalities. Note that we opt for minimizing the overall communication complexity as opposed to round complexity. We need three ingredients: (1) an equivocal lite blind signature scheme, (2) a UC-realization of the ideal functionality $\mathcal{F}_{\text{SVZK}}^{R_U}$, (3) a UC-realization of the ideal functionality $\mathcal{F}_{\text{SPZK}}^{R_S}$. Regarding (1) we will employ the equivocal lite blind signature scheme of Figure 4. Regarding the two ZK functionalities we will follow the design strategy outlined in the previous subsection. Recall that in Figure 6, $R_U = \{((crs, vk, \mathbf{u}), (m, \rho_1)) \mid \mathbf{u} = \mathsf{lbs}_1(crs, m; \rho_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2)) \mid \mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \wedge (vk, sk) \in \mathsf{KeyPair}\}$. Instantiating these relations for the protocol of Figure 4 we have that $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$ and $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (W v_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$.

Efficient instantiation of $\mathcal{F}_{\text{SVZK}}^{R_U}$. Based on the SVZK protocol discussed in the previous subsection (presented in Figure 16 of the appendix), and considering the underlying equivocal lite blind signature in Figure 4, we can realize $\mathcal{F}_{\text{SVZK}}^{R_U}$ efficiently. We instantiate the equivocal commitment $\texttt{COM}$ with a hashed Pedersen commitment [Ped91]. For the relation R_U, it is easy to derive an efficient Sigma protocol Σ^{R_U} (the reader may refer to Figure 19 in the appendix). In the resulting construction, the value W that is present within the statement proved by the user can play the role of an equivocal commitment and thus, as discussed in Remark 3.2, we can take advantage of this fact and save somewhat in the communication complexity (this is reflected in the full protocol description that is presented in the appendix Figure 22 and Figure 23). The resulting protocol consists of 5 moves (two moves for building the mixed commitment and 3 moves for the Sigma protocol).

Efficient instantiation of $\mathcal{F}_{\text{SPZK}}^{R_S}$. Based on the SPZK protocol discussed in the previous subsection (presented in Figure 18 of the appendix) we instantiate the extractable commitment by a Paillier encryption [Pai99]; an efficient Sigma protocol Σ^{R_S} for relation R_S is easy to be constructed (we present such protocol explicitly in Figure 20 in the appendix); the required equivocal commitment that is needed to employ

Damgård's trick [Dam00] is instantiated with a hashed Pedersen commitment. The resulting protocol requires 3 moves.

Communication rounds optimization. Based on the protocols put forth above, we can obtain an 8 moves blind signature protocol by having the user make the first proof and the signer respond (refer to Figure 7). Nevertheless, it is possible to achieve a 6 moves protocol by carefully interleaving the communication transcripts of the two sides (refer to Figure 8). This is possible by modifying the UC protocol that realizes $\mathcal{F}_{\text{SPZK}}$ in the following manner: the signer (who executes $\mathcal{F}_{\text{SPZK}}$) starts the proof prior to receiving the final communication of the user's protocol; this naturally puts the signer at risk as Theorem 3.1 does not apply directly anymore (the signer starts the proof prior to the user completing her part). We mitigate the problem by modifying the signer's ZK-protocol so that it leaks no information in the first move; this enables an early start for the signer; the trick relies on a commitment that is performed by the signer in the first step. The resulting protocol is also a realization of $\mathcal{F}_{\text{SPZK}}$ (and is presented in the appendix, Figure 21). This trick may be of general interest as it has the potential of reducing the number of rounds when two parties are bilaterally proving to each other statements in zero-knowledge.

For the final 6-move blind signature protocol, please refer to the appendix, Figure 22 and Figure 23.

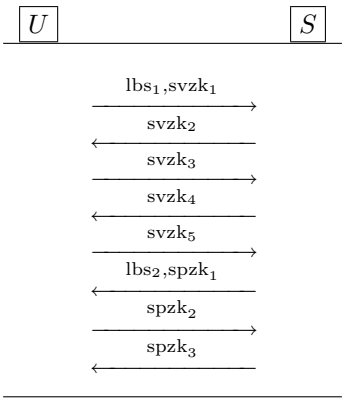

Figure 7: 8-move two side zero-knowledge proving for lite blind signature, SVZK, and SPZK.

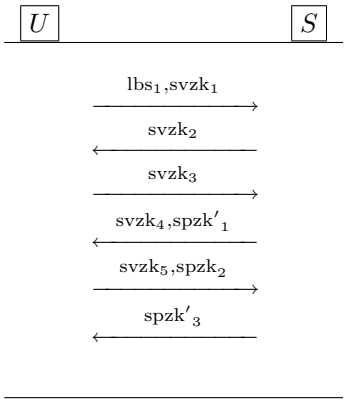

Figure 8: 6-move two side zero-knowledge proving for lite blind signature, SVZK, and SPZK. Here lbs_2 is committed inside spzk'_1, and will be opened until spzk'_3; except this difference, spzk'_1 and spzk'_3 are same as spzk_1, spzk_3 in Figure 7.

Finally, we can obtain the corollary below:

Corollary 3.3. *Under the DCR assumption, the DLOG assumption, and the 2SDH assumption, and assuming existence of collision resistant hash function, there exists a blind signature protocol that securely realizes $\mathcal{F}_{\text{BSIG}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model.*

References

[Abe01] Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 136–151. Springer, 2001.

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.

[ANN06] Michel Abdalla, Chanathip Namprempre, and Gregory Neven. On the (im)possibility of blind message authentication codes. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2006.

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2000.

[AO01] Masayuki Abe and Miyako Ohkubo. Provably secure fair blind signatures with tight revocation. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 583–602. Springer, 2001.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation (extended abstract). In *STOC 1996*, pages 629–638. ACM, 1996.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, 2004.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003. The preliminary version entitled as "The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme" appeared in Financial Cryptography 2001, Springer-Verlag(LNCS 2339).

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

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

[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/.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO 1982*, pages 199–203. Plemum Press, 1982.

[Che06] Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In *EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006.

[CKW04] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient blind signatures without random oracles. In Carlo Blundo and Stelvio Cimato, editors, *SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2004.

[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.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 573–590. Springer, 2007.

[Dam88] Ivan Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In Shafi Goldwasser, editor, *CRYPTO 1988*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer, 1988.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 418–430. Springer, 2000.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596. Springer, 2002. Full version at http://www.brics.dk/RS/01/41/BRICS-RS-01-41.pdf.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer-Verlag, 2006. Full version at http://www.minicrypt.cdc.informatik.tu-darmstadt.de/publications/fischlin.blind-sigs.2006.pdf.

[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.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *J. Cryptology*, 19(2):169–209, 2006. An extended abstract appeared in Eurocrypt 2003, Springer-Verlag (LNCS 2656), pages 177-194, 2003.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006. Full version at http://www.brics.dk/~jg/NIZKJournal3.pdf.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2007.

[HKKL07] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 1997.

[Kim04] Kwangjo Kim. Lessons from Internet voting during 2002 FIFA WorldCup Korea/Japan(TM). In *DIMACS Workshop on Electronic Voting – Theory and Practice*, 2004.

[KZ06] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent blind signatures without random oracles. In Roberto De Prisco and Moti Yung, editors, *SCN 2006*, volume 4116 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2006. http://eprint.iacr.org/2005/435/.

[KZ07a] Aggelos Kiayias and Hong-Sheng Zhou. Equivocal blind signatures and adaptive UC-security. In *Cryptology ePrint Archive: Report 2007/132*, 2007. Full version.

[KZ07b] Aggelos Kiayias and Hong-Sheng Zhou. Trading static for adaptive security in universally composable zero-knowledge. In *ICALP 2007*, 2007. To appear.

[Lin03] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC 2003*, pages 683–692. ACM, 2003. Full version at http://www.cs.biu.ac.il/~lindell/PAPERS/conc2party-upper.ps.

[Nie03] Jesper Buus Nielsen. On protocol security in the cryptographic model. *Dissertation Series DS-03-8, BRICS*, 2003. http://www.brics.dk/DS/03/8/BRICS-DS-03-8.pdf.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1992.

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99. Springer, 2006. An extended version at http://eprint.iacr.org/2006/102/.

[OO89] Tatsuaki Okamoto and Kazuo Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT 1989*, volume 434 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1989.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[Poi98] David Pointcheval. Strengthened security for blind signatures. In Kaisa Nyberg, editor, *EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 1998.

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT 1996*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 1996.

[PS97] David Pointcheval and Jacques Stern. New blind signatures equivalent to factorization (extended abstract). In *CCS 1997*, pages 92–99. ACM, 1997.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.

[PS05] Manoj Prabhakaran and Amit Sahai. Relaxing environmental security: Monitored functionalities and client-server computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 104–127. Springer, 2005.

[Sho04] Victor Shoup. Sequences of games: A tool for taming complexity in security proofs. 2004. http://shoup.net/papers/games.pdf.

A Preliminaries

A.1 The Universal Composibility Framework

Defining the security in the universal composibility framework includes the following steps: we first specify an ideal functionality, which describes the desired behavior of the protocol by using a trusted party; then we prove that a particular protocol operating in the real world securely realizes this ideal functionality. Below, we give a brief description of the framework. See [Can05] for more details.

Authenticated communication. We assume an asynchronous, authenticated, public network, without guaranteed delivery of messages. More precisely, the adversary is allowed to delay a message indefinitely, and to change the contents of the message, as long as the sender is corrupted at the time of delivery (even if the sender was uncorrupted at the time of transmission).

Corruptions strategy. There are two corruption strategies, static corruption and adaptive corruption. In the static case, the adversary corrupts parties only at the onset of the computation; in the adaptive case, the adversary chooses which parties to corrupt as the computation evolves. Once the adversary corrupts a party, it learns all its internal information, including the private input, the communication history, and the random bits used. Once they are corrupted, the behavior of the parties is arbitrary.

The real-world model. The real-world model is defined as a system of interactive Turing machines (ITMs) including the execution of a protocol π, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$ with input z, where the adversary represents all adversarial activities against the protocol execution, and the environment represents all other protocols instances and adversaries. The environment $\mathcal{Z}$ is activated first, then the adversary $\mathcal{A}$ is invoked by $\mathcal{Z}$. The parties in an instance of protocol π can be invoked by $\mathcal{Z}$ with an input message, or by $\mathcal{A}$ with an incoming communication message. Once the adversary is activated, it may deliver a message to some party by writing this message to the party's incoming communication tape, or corrupt a party if the environment allows it, or report some information to $\mathcal{Z}$. Once a party is activated, it follows its code and possibly writes outputs on the subroutine output tape of $\mathcal{Z}$, or writes outgoing messages on the incoming communication tape of $\mathcal{A}$, or invokes other ITM instances as subroutines by providing inputs to them and receiving outputs from them. Finally, the environment will output one bit and halt. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble of random variables describing $\mathcal{Z}$'s output when interacting with adversary $\mathcal{A}$ and parties running protocol π, on a security parameter λ, assuming uniformly-chosen random tapes for all entities.

The ideal-world model. Security of protocols is defined via comparing the real-world execution to an ideal-world process. We introduce an ideal functionality $\mathcal{F}$ into the ideal-world process to capture the desired functionality of the given task. The ideal-world process involves an ideal functionality $\mathcal{F}$, an ideal-world adversary (also known as the simulator) $\mathcal{S}$, an environment $\mathcal{Z}$ with input z, and a set of dummy parties. The ideal functionality can be seen as a "joint subroutine" of the dummy parties. As in the real model, the environment $\mathcal{Z}$ is always activated first, and then activates either the adversary $\mathcal{S}$ or some dummy party by writing an input. If the simulator $\mathcal{S}$ is activated, then it activates the ideal functionality $\mathcal{F}$ by delivering a message, or reports some information to $\mathcal{Z}$. When a dummy party is activated by an input message from $\mathcal{Z}$, it forwards the input message to $\mathcal{F}$. Based on its program and the inputs, $\mathcal{F}$ generates its outputs. Corruption of parties is captured as a request from the simulator $\mathcal{S}$ to the functionality $\mathcal{F}$. Let $\text{EXEC}^{\mathcal{F}}_{\pi_d,\mathcal{S},\mathcal{Z}}$ denote the ensemble of random variables describing $\mathcal{Z}$'s output after interacting with $\mathcal{S}$ and $\mathcal{F}$, on a security parameter λ, and assuming uniformly-chosen random tapes for all entities.

Securely realizing an ideal functionality. In the UC framework, a protocol π securely realizes an ideal functionality $\mathcal{F}$ if for any real-world adversary $\mathcal{A}$ there exists an ideal-world simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π in the real world, or with $\mathcal{S}$, $\mathcal{F}$ and dummy parties in the ideal world. More precisely, the

two distribution ensembles are indistinguishable, i.e. $\text{EXEC}^{\mathcal{F}}_{\pi_d,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.
Security with respect to the dummy adversary. Instead of quantifying over all possible adversaries $\mathcal{A}$, Canetti [Can05] shows that it suffices to require the simulator $\mathcal{S}$ to simulate a very simple adversary, called "dummy adversary", for any $\mathcal{Z}$. Here dummy adversary just delivers the messages between the environment $\mathcal{Z}$ and the parties. Now $\mathcal{Z}$ fully controls over the communication. We will ignore $\mathcal{A}$ and let $\text{EXEC}_{\pi,\mathcal{Z}}$ denote the ensemble of random variables describing $\mathcal{Z}$'s output when $\mathcal{A}$ is dummy.

The hybrid model. The hybrid model with a functionality $\mathcal{F}$ is similar to the real-world model, with the addition that the parties may invoke an unbounded number of $\mathcal{F}$ subroutines. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID). Let $\text{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble of random variables describing the output of $\mathcal{Z}$, after interacting with $\mathcal{A}$ and parties running protocol π in the $\mathcal{F}$-hybrid model. Assume now that protocol ϱ securely realizes $\mathcal{F}$. The composed protocol π^{ϱ} is constructed by replacing the first input to $\mathcal{F}$ in π by an invocation of a new copy of ϱ, with fresh random tapes, the same SID, and with the contents of that message as input; each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ϱ, with the contents of that message as new input to ϱ.
The composition theorem. In its general form, the composition theorem basically says that if ϱ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model for some functionality $\mathcal{G}$, then an execution of the composed protocol π^{ϱ}, running in the $\mathcal{G}$-hybrid model, "emulates" an execution of protocol π in the $\mathcal{F}$-hybrid model. That is, for any adversary $\mathcal{A}$ in the $\mathcal{G}$-hybrid model there exists an adversary $\mathcal{S}$ in the $\mathcal{F}$-hybrid model such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and π^{ϱ} in the $\mathcal{G}$-hybrid model or it is interacting with $\mathcal{S}$ and π in the $\mathcal{F}$-hybrid model, i.e. $\text{EXEC}^{\mathcal{G}}_{\pi^{\varrho},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}$.

A.2 Signature Scheme $\Sigma(\text{SIG})$ and Signature Functionality $\mathcal{F}_{\text{SIG}}$

Goldwasser et al. [GMR88] first introduced the security notion of existential forgeability against chosen message attacks (EU-CMA), for digital signatures.

Definition A.1 (EU-CMA Signature Schemes)**.** A signature scheme $\Sigma(\text{SIG}) = \langle \texttt{gen}, \texttt{sign}, \texttt{verify} \rangle$ is called EU-CMA if the following properties hold for any negligible function $\texttt{negl}(\cdot)$, and all large enough values of the security parameter λ,
Completeness: For any message $m \in \mathcal{M}$,

$$\Pr[(vk, sk) \leftarrow \texttt{gen}(1^{\lambda}); rnd \xleftarrow{r} \texttt{RND}; \sigma \leftarrow \texttt{sign}(vk, sk, m, rnd); 0 \leftarrow \texttt{verify}(vk, m, \sigma)] \leq \texttt{negl}(\lambda).$$

Consistency: For any $m \in \mathcal{M}$, the probability that $\texttt{gen}(1^{\lambda})$ generates $\langle vk, sk \rangle$ and $\texttt{verify}(vk, m, \sigma)$ generates two different outputs in two independent invocations is smaller than $\texttt{negl}(\lambda)$.
Unforgeability: For any PPT forger F,

$$\begin{aligned}\Pr[&(vk, sk) \leftarrow \texttt{gen}(1^{\lambda}); (m, \sigma) \leftarrow F^{\texttt{sign}(vk,sk,\cdot,\cdot)}(vk); \\ &1 \leftarrow \texttt{verify}(vk, m, \sigma) \text{ and } F \text{ never asked } \texttt{sign}(vk, sk, \cdot, \cdot) \text{ to sign } m] \leq \texttt{negl}(\lambda).\end{aligned}$$

Canetti[Can05] defines the signature functionality $\mathcal{F}_{\text{SIG}}$ in Figure 9 and proves the theorem below, where signature protocol $\pi_{\Sigma(\text{SIG})}$ presented in Figure 10 is transformed from $\Sigma(\text{SIG})$.

Theorem A.2. *$\Sigma(\text{SIG})$ is EU-CMA $\Leftrightarrow$ $\pi_{\Sigma(\text{SIG})}$ securely realizes $\mathcal{F}_{\text{SIG}}$.*

Functionality $\mathcal{F}_{\text{SIG}}$

Key generation: Upon receiving (KEYGEN, sid) from party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the input. Else, forward (KEYGEN, sid) to the adversary $\mathcal{S}$.

Upon receiving $(\text{ALGORITHMS}, sid, \mathsf{sig}, \mathsf{ver})$ from the adversary $\mathcal{S}$, record $\langle \mathsf{sig}, \mathsf{ver} \rangle$ in $history(S)$ and output $(\text{VERIFICATIONALG}, sid, \mathsf{ver})$ to party S, where sig is a signing algorithm, and ver is a verification algorithm.

Signature generation: Upon receiving (SIGN, sid, m) from party S where $sid = (S, sid')$, let $\sigma = \mathsf{sig}(m, rnd)$ where some random coins rnd may be used, and verify that $\mathsf{ver}(m, \sigma) = 1$. If so, then output $(\text{SIGNATURE}, sid, \sigma)$ to party S, and record $\langle m, \sigma, rnd \rangle$ into $history(S)$. Else, halt.

Signature verification: Upon receiving $(\text{VERIFY}, sid, m, \sigma, \mathsf{ver}')$ from party V, where $sid = (S, sid')$, do: if $\mathsf{ver}' = \mathsf{ver}$, the signer S is not corrupted, $\mathsf{ver}(m, \sigma) = 1$, and m is not recorded, then halt. Else, output $(\text{VERIFIED}, sid, \mathsf{ver}'(m, \sigma))$ to party V.

Corruption: Upon receiving $(\text{CORRUPT}, sid, J)$ from the adversary $\mathcal{S}$, return $(\text{CORRUPTED}, sid, history(J))$ to $\mathcal{S}$.

Figure 9: Signature functionality $\mathcal{F}_{\text{SIG}}$.

A.3 Bilinear Groups

Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups of prime order p so that (i) $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$; (ii) $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ is an isomorphism with $\psi(g_2) = g_1$ and (iii) $\hat{\mathsf{e}} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. We remark that in some cases it can be that $\mathbb{G}_1 = \mathbb{G}_2$ (and in this case ψ_2 would be the identity mapping). Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ groups as above with $|\mathbb{G}_1| = |\mathbb{G}_2| = p$; a bilinear map is a map $\hat{\mathsf{e}}$ s.t. for all $(u, v) \in \mathbb{G}_1 \times \mathbb{G}_2$ it holds that $\hat{\mathsf{e}}(a^x, b^y) = \hat{\mathsf{e}}(a, b)^{xy}$ and $\hat{\mathsf{e}}(g_1, g_2) \neq 1$.

A.4 Okamoto Signature

In our construction, we will use the signature recently proposed in section 5 in [Oka06], which is based on bilinear groups, and is EU-CMA secure under q-2SDH assumption.

Definition A.3 (2-Variable Strong Diffie-Hellman (2SDH) Assumption)**.** Let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups defined as above. The q-2SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: given a $(3q+4)$-tuple $\langle g_1, g_2, w_2 \leftarrow g_2^x, u_2 \leftarrow g_2^y, g_2^{\frac{y+b_1}{x+a_1}}, \ldots, g_2^{\frac{y+b_q}{x+a_q}}, a_1, \ldots, a_q, b_1, \ldots, b_q \rangle$ as input, output $\langle \varsigma \leftarrow g_1^{\frac{y+d}{fx+c}}, \alpha \leftarrow g_2^{fx+r}, d, V_1, V_2 \rangle$ where $a_1, \ldots, a_q, b_1, \ldots, b_q, d, f, r \in \mathbb{Z}_p$; $w_1 \leftarrow \psi(w_2), \varsigma, V_1 \in \mathbb{G}_1$; $\alpha, V_2 \in \mathbb{G}_2$; and $\hat{\mathsf{e}}(\varsigma, \alpha) = \hat{\mathsf{e}}(g_1, u_2 g_2^d)$, $\hat{\mathsf{e}}(V_1, \alpha) = \hat{\mathsf{e}}(w_1, w_2) \cdot \hat{\mathsf{e}}(g_1, V_2)$, $d \notin \{b_1, \ldots, b_q\}$. The q-2SDH assumption suggests that any PPT algorithm $\mathcal{A}$ solving the q-2SDH problem has negligible success probability, i.e.

$$\mathsf{Adv}^{\mathcal{A}}_{\text{2sdh}} \stackrel{\text{def}}{=} \left| \begin{array}{l} x, y \in \mathbb{Z}_p; g_2 \in \mathbb{G}_2; g_1 \leftarrow \psi(g_2); w_2 \leftarrow g_2^x; u_2 \leftarrow g_2^y; \\ a_1, \ldots, a_q, b_1, \ldots, b_q \in \mathbb{Z}_p; c_1 \leftarrow g_2^{\frac{y+b_1}{x+a_1}}, \ldots, c_q \leftarrow g_2^{\frac{y+b_q}{x+a_q}}; \\ (\varsigma, \alpha, d, V_1, V_2) \leftarrow \mathcal{A}(g_1, g_2, w_2, u_2, a_1, \ldots, a_q, b_1, \ldots, b_q, c_1, \ldots, c_q) : \\ \quad \hat{\mathsf{e}}(\varsigma, \alpha) = \hat{\mathsf{e}}(g_1, u_2 g_2^d) \wedge \hat{\mathsf{e}}(V_1, \alpha) = \hat{\mathsf{e}}(w_1, w_2) \cdot \hat{\mathsf{e}}(g_1, V_2) \wedge d \notin \{b_1, \ldots, b_q\} \end{array} \right| \leq \mathsf{negl}(\lambda).$$

Protocol $\pi_{\Sigma(\mathrm{SIG})}$

Key generation: When party S is invoked with $(\textsc{KeyGen}, sid)$ by $\mathcal{Z}$, it verifies that $sid = (S, sid')$ for some sid'; If not, it ignores the input; Otherwise, it runs $(vk, sk) \leftarrow \mathtt{gen}(1^\lambda)$, lets the signing algorithm $\mathsf{sig} := \mathtt{sign}(vk, sk, \cdot, \cdot)$ and the verification algorithm $\mathsf{ver} := \mathtt{verify}(vk, \cdot, \cdot)$, and outputs $(\textsc{VerificationAlg}, sid, \mathsf{ver})$ to $\mathcal{Z}$.

Signature generation: When party S is invoked with $(\textsc{Sign}, sid, m)$ by $\mathcal{Z}$ where $sid = (S, sid')$, it sets $\sigma \leftarrow \mathsf{sig}(m, rnd)$ where some random coins rnd may be used, and outputs $(\textsc{Signature}, sid, \sigma)$ to $\mathcal{Z}$.

Signature verification: When party V is invoked with $(\textsc{Verify}, sid, m, \sigma, \mathsf{ver}')$ by $\mathcal{Z}$ where $sid = (S, sid')$, it outputs $(\textsc{Verified}, sid, \mathsf{ver}'(m, \sigma))$ to $\mathcal{Z}$.

Corruption: When party J is invoked with $(\textsc{Corrupt}, sid, J)$ by $\mathcal{Z}$, it returns $(\textsc{Corrupted}, sid, history(J))$ to $\mathcal{Z}$.

Figure 10: Signature protocol $\pi_{\Sigma(\mathrm{SIG})}$.

Next we briefly introduce Okamoto signature.

Key generation: Randomly select $g_2, u_2, v_2 \xleftarrow{\mathtt{r}} \mathbb{G}_2$ and set $g_1 \leftarrow \psi(g_2)$, $u_1 \leftarrow \psi(u_2)$ and $v_1 \leftarrow \psi(v_2)$. Randomly select $x \xleftarrow{\mathtt{r}} \mathbb{Z}_p$ and compute $X \leftarrow g_2^x \in \mathbb{G}_2$. Set $vk = \langle g_1, g_2, u_2, v_2, X\rangle$ and secret key $sk = \langle x\rangle$.

Signature generation: Let $m \in \mathbb{Z}_p$ be the message to be signed. Signer S randomly selects r and s from $\mathbb{Z}_p$ s.t. $x + r \not\equiv 0 \bmod p$; and compute $\varsigma \leftarrow (g_1^m u_1 v_1^s)^{\frac{1}{fx+r}}$, $\alpha \leftarrow g_2^{fx+r}$, $V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h$, $V_2 \leftarrow X^{fh+\frac{r}{f}} g_2^{rh}$, where $f, r, s, h \xleftarrow{\mathtt{r}} \mathbb{Z}_p$. The signature for m is $\sigma = \langle \varsigma, \alpha, s, V_1, V_2\rangle$.

Signature verification: Given verification key $vk = \langle g_1, g_2, u_2, v_2, X\rangle$, message m, and signature $\sigma = \langle \varsigma, \alpha, s, V_1, V_2\rangle$, check that $m, s \in \mathbb{Z}_p$, $\varsigma, V_1 \in \mathbb{G}_1$, $\alpha, V_2 \in \mathbb{G}_2$, $\varsigma \neq 1$, $\alpha \neq 1$ and $\hat{\mathsf{e}}(\varsigma, \alpha) = \hat{\mathsf{e}}(g_1, g_2^m u_2 v_2^s)$, $\hat{\mathsf{e}}(V_1, \alpha) = \hat{\mathsf{e}}(\psi(X), X) \cdot \hat{\mathsf{e}}(g_1, V_2)$. If they hold, then the verification is valid; otherwise invalid.

A.5 Blind Signature Schemes

Definition A.4 (Blind Signature Schemes)**.** A blind digital signature scheme $\Sigma(\mathrm{BSIG})$ is a five-tuple, consisting of two interactive Turing machines S, U, where S denotes the signer, and U the user, and three algorithms $\mathtt{CRSgen}, \mathtt{gen}, \mathtt{verify}$ as follows:

- $\mathtt{CRSgen}(1^\lambda)$ is a probabilistic polynomial time CRS generation algorithm which takes as an input a security parameter λ and outputs a pair $\langle crs, \tau\rangle$ of CRS and its trapdoor.

- $\mathtt{gen}(crs)$ is a probabilistic polynomial time key generation algorithm which takes as an input crs and outputs a pair (vk, sk) of public and secret keys.

- S and U are a pair of probabilistic interactive Turing machines with the following tapes: a read-only public input tape, a read-only secret input tape, a write-only public output tape, a write-only secret output tape, a read/write secret work tape, a read-only secret random tape, and incoming/outgoing communication tapes. They are both given crs, vk on their public input tapes. Additionally S is given sk on his secret input tape and U is given message m on his secret input tape, where the length of all inputs must be polynomial in the security parameter λ. Both U and S engage in the interactive

protocol of some polynomial number of rounds. At the end of this protocol S outputs either completed or incompleted and U outputs either σ or $\perp$.

- $\texttt{verify}(crs, vk, m, \sigma)$ is a deterministic polynomial time algorithm, which outputs 1 or 0.

Remark A.5. Note that in the plain model, $\texttt{CRSgen}$ is not employed, and the blind signature scheme $\Sigma(\text{BSIG})$ consists of $\langle \texttt{gen}, S, U, \texttt{verify} \rangle$.

A signature scheme is naturally defined by a blind signature when party S and party U are same, i.e. $S = U$; in this case, the scheme $\Sigma(\text{BSIG})$ can collapse to a plain signature scheme $\Sigma(\text{SIG}) = \langle \texttt{gen}, \texttt{sign}, \texttt{verify} \rangle$. Here $\texttt{gen}$ and $\texttt{verify}$ are same as that defined in $\Sigma(\text{BSIG})$, and $\texttt{sign}$ is the signature generation algorithm which can be used to generate signature σ for m: $\sigma \leftarrow \texttt{sign}(vk, sk, m, rnd)$ where $rnd \xleftarrow{r} \texttt{RND}$ (such algorithm is immediately from the collapse of S, U into a single unit).

Definition A.6 (Secure Blind Signature Scheme)**.** A blind signature scheme is secure if it satisfies the following properties:

1. **Correctness.** For any message m, and for all $(crs, \tau) \leftarrow \texttt{CRSgen}(1^\lambda)$ and $(vk, sk) \leftarrow \texttt{gen}(crs)$, if both S and U follow the protocol then S outputs completed and U outputs σ, which satisfies $\texttt{verify}(crs, vk, m, \sigma) = 1$.

2. **Unforgeability.** A one-more forgery adversary against the blind signature is a PPT machine $\mathcal{A}$ which is given crs and vk where $crs \leftarrow \texttt{CRSgen}(1^\lambda)$ and $(vk, sk) \leftarrow \texttt{gen}(crs)$, engages in $L = \mathsf{poly}(\lambda)$ interactions with the signer in concurrent and interleaving fashion, and terminates by returning ℓ message-signature pairs $(m_1, \sigma_1), ..., (m_\ell, \sigma_\ell)$ where $m_i \neq m_j$, $1 \leq i \neq j \leq \ell$. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^{\mathcal{A},L}_{\text{unforge}}(\lambda) = \Pr[(\texttt{verify}(crs, vk, m_i, \sigma_i) = 1, 1 \leq i \leq \ell) \wedge (\ell > L)]$ and we say that the blind signature scheme is unforgeable if for all PPT $\mathcal{A}$ and for all polynomial L, $\mathsf{Adv}^{\mathcal{A},L}_{\text{unforge}}(\lambda) \leq \mathsf{negl}(\lambda)$.

3. **Blindness.** A blindness distinguisher adversary against the blind signature is a PPT machine $\mathcal{A}$ which is given crs where $crs \leftarrow \texttt{CRSgen}(1^\lambda)$, outputs two messages and the verification key $\langle m_0, m_1, vk \rangle$, engages in two honest user instantiations in concurrent and interleaving fashion with inputs $\langle crs, vk, m_b \rangle$ and $\langle crs, vk, m_{1-b} \rangle$, respectively, where bit b is hidden; next the two user instantiations output σ_b and σ_{1-b}, respectively. If $\sigma_b \neq \perp$ and $\sigma_{1-b} \neq \perp$, then $\mathcal{A}$ is supplied with $\langle \sigma_0, \sigma_1 \rangle$; if $\sigma_b \neq \perp$ and $\sigma_{1-b} = \perp$, then $\mathcal{A}$ is supplied with $\langle \top, \perp \rangle$; if $\sigma_b = \perp$ and $\sigma_{1-b} \neq \perp$, then $\mathcal{A}$ is supplied with $\langle \perp, \top \rangle$; if $\sigma_b = \perp$ and $\sigma_{1-b} = \perp$, then $\mathcal{A}$ is supplied with $\langle \perp, \perp \rangle$. Finally $\mathcal{A}$ terminates by returning a bit b^*. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^{\mathcal{A}}_{\text{blind}}(\lambda) = |\Pr[b^* = b] - \frac{1}{2}|$, and say that the blind signature scheme satisfies the blindness property if for all PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\text{blind}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Remark A.7. The unforgeability property considered here follows [Oka06, HKKL07]; a stronger notion is considered in [JLO97, Fis06]. The blindness property considered here is taken from [CNS07] which is an extension of the blindness defined in [Oka06, ANN06].

A.6 Paillier Encryption

Here we give a brief description of Paillier encryption [Pai99]. Paillier encryption has been proven semantically secure if and only if the Decisional Composite Residuosity (DCR) assumption is true.

Key generation: let p and q be random primes for which $\mathsf{p},\mathsf{q} > 2$, $\mathsf{p} \neq \mathsf{q}$, $|\mathsf{p}| = |\mathsf{q}|$ and $\gcd(\mathsf{pq}, (\mathsf{p}-1)(\mathsf{q}-1)) = 1$; let $\mathsf{n} = \mathsf{pq}$, $\mathsf{d} = \mathrm{lcm}(\mathsf{p}-1, \mathsf{q}-1)$, $\mathsf{K} = \mathsf{d}^{-1} \bmod \mathsf{n}$, and $\mathsf{g} = (1+\mathsf{n})$; the public key is $pk = \langle \mathsf{n}, \mathsf{g} \rangle$ while the secret key is $sk = \langle \mathsf{p}, \mathsf{q} \rangle$.

Encryption: the plaintext set is $\mathbb{Z}_{\mathsf{n}}$; given a plaintext m, choose a random $\zeta \in \mathbb{Z}_{\mathsf{n}}^*$, and let the ciphertext be $c = \mathsf{g}^m \zeta^{\mathsf{n}} \bmod \mathsf{n}^2$.

Decryption: given a ciphertext c, observe that $c^{\mathsf{dK}} = \mathsf{g}^{m \cdot \mathsf{dK}} \cdot \zeta^{\mathsf{n} \cdot \mathsf{dK}} = \mathsf{g}^{m \cdot \zeta \mathsf{K} \bmod \mathsf{n}} \cdot \zeta^{\mathsf{n} \cdot \mathsf{dK} \bmod \mathsf{nd}} = \mathsf{g}^{m \bmod \mathsf{n}} \cdot \zeta^{0 \bmod \mathsf{nd}} = \mathsf{g}^m = 1 + m\mathsf{n} \bmod \mathsf{n}^2$. Thus, it is possible to recover $m = \frac{(c^{\mathsf{dK}} \bmod \mathsf{n}^2) - 1}{\mathsf{n}} \bmod \mathsf{n}$.

Definition A.8 (Decisional Composite Residuosity Assumption)**.** There is no PPT distinguisher $\mathcal{A}$ for n-th residues modulo n^2, i.e.

$$\mathsf{Adv}_{\mathrm{dcr}}^{\mathcal{A}} \stackrel{\mathsf{def}}{=} \left| \Pr[y \in \mathbb{Z}_{\mathsf{n}^2}^*; z \leftarrow y^{\mathsf{n}} \bmod \mathsf{n}^2 : \mathcal{A}(z) = 1] - \Pr[z \in \mathbb{Z}_{\mathsf{n}^2}^* : \mathcal{A}(z) = 1] \right| \leq \mathsf{negl}(\lambda).$$

Damgård and Jurik [DJ01] generalize Paillier encryption from modular n^2 to modular n^{s+1} where $s \in \mathbb{N}$. They also show that the extension is based on the DCR assumption above.

A.7 Commitments

A commitment scheme includes following algorithms: a PPT algorithm $\mathtt{gen}$ that produces the committing key pk; a PPT committing algorithm $\mathtt{com}$ that based on pk commits a plaintext m to a commitment value c, i.e. $c \leftarrow \mathtt{com}(pk, m; \zeta)$ where ζ is randomly selected, and $\langle m, \zeta \rangle$ is the opening ("decommitment") value; a polynomial time verification algorithm $\mathtt{ver}$ that checks if the opening value $\langle m, \zeta \rangle$ is consistent to the commitment value c.

Definition A.9 (Commitment Scheme)**.** $\Sigma(\mathrm{COM}) = \langle \mathtt{gen}, \mathtt{com}, \mathtt{ver} \rangle$ is a commitment scheme if the following properties hold:

Completeness: For all $pk \leftarrow \mathtt{gen}(1^\lambda)$, for all $m \in \mathcal{M}$, for any $\zeta \stackrel{\mathtt{r}}{\leftarrow} \mathtt{RND}$, $\mathtt{ver}(pk, \mathtt{com}(pk, m; \zeta), m, \zeta) = 1$.

Binding: For all PPT adversaries $\mathcal{A}$,

$$\mathsf{Adv}_{\mathrm{binding}}^{\mathcal{A}} \stackrel{\mathsf{def}}{=} \Pr[pk \leftarrow \mathtt{gen}(1^\lambda); (c, m_1, \zeta_1, m_2, \zeta_2) \leftarrow \mathcal{A}(pk) : \mathtt{ver}(pk, c, m_1, \zeta_1) = \mathtt{ver}(pk, c, m_2, \zeta_2) = 1 \wedge m_1 \neq m_2] \leq \mathsf{negl}(\lambda).$$

Hiding: For all PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}_{\mathrm{hiding}}^{\mathcal{A}} \stackrel{\mathsf{def}}{=} \left| \Pr\left[\begin{array}{l} pk \leftarrow \mathtt{gen}(1^\lambda); (m_1, m_2) \leftarrow \mathcal{A}(pk); \\ \zeta_1 \stackrel{\mathtt{r}}{\leftarrow} \mathtt{RND}; c_1 \leftarrow \mathtt{com}(pk, m_1; \zeta_1) : \mathcal{A}(c_1) = 1 \end{array} \right] - \Pr\left[\begin{array}{l} pk \leftarrow \mathtt{gen}(1^\lambda); (m_1, m_2) \leftarrow \mathcal{A}(pk); \\ \zeta_2 \stackrel{\mathtt{r}}{\leftarrow} \mathtt{RND}; c_2 \leftarrow \mathtt{com}(pk, m_2; \zeta_2) : \mathcal{A}(c_2) = 1 \end{array} \right] \right| \leq \mathsf{negl}(\lambda).$$

Next we define equivocal commitment schemes and extractable commitment schemes.

Definition A.10 (Equivocal Commitment)**.** $\Sigma(\mathrm{EQC}) = \langle \mathtt{gen}, \mathtt{com}, \mathtt{ver}, \mathtt{fake}, \mathtt{equivocate} \rangle$ is an equivocal commitment scheme if $\mathtt{gen}(1^\lambda)$ outputs a key pair $\langle pk, ek \rangle$, and if $\mathtt{gen}_1$ is the algorithm returning only the first element of the output of $\mathtt{gen}$ then $\langle \mathtt{gen}_1, \mathtt{com}, \mathtt{ver} \rangle$ is a commitment scheme as well as the following property holds.

Equivocality: If for any PPT distinguisher $\mathcal{A}$ we have

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{eq}} \stackrel{\mathsf{def}}{=} \left| \begin{array}{l} \Pr[(pk, ek) \leftarrow \mathtt{gen}(1^\lambda) : \mathcal{A}^{\widehat{\mathcal{O}}_{pk,ek}}(pk) = 1] \\ \quad - \Pr[(pk, ek) \leftarrow \mathtt{gen}(1^\lambda) : \mathcal{A}^{\mathcal{O}_{pk}}(pk) = 1] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

The oracles are defined as follows: $\widehat{\mathcal{O}}_{pk,ek}$ on query $m \in \mathcal{M}$ returns (c, ζ), where $(c, aux) \leftarrow \mathtt{fake}(pk, ek)$ and $\zeta \leftarrow \mathtt{equivocate}(pk, c, aux, m)$; $\mathcal{O}_{pk}$ on query $m \in \mathcal{M}$ returns (c, ζ), where $c \leftarrow \mathtt{com}(pk, m; \zeta)$ and $\zeta \xleftarrow{\mathtt{r}} \mathtt{RND}$.

Definition A.11 (Extractable Commitment)**.** $\Sigma(\mathrm{EXC}) = \langle \mathtt{gen}, \mathtt{com}, \mathtt{ver}, \mathtt{extract} \rangle$ is an extractable commitment scheme if $\mathtt{gen}(1^\lambda)$ outputs a key pair $\langle pk, xk \rangle$, and if $\mathtt{gen}_1$ is the algorithm returning only the first element of the output of $\mathtt{gen}$ then $\langle \mathtt{gen}_1, \mathtt{com}, \mathtt{ver} \rangle$ is a commitment scheme as well as the following property holds.

Extractability: For all $(pk, xk) \leftarrow \mathtt{gen}(1^\lambda)$, for all $m \in \mathcal{M}$ and for any $\zeta \xleftarrow{\mathtt{r}} \mathtt{RND}$, there exists an extraction algorithm $\mathtt{extract}$ such that $\mathtt{extract}(pk, xk, \mathtt{com}(pk, m; \zeta)) = m$.

A.8 Non-Interactive Zero-Knowledge

Here we briefly introduce non-interactive zero-knowledge schemes. The reader can refer to e.g., [GOS06] for a more detailed discussion. Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. Let $\mathcal{L}_R$ be the language consisting of statements in R, i.e. $\mathcal{L}_R \stackrel{\mathsf{def}}{=} \{x | \exists w \text{ s.t. } (x, w) \in R\}$. An NIZK scheme includes following algorithms: a PPT algorithm $\mathtt{gen}$ producing a CRS crs together with a trapdoor τ; a PPT algorithm $\mathtt{prove}$ that takes as input crs and $(x, w) \in R$ and outputs a proof ϖ; a polynomial time algorithm $\mathtt{verify}$ takes as input (crs, x, ϖ) and outputs 1 if the proof is valid and 0 otherwise.

Definition A.12 (Non-Interactive Zero-Knowledge (NIZK) Scheme)**.** $\Sigma(\mathrm{NIZK}) = \langle \mathtt{gen}, \mathtt{prove}, \mathtt{verify}, \mathtt{simulate} \rangle$ is an NIZK scheme for the relation R if the following properties hold:

Completeness: For any $(x, w) \in R$,

$$\Pr[(crs, \tau) \leftarrow \mathtt{gen}(1^\lambda); \zeta \xleftarrow{\mathtt{r}} \mathtt{RND}; \varpi \leftarrow \mathtt{prove}(crs, x, w; \zeta) : \mathtt{verify}(crs, x, \varpi) = 0] \leq \mathsf{negl}(\lambda).$$

Soundness: For all PPT adversary $\mathcal{A}$,

$$\Pr[(crs, \tau) \leftarrow \mathtt{gen}(1^\lambda); (x, \varpi) \leftarrow \mathcal{A}(crs) : x \notin \mathcal{L}_R \wedge \mathtt{verify}(crs, x, \varpi) = 1] \leq \mathsf{negl}(\lambda).$$

Zero-knowledge: If for any PPT distinguisher $\mathcal{A}$ we have

$$\left| \begin{array}{l} \Pr[(crs, \tau) \leftarrow \mathtt{gen}(1^\lambda) : \mathcal{A}^{\widehat{\mathcal{O}}_{crs,\tau}}(crs) = 1] \\ \quad - \Pr[(crs, \tau) \leftarrow \mathtt{gen}(1^\lambda) : \mathcal{A}^{\mathcal{O}_{crs}}(crs) = 1] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

The oracles are defined as follows: $\widehat{\mathcal{O}}_{crs,\tau}$ on query $(x, w) \in R$ returns ϖ, where $(\varpi, aux) \leftarrow \mathtt{simulate}(crs, \tau, x)$; $\mathcal{O}_{crs}$ on query $(x, w) \in R$ returns ϖ, where $\varpi \leftarrow \mathtt{prove}(crs, x, w; \zeta)$ and $\zeta \xleftarrow{\mathtt{r}} \mathtt{RND}$.

Next we define non-erasure NIZK schemes.

Definition A.13 (Non-Erasure NIZK (NENIZK))**.** $\Sigma(\text{NENIZK}) = \langle \texttt{gen}, \texttt{prove}, \texttt{verify}, \texttt{simulate}, \texttt{reconstruct} \rangle$ is a non-erasure NIZK scheme if $\langle \texttt{gen}, \texttt{prove}, \texttt{verify}, \texttt{simulate} \rangle$ is an NIZK scheme and the following property holds.

Non-erasure zero-knowledge: If for any PPT distinguisher $\mathcal{A}$ we have

$$\left| \begin{array}{l} \Pr[(crs, \tau) \leftarrow \texttt{gen}(1^\lambda) : \mathcal{A}^{\widehat{\mathcal{O}}_{crs,\tau}}(crs) = 1] \\ \quad - \Pr[(crs, \tau) \leftarrow \texttt{gen}(1^\lambda) : \mathcal{A}^{\mathcal{O}_{crs}}(crs) = 1] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

The oracles are defined as follows: $\widehat{\mathcal{O}}_{crs,\tau}$ on query $(x, w) \in R$ returns (ϖ, ζ), where $(\varpi, aux) \leftarrow \texttt{simulate}(crs, \tau, x)$ and $\zeta \leftarrow \texttt{reconstruct}(crs, x, \varpi, aux, w)$; $\mathcal{O}_{crs}$ on query $(x, w) \in R$ returns (ϖ, ζ), where $\varpi \leftarrow \texttt{prove}(crs, x, w; \zeta)$ and $\zeta \xleftarrow{\texttt{r}} \texttt{RND}$.

A.9 Sigma Protocols

In this subsection we introduce Sigma protocols, [CDS94]. Informally, a Sigma-protocol is a three move public randomness protocol (cf. Figure 11) with an honest verifier zero-knowledge property and a special soundness property. In our formulation of Sigma-protocols below we will formalize soundness only to satisfy membership consistency (rather than knowledge extraction).

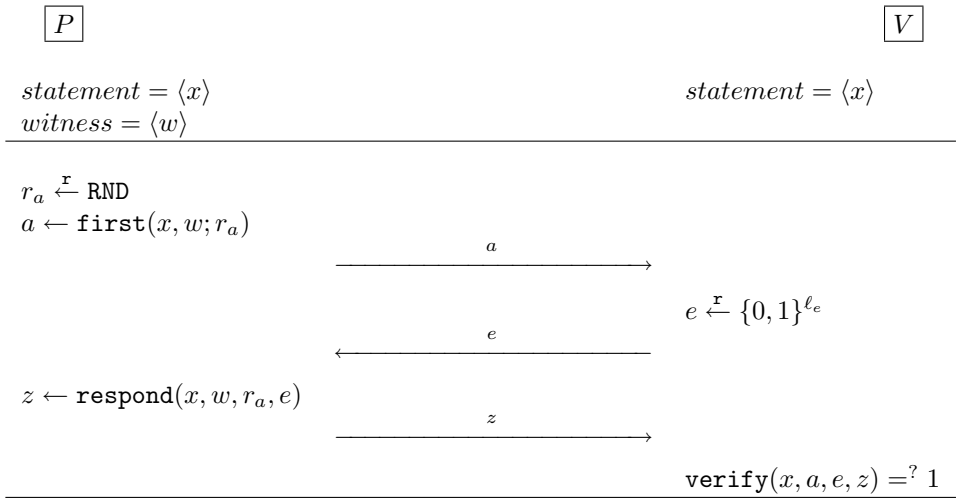

Figure 11: Three move public randomness protocol for relation R. Here ℓ_e is the length of the challenge e.

Definition A.14 (Sigma Protocol)**.** A Sigma protocol for the relation R is a tuple $\langle \texttt{first}, \texttt{respond}, \texttt{verify}, \texttt{simulate} \rangle$, where $\langle \texttt{first}, \texttt{respond}, \texttt{verify} \rangle$ is a three-move public randomness protocol, $\texttt{simulate}$ is a PPT algorithm, and where the following properties hold:

Completeness: For any $(x, w) \in R$,

$$\Pr[r_a \xleftarrow{\texttt{r}} \texttt{RND}; a \leftarrow \texttt{first}(x, w; r_a); z \leftarrow \texttt{respond}(x, w, r_a, e) : \texttt{verify}(x, a, e, z) = 0] \leq \mathsf{negl}(\lambda).$$

Special membership soundness: For all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, for any statement x,

$$\mathsf{Adv}^{\mathcal{A}}_{\text{sound}} \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l} e \xleftarrow{\texttt{r}} \{0,1\}^{\ell_e}; (a, aux) \leftarrow \mathcal{A}_1(x); z \leftarrow \mathcal{A}_2(e, aux)) : \\ \texttt{verify}(x, a, e, z) = 1 \wedge x \notin \mathcal{L}_R \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Honest-verifier zero-knowledge: For any PPT distinguisher $\mathcal{A}$ and for any $(x, w) \in R$, we have

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{zk}} \stackrel{\mathsf{def}}{=} \left| \begin{array}{l} \Pr\left[\begin{array}{l} r_a \xleftarrow{\mathtt{r}} \mathtt{RND}; a \leftarrow \mathtt{first}(x, w; r_a); e \xleftarrow{\mathtt{r}} \{0,1\}^{\ell_e}; \\ z \leftarrow \mathtt{respond}(x, w, r_a, e) : \mathcal{A}(x, w, a, e, z) = 1 \end{array} \right] \\ \quad - \Pr\left[\begin{array}{l} e \xleftarrow{\mathtt{r}} \{0,1\}^{\ell_e}; (a, z, aux) \leftarrow \mathtt{simulate}(x, e) : \\ \mathcal{A}(x, w, a, e, z) = 1 \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

Next we will introduce the non-erasure property for Sigma protocols.

Definition A.15 (Non-erasure sigma protocol)**.** A non-erasure Sigma protocol for relation R is a tuple $\langle \mathtt{first}, \mathtt{respond}, \mathtt{verify}, \mathtt{simulate}, \mathtt{reconstruct} \rangle$, where $\langle \mathtt{first}, \mathtt{respond}, \mathtt{verify}, \mathtt{simulate} \rangle$ is a Sigma protocol, `reconstruct` is a PPT algorithm, and the following property also holds:

Non-erasure honest-verifier zero-knowledge: For any PPT distinguisher $\mathcal{A}$ and any $(x, w) \in R$, we have

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{nezk}} \stackrel{\mathsf{def}}{=} \left| \begin{array}{l} \Pr\left[\begin{array}{l} r_a \xleftarrow{\mathtt{r}} \mathtt{RND}; a \leftarrow \mathtt{first}(x, w; r_a); e \xleftarrow{\mathtt{r}} \{0,1\}^{\ell_e}; \\ z \leftarrow \mathtt{respond}(x, w, r_a, e) : \mathcal{A}(x, w, a, e, z, r_a) = 1 \end{array} \right] \\ \quad - \Pr\left[\begin{array}{l} e \xleftarrow{\mathtt{r}} \{0,1\}^{\ell_e}; (a, z, aux) \leftarrow \mathtt{simulate}(x, e); \\ r_a \leftarrow \mathtt{reconstruct}(x, a, z, w, e, aux) : \mathcal{A}(x, w, a, e, z, r_a) = 1 \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

We note that the zero-knowledge formulated above is weak (honest verifier) and thus unsuitable for many settings. Nevertheless, there are generic methods of transforming an honest-verifier zero-knowledge protocol to one that satisfies zero-knowledge in more reasonable adversarial settings. For example, Damgård showed how one can use an equivocal commitment to extend a Sigma protocol to achieve concurrent zero-knowledge in the CRS model [Dam00]. Please refer to Section 2.9 of Nielsen's PhD thesis [Nie03] for a wonderful explanation of Sigma protocols.

A.10 Other Primitives

Here we present some other definitions that will be useful in the sequel.

Definition A.16 (Collision Resistent Hash Function Family)**.** Let $\mathscr{H}$ be a finite family of functions such that $\forall \mathcal{H} \in \mathscr{H}$, we have $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_Q$. We say $\mathscr{H}$ is a collision resistent hash function family if for any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{crhf}} \stackrel{\mathsf{def}}{=} \Pr[\mathcal{H} \leftarrow \mathscr{H}; x, y \leftarrow \mathcal{A}(1^\lambda, program(\mathcal{H})) : \mathcal{H}(x) = \mathcal{H}(y) \wedge x \neq y] \leq \mathsf{negl}(\lambda).$$

where $program(\mathcal{H})$ denotes an implementation of $\mathcal{H}$.

Definition A.17 (Discrete Logarithm (DLOG) Assumption)**.** Let $\mathbf{G}$ be a cyclic group of prime order p where p is at least λ-bits. The DLOG problem defined as follows: given $\mathbf{g}, \mathbf{g}^a \in \mathbf{G}$, output $a \in \mathbb{Z}_p$. The DLOG assumption suggests that any PPT algorithm $\mathcal{A}$ solving the DLOG problem has negligible probability, i.e.

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{dlog}} \stackrel{\mathsf{def}}{=} \Pr\left[\mathbf{g} \in \mathbf{G}; a \xleftarrow{\mathtt{r}} \mathbb{Z}_p : \mathcal{A}(\mathbf{g}, \mathbf{g}^a) = a\right] \leq \mathsf{negl}(\lambda).$$

B More on UC Blind Signatures Definition

In the body part, we define the ideal functionality $\mathcal{F}_{\text{BSIG}}$ in Figure 5. Here we provide the definition of UC blind signature protocol and show that protocols realizing such $\mathcal{F}_{\text{BSIG}}$ will be secure in some existing model e.g. of [Oka06, HKKL07].

As defined in Definition A.4, a blind signature scheme is a tuple $\Sigma(\text{BSIG}) = \langle \texttt{CRSgen}, \texttt{gen}, U, S, \texttt{verify} \rangle$ where U, S is an interactive protocol between the user and the signer. In each round of the protocol the user and the signer exchange messages denoted as $\mathsf{UserMsg}_i, \mathsf{SignerMsg}_i$ (note that U, S may extend to many rounds). Given such protocol we can also define a `sign` algorithm by collapsing the interactive protocol U, S into a non-interactive algorithm (that simulates both parties). Given such a scheme each party in the UC-framework will execute a program $\pi_{\Sigma(\text{BSIG})}$ that is described in Figure 12.

Definition B.1. A blind signature scheme $\Sigma(\text{BSIG}) = \langle \texttt{CRSgen}, \texttt{gen}, U, S, \texttt{verify} \rangle$ is UC-secure in the CRS model if $\pi_{\Sigma(\text{BSIG})}$ realizes the ideal functionality $\mathcal{F}_{\text{BSIG}}$ in Figure 5 in the $\mathcal{F}_{\text{CRS}}$-hybrid world, where $\mathcal{F}_{\text{CRS}}$ is an implementation of the `CRSgen` algorithm as given in the specification of the scheme.

We note that each party that acts as a user in our framework is programmed to ask for a single signature; we make this choice without loss of generality and for the sake of keeping the description of the scheme simple. We prove that if a blind signature protocol securely realizes $\mathcal{F}_{\text{BSIG}}$, then the scheme is also secure in the security model for blind signatures that was proposed on the concurrent security model of [Oka06, HKKL07] (cf. Section A.6 and Remark A.7).

The theorem below is a sanity-check that shows that the ideal functionality we propose is consistent with some of the previous modeling attempts. Naturally the intention is that realizing the ideal functionality goes much beyond the satisfaction of such previous game-based definitions. Still establishing results as the one below is important and non-trivial due to the fact that ideal functionalities interact with the ideal-world adversary substantially something that may (if they are badly designed) lead to disparities with game-based definitions.

Theorem B.2. *If $\Sigma(\text{BSIG})$ can realize our $\mathcal{F}_{\text{BSIG}}$ in Figure 5, then it is secure in the blind signature security model of [Oka06, HKKL07].*

Proof. The general plan of the proof is as follows: we first assume $\Sigma(\text{BSIG})$ is not secure according to one of the previous definitions. Then we construct an environment $\mathcal{Z}$ so that for all $\mathcal{S}$ it can distinguish the interaction with $\pi_{\Sigma(\text{BSIG})}$, from the interaction with the ideal world adversary $\mathcal{S}$ and $\mathcal{F}_{\text{BSIG}}$.

(I) We first assume that there is a successful forger F for $\Sigma(\text{BSIG})$. Then $\mathcal{Z}$ internally runs an instance of F. The environment $\mathcal{Z}$ invokes party S with (KeyGen, sid), and gives the returned verification algorithm ver to F.

When the simulated F outputs user_1 message on behalf of some party U, $\mathcal{Z}$ creates party U; it corrupts party U and forces it to send an outgoing user_1 message (through $\mathcal{Z}$) to party S; when F outputs a user_i message where $i > 1$, $\mathcal{Z}$ forces party U to send the user_i message to party S; when the corrupted party U receives an incoming signer_j message, $\mathcal{Z}$ forwards it to the simulated F. $\mathcal{Z}$ uses $counter_1$ to count the number of successful final signer messages from the party S. $\mathcal{Z}$ uses $counter_2$ to count the number of $(\text{SignStatus}, sid, U, \mathsf{completed})$ messages obtained from party S (as this party returns this value to the subroutine output tape of $\mathcal{Z}$). When the simulated F outputs a number of, say L, forged message-signature pairs, $\mathcal{Z}$ activates L verifiers with $(\text{Verify}, sid, \widehat{m}_i, \widehat{\sigma}_i, \mathsf{ver})$, where $i = 1, \ldots, L$; the verifiers will return 1, i.e. $(\text{Verified}, sid, 1)$, for the successful forged pairs. $\mathcal{Z}$ uses $counter_3$ to count the number of 1's. If $\mathcal{Z}$ finds that $counter_3 \leq counter_2$ it returns 0 otherwise it returns 1.

Blind Signature Protocol $\pi_{\Sigma(\text{BSIG})}$

CRS generation: $crs \leftarrow \texttt{CRSgen}(1^\lambda)$ where λ is the security parameter.

Key generation: When party S is invoked with input (KEYGEN, sid) by $\mathcal{Z}$, it verifies that $sid = (S, sid')$ for some sid'; If not, it ignores the input; Otherwise, it runs $(vk, sk) \leftarrow \texttt{gen}(crs)$, lets the verification algorithm $\mathsf{ver} := \texttt{verify}(crs, vk, \cdot, \cdot)$, records $\langle sk, vk \rangle$ in $history(S)$, and outputs $(\text{VERIFICATIONALG}, sid, \mathsf{ver})$ to $\mathcal{Z}$.

Signature generation: When party U is invoked with input $(\text{SIGN}, sid, m, \mathsf{ver}')$ by $\mathcal{Z}$ where $sid = (S, sid')$ and $U \neq S$, it computes $\mathsf{UserMsg}_1$ where some random coins rnd_1^U may be used, records $\langle m, \mathsf{ver}', rnd_1^U \rangle$ in $history(U)$, and sends outgoing $(\text{USER}_1, sid, \mathsf{UserMsg}_1)$ to party S (through $\mathcal{Z}$). If party U cannot compute $\mathsf{UserMsg}_1$, then outputs $(\text{SIGNATURE}, sid, \perp)$ to $\mathcal{Z}$.

When party U is invoked with incoming $(\text{SIGNER}_i, sid, \mathsf{SignerMsg}_i)$ from party S, where $sid = (S, sid')$, it computes $\mathsf{UserMsg}_{i+1}$ where some random coins rnd_{i+1}^U may be used, records $\langle rnd_{i+1}^U \rangle$ in $history(U)$, and sends outgoing $(\text{USER}_{i+1}, sid, \mathsf{UserMsg}_{i+1})$ to party S. If U cannot compute $\mathsf{UserMsg}_{i+1}$, then outputs $(\text{SIGNATURE}, sid, \perp)$ to $\mathcal{Z}$, and may also send outgoing (USER_{i+1}, sid) to party S.

When party S is invoked with incoming $(\text{USER}_i, sid, \mathsf{UserMsg}_i)$ from party U, where $sid = (S, sid')$, it computes $\mathsf{SignerMsg}_i$ where some random coins rnd_i^S may be used, records $\langle rnd_i^S, U \rangle$ in $history(S)$, and sends outgoing $(\text{SIGNER}_i, sid, \mathsf{SignerMsg}_i)$ to party U. If S cannot compute $\mathsf{SignerMsg}_i$, then outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{incompleted})$ to $\mathcal{Z}$, and may also send outgoing (SIGNER_i, sid) to party U.

When party S is invoked with incoming $(\text{USER}_{last}, sid, \mathsf{UserMsg}_{last})$, which is the last user message from party U, where $sid = (S, sid')$, it computes $\mathsf{SignerMsg}_{last}$ where some random coins rnd_{last}^S may be used, records $\langle rnd_{last}^U, U \rangle$ in $history(S)$, and sends outgoing $(\text{SIGNER}_{last}, sid, \mathsf{SignerMsg}_{last})$ to party U, and outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{completed})$ to $\mathcal{Z}$. If party S cannot compute $\mathsf{SignerMsg}_{last}$, then outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{incompleted})$ to $\mathcal{Z}$, and may also send outgoing $(\text{SIGNER}_{last}, sid)$ to party U.

When party U is invoked with incoming $(\text{SIGNER}_{last}, sid, \mathsf{SignerMsg}_{last})$, which is the last signer message from party S, where $sid = (S, sid')$, it computes signature σ for m where some random coins rnd_{last+1}^U may be used, records $\langle \sigma, rnd_{last+1}^U \rangle$ in $history(U)$, and outputs $(\text{SIGNATURE}, sid, \sigma)$ to $\mathcal{Z}$. If σ cannot be computed, then it outputs $(\text{SIGNATURE}, sid, \perp)$ to $\mathcal{Z}$.

Signature verification: When party V is invoked with input $(\text{VERIFY}, sid, m, \sigma, \mathsf{ver}')$ by $\mathcal{Z}$ where $sid = (S, sid')$, it outputs $(\text{VERIFIED}, sid, \mathsf{ver}'(m, \sigma))$ to $\mathcal{Z}$.

Corruption: When party J is invoked with incoming $(\text{CORRUPT}, sid, J)$ by $\mathcal{Z}$, it sends outgoing $(\text{CORRUPTED}, sid, history(J))$ to $\mathcal{Z}$. Here J can be party U or party S.

Figure 12: Blind signature protocol $\pi_{\Sigma(\text{BSIG})}$.

In the real world, because F is a successful forger, with non-negligible probability, $\mathcal{Z}$ will observe $counter_3 > counter_1$. Moreover, note that $counter_1 = counter_2$ will hold in the real world, which is based on the fact: when party S sends his last outgoing SIGNER message to party U, he also sends output $(\text{SIGNSTATUS}, sid, U, \mathsf{completed})$ to $\mathcal{Z}$. It follows that when $\mathcal{Z}$ operates in the real world it returns 1 with non-negligible probability.

Next we turn to the ideal world. In the verification stage, when a verifier receives $(\text{VERIFY}, sid, \widehat{m}_i, \widehat{\sigma}_i, \mathsf{ver})$, he will forward such message to $\mathcal{F}_{\text{BSIG}}$, and $\mathcal{F}_{\text{BSIG}}$ will check if the message is a forgery using the information he possesses. Based on the definition of $\mathcal{F}_{\text{BSIG}}$, such check will return $\mathsf{ver}(\widehat{m}_i, \widehat{\sigma}_i)$ to the verifier, as long $\widehat{m}_i$ is recorded with done. In case the message is not recorded with done, $\mathcal{F}_{\text{BSIG}}$ would either return $(\text{VERIFIED}, sid, \widehat{m}_i, \widehat{\sigma}_i, 0)$ or halt. It follows that $counter_1$ will be incremented only due to messages recorded with done. Given that all users are corrupted on start there is only one possibility that a message is recorded with done within $\mathcal{F}_{\text{BSIG}}$: $\mathcal{S}$ has patched the particular message into some corrupted user instance inside $\mathcal{F}_{\text{BSIG}}$. Note that a message is recorded with done by $\mathcal{F}_{\text{BSIG}}$ only after a message $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerComplete})$ is received from $\mathcal{S}$. The environment tracks the $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerComplete})$ messages into $counter_2$ thus we can be certain that in the ideal world it would be impossible to have $counter_3 > counter_2$. It follows that in the ideal world, $counter_3 \leq counter_2$; thus it follows that the environment always returns 0 and it is a distinguisher between the real and the ideal world for any implementation of $\mathcal{S}$.

(II) Next assume there is a successful blindness distinguisher D for $\Sigma(\text{BSIG})$. Then $\mathcal{Z}$ internally runs an instance of D. Such D can be viewed as a signer inside $\mathcal{Z}$. When D outputs the verification algorithm ver and two messages $\langle m_0, m_1 \rangle$, $\mathcal{Z}$ activates two parties U_L and U_R with $(\text{SIGN}, sid, m_b, \mathsf{ver})$ and $(\text{SIGN}, sid, m_{1-b}, \mathsf{ver})$, respectively, where $b \xleftarrow{\text{r}} \{0, 1\}$ is randomly chosen by $\mathcal{Z}$.

Subsequently $\mathcal{Z}$ relays all messages communicated between the party D and the two user protocols. Next if $\mathcal{Z}$ obtains two valid responses $(\text{SIGNATURE}, sid, \sigma_b)$, $(\text{SIGNATURE}, sid, \sigma_{1-b})$ from the two parties U_L and U_R respectively. If $\sigma_b \neq \perp$ and $\sigma_{1-b} \neq \perp$, then D is supplied with $\langle \sigma_0, \sigma_1 \rangle$; If $\sigma_b \neq \perp$ and $\sigma_{1-b} = \perp$, then D is supplied with $\langle \top, \perp \rangle$; If $\sigma_b = \perp$ and $\sigma_{1-b} \neq \perp$, then D is supplied with $\langle \perp, \top \rangle$; If $\sigma_b = \perp$ and $\sigma_{1-b} = \perp$, then D is supplied with $\langle \perp, \perp \rangle$; Finally D returns b^* as the guess of the random coin b which is chosen by $\mathcal{Z}$. Here $\mathcal{Z}$ returns $b^* \stackrel{?}{=} b$.

Consider now that D is a successful blindness distinguisher for $\Sigma(\text{BSIG})$; in the real world, D will guess the coin b with non-negligible advantage. However, in the ideal world, no matter how the simulator $\mathcal{S}$ is implemented we observe that the bit b remains secured in $\mathcal{Z}$ and the ideal functionality $\mathcal{F}_{\text{BSIG}}$ does not communicate any information related to b to $\mathcal{S}$. Moreover, even if $\mathcal{S}$ jams one of the user instantiations the environment $\mathcal{Z}$ following its program will only return $\langle \top, \perp \rangle$ or $\langle \perp, \top \rangle$ to the distinguisher D. Therefore, $\mathcal{S}$ cannot help D to figure out b.

Due to an attack in [HK07], still D may use two different signing algorithms sig_L and sig_R for the interactions with the users U_L and U_R respectively; note that both sig_L and sig_R correspond to the same verification algorithm ver; once D receives $\langle \sigma_0, \sigma_1 \rangle$ from $\mathcal{Z}$, he may use sig_L and sig_R to relate the message-signature pairs to the users correctly and figure out b. However, once D uses two different signing algorithms, both of them will be patched into $\mathcal{F}_{\text{BSIG}}$; now both $\langle U_L, \mathsf{sig}_L, \mathsf{ver} \rangle$ and $\langle U_R, \mathsf{sig}_R, \mathsf{ver} \rangle$ will be recorded which will halt the execution, and $\mathcal{Z}$ can easily distinguish the two worlds. This follows that the coins b are independent from D, and even an unbounded D cannot guess such b with probability better than $1/2$. It follows that $\mathcal{Z}$ is a distinguisher between the real and the ideal worlds. □

Functionality $\mathcal{F}_{\mathrm{SVZK}}^{R}$

$\mathcal{F}_{\mathrm{SVZK}}^{R}$ is parameterized by a binary relation R.

Proof stage: Upon receiving $(\text{PROVESVZK}, sid, P_i, x, w)$ from party P_i, verify that $sid = (V, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R$ then record $\langle x, w \rangle$ into $history(P_i)$, and forward $(\text{PROVESVZK}, sid, P_i, x)$ to the adversary $\mathcal{S}$.

Upon receiving $(\text{PROVESVZK}, sid, P_i, \mathsf{VerifierComplete})$ from the adversary $\mathcal{S}$, output $(\text{VERIFIEDSVZK}, sid, P_i, x)$ to party V.

Corruption: Upon receiving $(\text{CORRUPTPROVERSVZK}, sid, P_i)$ from the adversary $\mathcal{S}$, return $\mathcal{S}$ $(\text{CORRUPTEDPROVERSVZK}, sid, history(P_i))$.

After the successful corruption of P_i, if no output $(\text{VERIFIEDSVZK}, sid, P_i, ...)$ was returned to party V yet, upon receiving $(\text{PATCHSVZK}, sid, P_i, x', w')$ from the adversary $\mathcal{S}$, if $(x', w') \in R$ then output $(\text{VERIFIEDSVZK}, sid, P_i, x')$ to party V, else if $(x', w') \notin R$, then output $(\text{VERIFIEDSVZK}, sid, P_i, \perp)$ to party V; upon receiving $(\text{PATCHSVZK}, sid, P_i, \mathsf{VerifierError})$ from the adversary $\mathcal{S}$, then output $(\text{VERIFIEDSVZK}, sid, P_i, \perp)$ to party V.

Figure 13: Single-verifier zero-knowledge functionality $\mathcal{F}_{\mathrm{SVZK}}^{R}$.

Functionality $\mathcal{F}_{\mathrm{SPZK}}^{R}$

$\mathcal{F}_{\mathrm{SPZK}}^{R}$ is parameterized by a binary relation R.

Proof stage: Upon receiving $(\text{PROVESPZK}, sid, V_i, x, w)$ from party P, verify that $sid = (P, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R$ then record $\langle V_i, x, w \rangle$ in $history(P)$, and forward $(\text{PROVESPZK}, sid, V_i, x)$ to the adversary $\mathcal{S}$.

Upon receiving $(\text{PROVESPZK}, sid, V_i, \mathsf{VerifierComplete})$ from the adversary $\mathcal{S}$, output $(\text{VERIFIEDSPZK}, sid, V_i, x)$ to party V_i.

Corruption: Upon receiving $(\text{CORRUPTPROVERSPZK}, sid)$ from the adversary $\mathcal{S}$, return $\mathcal{S}$ $(\text{CORRUPTEDPROVERSPZK}, sid, history(P))$.

After the corruption has occurred successfully, if no output $(\text{VERIFIEDSPZK}, sid, V_i, ...)$ was returned to party V_i yet, upon receiving $(\text{PATCHSPZK}, sid, V_i, x', w')$ from the adversary $\mathcal{S}$, if $(x', w') \in R$ then output $(\text{VERIFIEDSPZK}, sid, V_i, x')$ to party V_i, else if $(x', w') \notin R$ then output $(\text{VERIFIEDSPZK}, sid, V_i, \perp)$ to party V_i; upon receiving $(\text{PATCHSPZK}, sid, V_i, \mathsf{VerifierError})$ from the adversary $\mathcal{S}$, then output $(\text{VERIFIEDSPZK}, sid, V_i, \perp)$ to party V_i.

Figure 14: Single-prover zero-knowledge functionality $\mathcal{F}_{\mathrm{SPZK}}^{R}$.

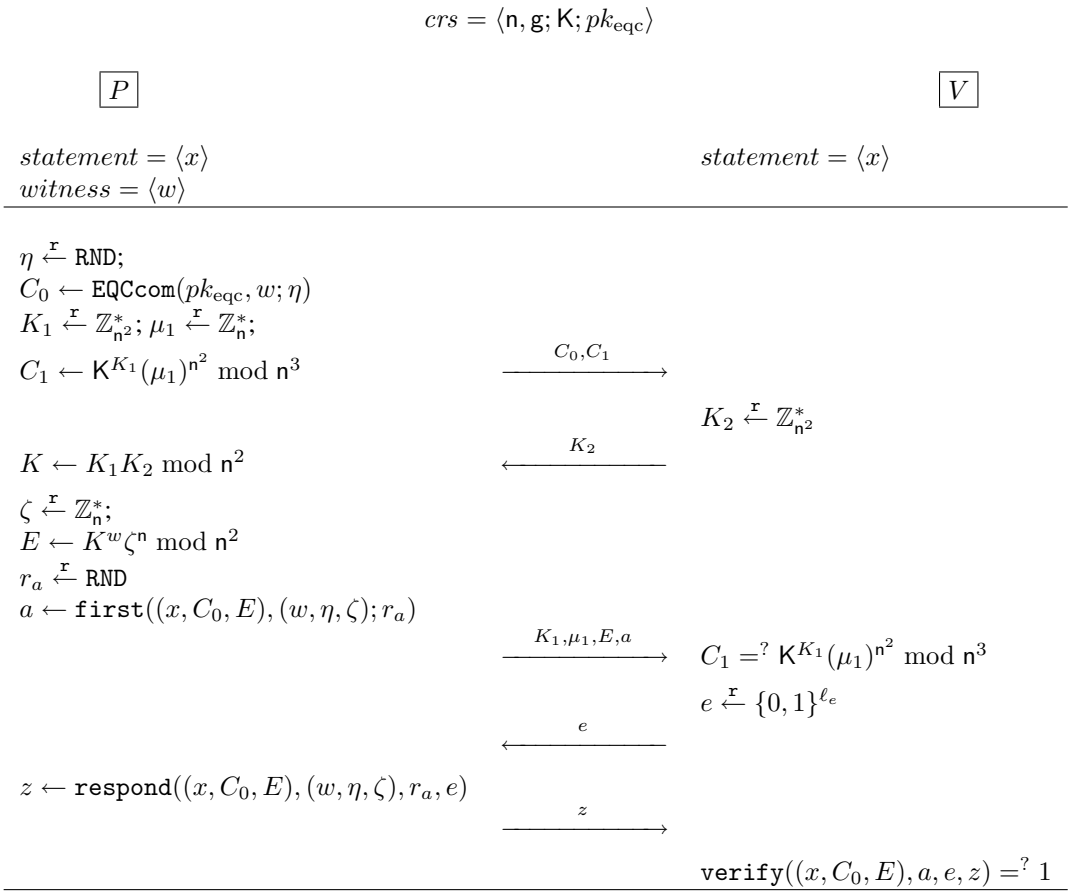

Figure 15: A combination of a committing step with a non-erasure Sigma protocol for relation $R'_U = \{((crs, x, C_0, E), (w, \eta, \zeta)) \mid (x, w) \in R_U \wedge E = K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2 \wedge C_0 = \mathtt{EQCcom}(pk_{\mathrm{eqc}}, w; \eta)\}$ in the single-verifier setting. Here EQCcom is the committing algorithm for equivocal commitment scheme EQC.

C More on SVZK and SPZK

Here we first describe the functionalities $\mathcal{F}^R_{\mathrm{SVZK}}$ and $\mathcal{F}^R_{\mathrm{SPZK}}$ in Figure 13 and Figure 14 respectively; then we show the realization of the two functionalities. Please refer to the full version for the proofs and also more details of this part.

C.1 Implementation for $\mathcal{F}_{\mathrm{SVZK}}$

Following the implementation strategy for $\mathcal{F}^{R_U}_{\mathrm{SVZK}}$ in Section 3.2 in the body part, an SVZK instantiation is presented in Figure 16: (1) it is based on a two-move mixed-commitment in the single-verifier setting, and EQC is an equivocal commitment which is used to strengthen the mixed-commitment by binding the committer from the beginning (please refer to section 9.4 in [Nie03] for more discussion); (2) a non-erasure Sigma protocol $\langle \mathtt{first}, \mathtt{respond}, \mathtt{verify}, \mathtt{simulate}, \mathtt{reconstruct} \rangle$ is used to show the consistency of the relation R_U and the strengthened mixed-commitment, i.e. for relation $R'_U = \{((crs, x, C_0, E), (w, \eta, \zeta)) \mid (x, w) \in R_U \wedge E = K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2 \wedge C_0 = \mathtt{EQCcom}(pk_{\mathrm{eqc}}, w; \eta)\}$, and a combination of the Sigma pro-

Protocol $\pi_{\Sigma(\text{SVZK})}$ in the $\mathcal{F}_{\text{CRS}}$-Hybrid World

Proof stage: When party P is invoked with input $(\text{PROVESPZK}, sid, P, x, w)$ by $\mathcal{Z}$, it verifies that $sid = (V, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R_U$, it computes $C_0 \leftarrow \texttt{EQCcom}(pk_{\text{eqc}}, w; \eta)$ and $C_1 \leftarrow \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$ where $\eta \xleftarrow{r} \texttt{RND}$, $K_1 \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}^2}$, $\mu_1 \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}}$, and then sends message $(\text{PROVER}_1, sid, C_0, C_1)$ to party V.

When party V is invoked with incoming PROVER_1 message, it randomly selects $K_2 \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}^2}$, and sends message $(\text{VERIFIER}_1, sid, K_2)$ to party P.

When party P is invoked with incoming VERIFIER_1 message, it computes $K \leftarrow K_1 K_2 \bmod \mathsf{n}^2$, $E \leftarrow K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2$, $a \leftarrow \texttt{first}((x, C_0, E), (w, \eta, \zeta); r_a)$ and $c \leftarrow \texttt{COMcom}(pk_{\text{com}}, a; r)$ where $\zeta \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}}$, $r_a, r \xleftarrow{r} \texttt{RND}$, and sends message $(\text{PROVER}_2, sid, K_1, \mu_1, x, E, c)$ to party V.

When party V is invoked with incoming PROVER_2 message, it verifies if $C_1 = \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$ holds. If the equation holds, then it randomly selects $e \xleftarrow{r} \{0,1\}^{\ell_e}$, and sends message $(\text{VERIFIER}_2, sid, e)$ to party P.

When party P is invoked with incoming VERIFIER_2 message, it computes $z \leftarrow \texttt{respond}((x, C_0, E), (w, \eta, \zeta), r_a, e)$, and sends message $(\text{PROVER}_3, sid, a, r, z)$ to party V.

When party V is invoked with incoming PROVER_3 message, it verifies $c = \texttt{COMcom}(pk_{\text{com}}, a; r)$ and $\texttt{verify}((x, C_0, E), a, e, z) = 1$; if both hold, then it returns message $(\text{VERIFIEDSVZK}, sid, x)$ to $\mathcal{Z}$.

Corruption: When party P is invoked with incoming $(\text{CORRUPT}, sid, P)$ by $\mathcal{Z}$, it sends outgoing $(\text{CORRUPTED}, sid, history(P))$ to $\mathcal{Z}$.

Figure 16: Single-verifier zero-knowledge protocol $\pi_{\Sigma(\text{SVZK})}$ for relation R_U in the $\mathcal{F}_{\text{CRS}}$-hybrid world.

tocol with the committing step of mixed-commitment can be found in Figure 15; (3) COM is an equivocal commitment schemes for applying Damgård trick to the non-erasure Sigma protocol.

Theorem C.1. *Given two equivocal commitment schemes* EQC *and* COM*, and a non-erasure Sigma protocol* $\langle \texttt{first}, \texttt{respond}, \texttt{verify}, \texttt{simulate}, \texttt{reconstruct} \rangle$*, the SVZK protocol* $\pi_{\Sigma(\text{SVZK})}$ *in Figure 16 securely realizes* $\mathcal{F}^{R_U}_{\text{SVZK}}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model.*

C.2 Implementation for $\mathcal{F}_{\text{SPZK}}$

Similarly, following the implementation strategy for $\mathcal{F}^{R_S}_{\text{SPZK}}$ in Section 3.2 in the body part, an SPZK instantiation against static adversaries can be found in Figure 18: (1) it is based on an extractable commitment EXC; (2) a Sigma protocol $\langle \texttt{first}, \texttt{respond}, \texttt{verify}, \texttt{simulate} \rangle$, which is not necessary to be non-erasure, is used to show the consistency of the relation R_S and the extractable commitment, i.e. for relation $R'_S = \{(crs, x, E), (w, \zeta) \mid (x, w) \in R_S \wedge E = \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)\}$, and a combination of the Sigma protocol with the committing step of extractable commitment can be found in Figure 17; (3) also, COM is an equivocal commitment schemes for applying Damgård trick to the Sigma protocol.

Theorem C.2. *Given an extractable commitment scheme* EXC*, an equivocal commitment scheme* COM*, and a Sigma protocol* $\langle \texttt{first}, \texttt{respond}, \texttt{verify}, \texttt{simulate} \rangle$*, the SPZK protocol* $\pi_{\Sigma(\text{SPZK})}$ *in Figure 18 securely realizes* $\mathcal{F}^{R_S}_{\text{SPZK}}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model against static adversaries.*

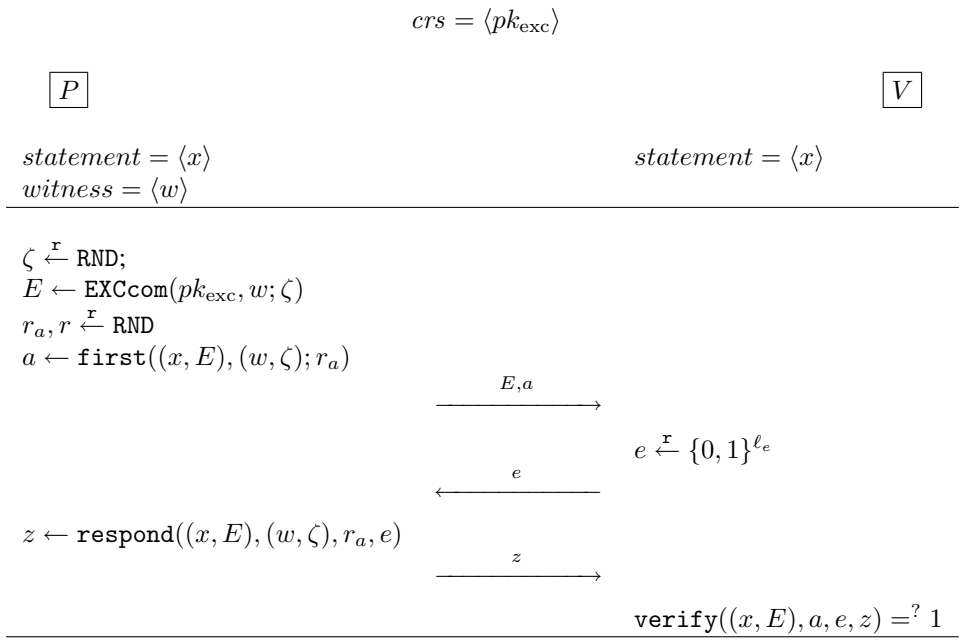

Figure 17: A combination of a committing step with a Sigma protocol for relation $R'_S = \{(crs, x, E), (w, \zeta) \mid (x, w) \in R_S \wedge E = \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)\}$ in the single-prover setting. Here EXCcom is the committing algorithm for extractable commitment scheme EXC.

Protocol $\pi_{\Sigma(\text{SPZK})}$ in the $\mathcal{F}_{\text{CRS}}$-Hybrid World

Proof stage: When party P is invoked with input $(\text{PROVESPZK}, sid, V, x, w)$ by $\mathcal{Z}$, it verifies that $sid = (P, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R_S$, it computes $E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)$, $a \leftarrow \texttt{first}((x, E), (w, \zeta); r_a)$ and $c \leftarrow \texttt{COMcom}(pk_{\text{com}}, a; r)$ where $\zeta, r_a, r \xleftarrow{r} \texttt{RND}$, and then sends message $(\text{PROVER}_1, sid, x, E, c)$ to party V.

When party V is invoked with incoming PROVER_1 message, it randomly selects $e \xleftarrow{r} \{0,1\}^{\ell_e}$, and sends message $(\text{VERIFIER}_1, sid, e)$ to party P.

When party P is invoked with incoming VERIFIER_1 message, it computes $z \leftarrow \texttt{respond}((x, E), (w, \zeta), r_a, e)$, and sends message $(\text{PROVER}_2, sid, a, r, z)$ to party V.

When party V is invoked with incoming PROVER_2 message, it verifies $c = \texttt{COMcom}(pk_{\text{com}}, a; r)$ and $\texttt{verify}((x, E), a, e, z) = 1$; if both hold, then it returns message $(\text{VERIFIEDSPZK}, sid, x)$ to $\mathcal{Z}$.

Corruption: When party P is invoked with incoming $(\text{CORRUPT}, sid, P)$ by $\mathcal{Z}$, it sends outgoing $(\text{CORRUPTED}, sid, history(P))$ to $\mathcal{Z}$.

Figure 18: Single-prover zero-knowledge protocol $\pi_{\Sigma(\text{SPZK})}$ for relation R_S in the $\mathcal{F}_{\text{CRS}}$-hybrid world.

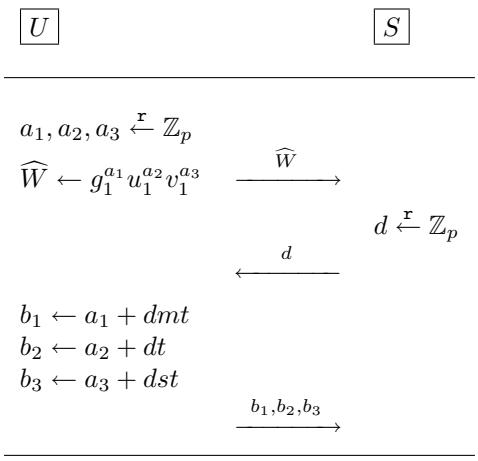

Figure 19: Σ^{R_U}-protocol, where $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$.

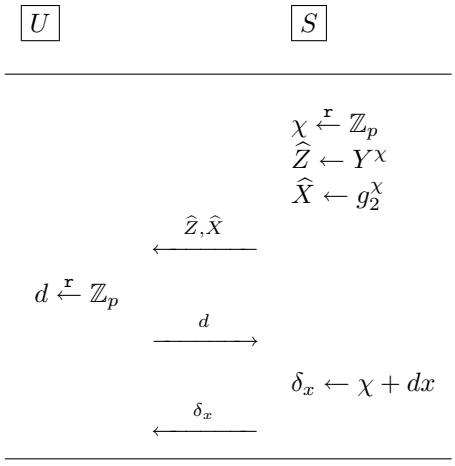

Figure 20: Σ^{R_S}-protocol, where $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (W v_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$.

D More on Concrete Blind Signature Construction

D.1 An Alternative SPZK Protocol

Based on discussed in Section 3.3, a static realization of $\mathcal{F}_{\text{SPZK}}$ with the potential of reducing communication rounds complexity can be found in Figure 21.

Protocol $\pi'_{\Sigma(\text{SPZK})}$ in the $\mathcal{F}_{\text{CRS}}$-Hybrid World

Proof stage: When party P is invoked with input $(\text{PROVESPZK}, sid, V, x, w)$ by $\mathcal{Z}$, it verifies that $sid = (P, sid')$ for some sid'. If not, then ignore the input. Else, if $(x, w) \in R_S$, it computes $E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)$, $a \leftarrow \texttt{first}((x, E), (w, \zeta); r_a)$ and $c \leftarrow \texttt{COMcom}(pk_{\text{com}}, x, E, a; r)$ where $\zeta, r_a, r \xleftarrow{r} \texttt{RND}$, and then sends message $(\text{PROVER}_1, sid, c)$ to party V.

When party V is invoked with incoming PROVER_1 message, it randomly selects $e \xleftarrow{r} \{0,1\}^{\ell_e}$, and sends message $(\text{VERIFIER}_1, sid, e)$ to party P.

When party P is invoked with incoming VERIFIER_1 message, it computes $z \leftarrow \texttt{respond}((x, E), (w, \zeta), r_a, e)$, and sends message $(\text{PROVER}_2, sid, x, E, a, r, z)$ to party V.

When party V is invoked with incoming PROVER_2 message, it verifies $c = \texttt{COMcom}(pk_{\text{com}}, x, E, a; r)$ and $\texttt{verify}((x, E), a, e, z) = 1$; if both hold, then it returns message $(\text{VERIFIEDSPZK}, sid, x)$ to $\mathcal{Z}$.

Corruption: When party P is invoked with incoming $(\text{CORRUPT}, sid, P)$ by $\mathcal{Z}$, it sends outgoing $(\text{CORRUPTED}, sid, history(P))$ to $\mathcal{Z}$.

Figure 21: Single-prover zero-knowledge protocol $\pi'_{\Sigma(\text{SPZK})}$ for relation R_S in the $\mathcal{F}_{\text{CRS}}$-hybrid world.

Theorem D.1. *Given an extractable commitment scheme* EXC*, an equivocal commitment scheme* COM*, and a Sigma protocol* $\langle \texttt{first}, \texttt{respond}, \texttt{verify}, \texttt{simulate} \rangle$*, the SPZK protocol* $\pi'_{\Sigma(\text{SPZK})}$ *in Figure 21 securely realizes* $\mathcal{F}_{\text{SPZK}}^{R_S}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model against static adversaries.*

D.2 Description of the Concrete UC Blind Signature

Based on the design in Section 3.3, we obtain a concrete UC blind signature protocol described in Figure 22 and Figure 23. A step-by-step description of the protocol can be found in the full version. Here we describe the CRS generation, the choice of parameters and also communication efficiency.

Common reference string generation. The common reference string $crs = \{\mathsf{n}, \mathsf{g}; \mathsf{K}; p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2; Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H}\}$. Here $\langle \mathsf{n}, \mathsf{g} \rangle$ is a public key for Paillier encryption; K is a public key for an equivocal commitment; $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$ is a part of public key for Okamoto signature; $\langle Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H} \rangle$ is Pedersen commitments public key. The parameters generated as follows.

First, we generate parameters for Paillier encryption: let p and q be random primes for which it holds $\mathsf{p} \neq \mathsf{q}$, $|\mathsf{p}| = |\mathsf{q}|$ and $\gcd(\mathsf{pq}, (\mathsf{p}-1)(\mathsf{q}-1)) = 1$; let $\mathsf{n} \leftarrow \mathsf{pq}$, and $\mathsf{g} \leftarrow (1+\mathsf{n})$; set $\langle \mathsf{n}, \mathsf{g} \rangle$ as a Paillier public key, and $\langle \mathsf{p}, \mathsf{q} \rangle$ as the X-trapdoor. Second, randomly select $\tau_{\mathsf{K}} \xleftarrow{\mathsf{r}} \mathbb{Z}_{\mathsf{n}}$ and compute $\mathsf{K} \leftarrow (\tau_{\mathsf{K}})^{\mathsf{n}^2} \bmod \mathsf{n}^3$; set K as an E-key, and τ_{K} as the E-trapdoor. Third, let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups as defined in Section A.4. Randomly select $g_2 \xleftarrow{\mathsf{r}} \mathbb{G}_2$, $\tau_{u_2}, \tau_{v_2} \xleftarrow{\mathsf{r}} \mathbb{Z}_p$, compute $u_2 \leftarrow g_2^{\tau_{u_2}}, v_2 \leftarrow g_2^{\tau_{v_2}}$, and set $g_1 \leftarrow \psi(g_2)$, $u_1 \leftarrow \psi(u_2)$ and $v_1 \leftarrow \psi(v_2)$. Set $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$ as the public information, and $\langle \tau_{u_2}, \tau_{v_2} \rangle$ as the trapdoor. Fourth, we generate parameters for a Pedersen-like [Ped91] commitment scheme over an elliptic curve group: let $\mathbf{G} = \langle \mathbf{g} \rangle$ be a cyclic elliptic curve group of prime order Q; select $\tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3} \xleftarrow{\mathsf{r}} \mathbb{Z}_Q$ and compute $\mathbf{h}_2 \leftarrow \mathbf{g}^{\tau_{\mathbf{h}_2}}, \mathbf{h}_3 \leftarrow \mathbf{g}^{\tau_{\mathbf{h}_3}}$; select $\mathcal{H}$ from a collision-resistent hash family $\mathscr{H}$, i.e. $\mathcal{H} \leftarrow \mathscr{H}$, such that $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_Q$; set $\langle Q, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathbf{G}, \mathcal{H} \rangle$ as public information, and $\tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3}$ as the trapdoor. Finally, set $crs = \{\mathsf{n}, \mathsf{g}; \mathsf{K}; p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2; Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H}\}$, and discard the corresponding trapdoors $\{\mathsf{p}, \mathsf{q}; \tau_{\mathsf{K}}; \tau_{u_2}, \tau_{v_2}; \tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3}\}$.

Choice of parameter lengths. Let the length of each parameter p, n, Q be ℓ_p, ℓ_{n}, ℓ_Q respectively. In the protocol, $d_U < p$, $d_S < p$, i.e. $\lambda_U < \ell_p$, $\lambda_S < \ell_p$, and $\kappa > \lambda_0 + \lambda_U + \ell_p + 3$, $\ell_{\mathsf{n}} \geq 3\kappa$. The parameters should be selected so that the following requirements: (i) The 2SDH assumption holds over the bilinear group parameter $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$, (ii) The DLOG assumption holds over the elliptic curve cyclic group $\mathbf{G}$, (iii) The DCR assumption holds over $\mathbb{Z}^*_{\mathsf{n}^2}$. Based on the present state of the art with respect to the solvability of the above problems, a possible choice of the parameters is for example $\ell_p = 171$ bits, $\ell_{\mathsf{n}} = 1024$ bits, $\ell_Q = 171$ bits. Notice that we should avoid using elliptic curves that have small $p+1$ divisors and $p-1$ divisors apart from 2, which suffer from a recent attack on SDH-related assumptions by Cheon [Che06].

Communication efficiency. We count the bits of six flows for generating a UC blind signature as:

flow1: $3\ell_{\mathsf{n}} + \ell_p$
flow2: $2\ell_{\mathsf{n}}$
flow3: $2\ell_{\mathsf{n}} + \ell_{\mathsf{n}} + 2\ell_{\mathsf{n}} + \ell_Q$
flow4: $2\ell_{\mathsf{n}} + \ell_Q + \lambda_U$
flow5: $3(\lambda_0 + \lambda_U + \ell_p + 1) + \ell_{\mathsf{n}} + 2\ell_{\mathsf{n}} + \ell_p + \ell_Q + \lambda_S$
flow6: $(\lambda_0 + \lambda_S + \ell_p + 1) + \ell_{\mathsf{n}} + 5\ell_p + 2\ell_{\mathsf{n}} + \ell_Q$

Based on the parameters: $\lambda_0 = \lambda_U = \lambda_S = 80$bits, $\ell_p = 171$bits, $\ell_{\mathsf{n}} = 1024$bits, $\ell_Q = 171$bits, $\kappa = 341$bits, and each element in $\mathbb{G}_1$ is with length of ℓ_p, we can compute the whole communication which is about 22.3 Kbits, i.e. less than 3 Kbytes.

Signature length. The length of signature $\sigma = \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle$ is: $\ell_p + 6 \cdot \ell_p + \ell_p + \ell_p + 6 \cdot \ell_p = 15\ell_p = 2565$bits, i.e. about 2.6 Kbits. Here using the families of curves in [BLS04], we use group $\mathbb{G}_1$ where each element is 171bits and group $\mathbb{G}_2$ where each element 6×171bits.

$$crs = \langle \mathsf{n}, \mathsf{g}; \mathsf{K}; p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2; Q, \mathbf{G}, \mathbf{g}, \mathbf{h}_2, \mathbf{h}_3, \mathcal{H} \rangle$$

$\boxed{U}$		$\boxed{S}$
$vk = \langle X = g_2^x \rangle$ $msg = \langle m \rangle, m \in \mathbb{Z}_p$		$vk = \langle X = g_2^x \rangle$ $sk = \langle x \rangle$
$K_1 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}^2}$; $\mu_1 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}}$; $C_1 \leftarrow \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$ $t, s \xleftarrow{\mathsf{r}} \mathbb{Z}_p$; $W \leftarrow g_1^{mt} u_1^t v_1^{st}$		
	$\xrightarrow{C_1, W}$	
		$K_2 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}^2}$
	$\xleftarrow{K_2}$	
$K \leftarrow K_1 K_2 \bmod \mathsf{n}^2$; $\theta \leftarrow (mt \bmod p) + t2^\kappa + (st \bmod p)2^{2\kappa}$ $A_\theta, B_\theta \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}}$ $E_\theta \leftarrow K^\theta (A_\theta)^{\mathsf{n}} \bmod \mathsf{n}^2$ $a_i \xleftarrow{\mathsf{r}} \pm[0, 2^{\lambda_0 + \lambda_U + \ell_p}]$, $i = 1, 2, 3$ $\vartheta \leftarrow a_1 + a_2 2^\kappa + a_3 2^{2\kappa}$ $\widehat{E}_\theta \leftarrow K^\vartheta (B_\theta)^{\mathsf{n}} \bmod \mathsf{n}^2$ $\widehat{W} \leftarrow g_1^{a_1} u_1^{a_2} v_1^{a_3}$ $\mu_2 \xleftarrow{\mathsf{r}} \mathbb{Z}_Q$; $\omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$ $C_2 \leftarrow \mathbf{g}^{\omega_2} \mathbf{h}_2^{\mu_2}$		
	$\xrightarrow{\langle K_1, \mu_1 \rangle, \langle E_\theta, C_2 \rangle}$	
		$C_1 =^? \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$ $d_U \xleftarrow{\mathsf{r}} \{0,1\}^{\lambda_U}$ $r \xleftarrow{\mathsf{r}} \mathbb{Z}_p$ s.t. $x + r \not\equiv 0 \bmod p$ $\chi \xleftarrow{\mathsf{r}} \pm[0, 2^{\lambda_0 + \lambda_S + \ell_p}]$; $A_x, B_x \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}}$, $l \xleftarrow{\mathsf{r}} \mathbb{Z}_p$ $Y \leftarrow (W v_1^l)^{\frac{1}{x+r}}$, $\widehat{Z} \leftarrow Y^\chi$ $\widehat{X} \leftarrow g_2^\chi$ $E_x \leftarrow \mathsf{g}^x (A_x)^{\mathsf{n}} \bmod \mathsf{n}^2$ $\widehat{E}_x \leftarrow \mathsf{g}^\chi (B_x)^{\mathsf{n}} \bmod \mathsf{n}^2$ $\mu_3 \xleftarrow{\mathsf{r}} \mathbb{Z}_Q$; $\omega_3 \leftarrow \mathcal{H}(Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x)$ $C_3 \leftarrow \mathbf{g}^{\omega_3} \mathbf{h}_3^{\mu_3}$
	$\xleftarrow{\langle E_x, C_3 \rangle, d_U}$	

(to be continued in Figure 23)

Figure 22: Blind signature generation protocol (part 1).

U		S
(continued from Figure 22)		
$b_1 \leftarrow a_1 + d_U \cdot (mt \bmod p)$ $b_2 \leftarrow a_2 + d_U \cdot t$ $b_3 \leftarrow a_3 + d_U \cdot (st \bmod p)$ $F_\theta \leftarrow B_\theta(A_\theta)^{d_U} \bmod \mathsf{n}$ $d_S \xleftarrow{\mathsf{r}} \{0,1\}^{\lambda_S}$	$\xrightarrow{\langle b_1,b_2,b_3,F_\theta\rangle,\langle \widehat{E}_\theta,\widehat{W},\mu_2\rangle,d_S}$	
		$\omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$; $C_2 =^? \mathbf{g}^{\omega_2}\mathbf{h}_2^{\mu_2}$ $E_\theta \in^? \mathbb{Z}^*_{\mathsf{n}^2}$, $W \in^? \mathbb{G}_1$ $b_i \in^? \pm[0, 2^{\lambda_0+\lambda_U+\ell_p+1}]$, $i = 1,2,3$ $g_1^{b_1} u_1^{b_2} v_1^{b_3} =^? \widehat{W} W^{d_U}$ $K \leftarrow K_1 K_2 \bmod \mathsf{n}^2$ $\eta \leftarrow b_1 + b_2 2^\kappa + b_3 2^{2\kappa}$ $K^\eta (F_\theta)^{\mathsf{n}} =^? \widehat{E}_\theta (E_\theta)^{d_U} \bmod \mathsf{n}^2$ $\delta_x \leftarrow \chi + d_S \cdot x$, $F_x \leftarrow B_x(A_x)^{d_S} \bmod \mathsf{n}$
	$\xleftarrow{\langle \delta_x,F_x\rangle,\langle Y,l,r,\widehat{Z},\widehat{X},\widehat{E}_x,\mu_3\rangle}$	
$\omega_3 \leftarrow \mathcal{H}(Y,l,r,\widehat{Z},\widehat{X},\widehat{E}_x)$ $C_3 =^? \mathbf{g}^{\omega_3}\mathbf{h}_3^{\mu_3}$ $E_x \in^? \mathbb{Z}^*_{\mathsf{n}^2}$, $\delta_x \in^? \pm[0, 2^{\lambda_0+\lambda_S+\ell_p+1}]$ $g_2^{\delta_x} =^? \widehat{X} X^{d_S}$ $\mathbf{g}^{\delta_x}(F_x)^{\mathsf{n}} =^? \widehat{E}_x (E_x)^{d_S} \bmod \mathsf{n}^2$ $Y^{\delta_x} =^? \widehat{Z}(W v_1^l Y^{-r})^{d_S}$ $f, h \xleftarrow{\mathsf{r}} \mathbb{Z}_p$; $\varsigma \leftarrow Y^{\frac{1}{fl}} \bmod p$ $\alpha \leftarrow X^f g_2^{fr}$; $\beta \leftarrow s + \frac{l}{t} \bmod p$ $V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h$; $V_2 \leftarrow X^{fh+r} g_2^{frh}$ $\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2\rangle$ $\mathtt{verify}(crs, vk, m, \sigma) =^? 1$		
output $(m;\sigma)$		

Figure 23: Blind signature generation protocol (part 2).

E Proofs in the Body Part

E.1 Proof for Theorem 2.4

Proof. **(I)** (sketch) Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to attack the signature scheme $\mathtt{SIG}$ to produce a forgery. Note that $\mathcal{B}$ is given vk, and is allowed to query the signing oracle with $\mathbf{u}_i$ and obtain $\mathbf{s}_i$ such that $\mathtt{SIGverify}(vk, \mathbf{u}_i, \mathbf{s}_i) = 1$. The goal of $\mathcal{B}$ is to obtain a valid pair $\langle \mathbf{u}^*, \mathbf{s}^* \rangle$ where $\mathbf{u}^*$ is not queried.

$\mathcal{B}$ runs a copy of $\mathcal{A}$ inside, and supplies $\mathcal{A}$ with crs and vk; note that $\mathcal{A}$ is allowed to query $(m_i, \rho_{1,i})$ where $\mathbf{u}_i = \mathtt{EQCcom}(pk_{\text{eqc}}, m_i; \rho_{1,i})$; given such query, $\mathcal{B}$ queries his signing oracle with $\mathbf{u}_i$, obtains $\mathbf{s}_i$ and then gives such $\mathbf{s}_i$ to $\mathcal{A}$, where $\mathtt{SIGverify}(vk, \mathbf{u}_i, \mathbf{s}_i) = 1$. At some point, $\mathcal{A}$ produces a pair $\langle m^*, \sigma^* \rangle$ where $\sigma^* = E^* || \varpi^*$, and $\mathtt{NIZKverify}(crs, vk, E^*, m^*; \varpi^*) = 1$. Under the soundness of NIZK, there exist $\langle \mathbf{u}^*, \mathbf{s}^*, \rho_1^*, \rho_3^* \rangle$ such that $((crs, vk, E^*, m^*), (\mathbf{u}^*, \mathbf{s}^*, \rho_1^*, \rho_3^*)) \in R$. Given that commitment scheme $\mathtt{EQC}$ is binding, we have only negligible probability to find $m_i' \neq m_i$ such that $\mathbf{u}_i = \mathtt{EQCcom}(pk_{\text{eqc}}, m_i; \rho_{1,i}) = \mathtt{EQCcom}(pk_{\text{eqc}}, m_i'; \rho_{1,i}')$. Therefore m^* cannot be based on any queried $\mathbf{u}_i$. By using the extractable trapdoor xk_{exc} of commitment scheme $\mathtt{EXC}$, $\mathcal{B}$ can extract $\mathbf{u}^*$ and $\mathbf{s}^*$ from E^*. Note that $\mathtt{SIGverify}(vk, \mathbf{u}^*, \mathbf{s}^*) = 1$, and $\mathbf{u}^*$ has never been queried, $\mathcal{B}$ can obtain a forgery $\langle \mathbf{u}^*, \mathbf{s}^* \rangle$ for $\mathtt{SIG}$.

(II) Here we prove the lite blind signature in Figure 2.4 satisfies equivocality defined in Definition 2.3. Generate $(crs, \tau) \leftarrow \mathtt{CRSgen}(1^\lambda)$; here $crs = \langle pk_{\text{eqc}}, pk_{\text{exc}}, crs_{\text{nizk}} \rangle$, $\tau = \langle ek_{\text{eqc}}, \tau_{\text{nizk}} \rangle$ where ek_{eqc} is the equivocal key for commitment scheme $\mathtt{EQC}$ and τ_{nizk} is the trapdoor for state reconstruction of $\mathtt{NIZK}$. Send crs to $\mathcal{A}$.

We first describe how oracle $Users(crs, \cdot)$ operates.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, select $\rho_1 \xleftarrow{\text{r}} \mathtt{RND}$ and compute $\mathbf{u} \leftarrow \mathtt{EQCcom}(pk_{\text{eqc}}, m; \rho_1)$, record $\langle i, m, vk, \mathbf{u}, \rho_1 \rangle$ into $history_i$, and return message $(i, \mathbf{u})$ to $\mathcal{A}$.

- Upon receiving message $(i, \mathbf{s}, \rho_2, sk)$ from $\mathcal{A}$, if there exists a record $\langle i, m, vk, \mathbf{u}, \rho_1 \rangle$ in $history_i$ and $(vk, sk) \in \mathsf{KeyPair}$ and $\mathbf{s} = \mathtt{SIGsign}(vk, sk, \mathbf{u}; \rho_2)$, then select $\rho_3, \rho_4 \xleftarrow{\text{r}} \mathtt{RND}$ and compute $E \leftarrow \mathtt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)$, and $\varpi \leftarrow \mathtt{NIZKprove}((crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \rho_1, \rho_3); \rho_4 : \mathbf{u} = \mathtt{EQCcom}(pk_{\text{eqc}}, m; \rho_1) \wedge \mathtt{SIGverify}(vk, \mathbf{u}, \mathbf{s}) = 1 \wedge E = \mathtt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3))$ and set $\sigma \leftarrow E || \varpi$. If $\mathtt{verify}(crs, vk, m, \sigma) = 1$, then update $\langle i, m, vk, \mathbf{u}, \rho_1 \rangle$ in $history_i$ into $\langle i, m, vk, \mathbf{u}, \sigma, \rho_1, \rho_3, \rho_4 \rangle$, and return $\mathcal{A}$ with message (i, σ); otherwise return $\mathcal{A}$ with message $(i, \perp)$.

- Upon receiving message (i, open), return $\mathcal{A}$ with message $(i, history_i)$.

Next we construct $\mathcal{I}(crs, \tau, \cdot)$.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, run $(\mathbf{u}, aux) \leftarrow \mathcal{I}_1(crs, \tau, vk)$, i.e. compute $(\mathbf{u}, state_{\text{eqc}}) \leftarrow \mathtt{EQCfake}(pk_{\text{eqc}}, ek_{\text{eqc}})$, and record $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}} \rangle$ into $temp$, and return message $(i, \mathbf{u})$ to $\mathcal{A}$; here $\mathtt{EQCfake}$ is an algorithm of producing a fake commitment based on $\mathtt{EQC}$'s committing key pk_{eqc}.

- Upon receiving message $(i, \mathbf{s}, \rho_2, sk)$ from $\mathcal{A}$, if there exists a record $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}} \rangle$ in $temp$ and $(vk, sk) \in \mathsf{KeyPair}$ and $\mathbf{s} = \mathtt{SIGsign}(vk, sk, \mathbf{u}; \rho_2)$; then compute $\sigma \leftarrow E || \varpi$, where $E \leftarrow \mathtt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)$ and $(\varpi, state_{\text{nizk}}) \leftarrow \mathtt{NIZKsimulate}((crs, vk, E, m), \tau_{\text{nizk}})$ and $\rho_3 \xleftarrow{\text{r}} \mathtt{RND}$. Update $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}} \rangle$ in $temp$ into $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}}; \mathbf{s}, sk, \rho_2; E, \varpi, \rho_3, state_{\text{nizk}} \rangle$, and return message (i, σ) to $\mathcal{A}$; otherwise return $\mathcal{A}$ with message $(i, \perp)$; here $\mathtt{NIZKsimulate}$ is used to produce a simulated proof for $\mathtt{NIZK}$.

- Upon receiving message (i, open),

  - if there exists a record $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}}\rangle$ in $temp$ then run $\rho_1 \leftarrow \mathcal{I}_2(i, temp)$, i.e. compute $\rho_1 \leftarrow \texttt{EQCequivocate}(pk_{\text{eqc}}, \mathbf{u}, state_{\text{eqc}}, m)$; and record $\langle i, m, vk, \mathbf{u}, \rho_1\rangle$ into $history_i$, and return $\mathcal{A}$ with message $(i, history_i)$; here $\texttt{EQCequivocate}$ is equivocation algorithm associated with the equivocal commitment $\texttt{EQC}$;
  - if there exists a record $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}}; \mathbf{s}, sk, \rho_2; E, \varpi, \rho_3, state_{\text{nizk}}\rangle$ in $temp$, then run $\rho_4 \leftarrow \mathcal{I}_2(i, temp)$, i.e. compute $\rho_1 \leftarrow \texttt{EQCequivocate}(pk_{\text{eqc}}, \mathbf{u}, state_{\text{eqc}}, m)$ and compute $\rho_4 \leftarrow \texttt{NIZKreconstruct}((crs, vk, E, m), \varpi, state_{\text{nizk}}, (\mathbf{u}, \mathbf{s}, \rho_1, \rho_3))$, and then record $\langle i, m, vk, \mathbf{u}, \sigma, \rho_1, \rho_3, \rho_4\rangle$ into $history_i$, and return $\mathcal{A}$ with message $(i, history_i)$; here $\texttt{NIZKreconstruct}$ is state reconstruction algorithm associated with $\texttt{NIZK}$.

In the interaction with oracle $Users(crs, \cdot)$, or with oracle $\mathcal{I}(crs, \tau, \cdot)$, $\langle \rho_1, \rho_3, \rho_4\rangle$ have the same distributions. Based on the construction of $\mathcal{I}$ we know that in the two experiments $\mathbf{u}$ has exactly the same distributions, and ϖ and E cannot be distinguished based on the zero-knowledge property of $\texttt{NIZK}$ and hiding property of $\texttt{EXC}$ respectively. Therefore $\mathcal{A}$ will output 1 with the same probability except negligible probability in the two interactions. □

E.2 Proof for Theorem 2.5

Proof. **(I)** For lite-unforgeability, we can follow similarly the unforgeability proof of Okamoto signature (refer to the proof of theorem 2, page 14, in [Oka06]). Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to break the 2SDH assumption.

We consider two types of forgers, Type-1 forger and Type-2 forger. Let $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathrm{e}}, \psi, u_2, v_2, X\rangle$ be the verification key for the forger $\mathcal{A}$ and $v_2 = g_2^z$. Now $\mathcal{A}$ queries with $\langle m_i, t_i, s_i\rangle$ and is responded with $\langle Y_i, l_i, r_i\rangle$ for $i = 1, \ldots, L$. The two types of forgers are:

- Type-1 forger: outputs a forgery $\langle m^*, \sigma^*\rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^*\rangle$, and $m^* + \beta^* z \not\equiv m_i + \beta_i z$ for $i = 1, \ldots, L$.
- Type-2 forger: outputs a forgery $\langle m^*, \sigma^*\rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^*\rangle$, and $m^* + \beta^* z \equiv m_i + \beta_i z$ for $i = 1, \ldots, L$.

$\mathcal{B}$ operates as follows:

1. $\mathcal{B}$ is given $\langle g_1, g_2, A, B, C_1, \ldots, C_L, a_1, \ldots, a_L, b_1, \ldots, b_L\rangle$ where $A = g_2^x, B = g_2^y, C_i = g^{\frac{y+b_i}{x+a_i}}$, $i = 1, \ldots, L$.
2. $\mathcal{B}$ randomly selects $c_{\text{type}} \in \{1, 2\}$.
3. If $c_{\text{type}} = 1$: $\mathcal{B}$ randomly selects $z \in \mathbb{Z}_p$, and $\mathcal{B}$ sets $X \leftarrow A = g_2^x$, $u_2 \leftarrow B = g_2^y$, and $v_2 \leftarrow g_2^z$; $\mathcal{B}$ then gives $\langle g_1, g_2, u_2, v_2, X\rangle$ to $\mathcal{A}$ as the verification key. When $\mathcal{A}$ queries with $\langle m_i, t_i, s_i\rangle$, $\mathcal{B}$ computes $\beta_i \leftarrow \frac{b_i - m_i}{z}$, $r_i \leftarrow a_i$, $\varsigma \leftarrow \psi(C_i) = g_1^{\frac{y+b_i}{x+a_i}} = g_i^{\frac{m_i + y + \beta_i z}{x + r_i}}$, and $Y_i \leftarrow \varsigma_i^{t_i}$, $l_i \leftarrow t_i(\beta_i - s_i)$. Then $\mathcal{B}$ returns $\langle Y_i, l_i, r_i\rangle$. When $\mathcal{A}$ outputs a successful forgery $\langle m^*, \sigma^*\rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^*\rangle$, $\mathcal{B}$ checks $m^* + \beta^* z \not\equiv m_i + \beta_i z$ for all $i = 1, \ldots, L$. If not, $\mathcal{B}$ outputs $\texttt{failure}$ and aborts. Else $\mathcal{B}$ sets $b^* \leftarrow m^* + \beta^* z$, and outputs $\langle \varsigma^*, \alpha^*, b^*, V_1^*, V_2^*\rangle$.

4. If $c_{\text{type}} = 2$: $\mathcal{B}$ randomly selects $x', y' \in \mathbb{Z}_p$ and computes $X \leftarrow g_2^{x'}$, $u_2 \leftarrow g_2^{y'}$, $v_2 \leftarrow A = g_2^x$. $\mathcal{B}$ then gives $\langle g_1, g_2, u_2, v_2, X\rangle$ to $\mathcal{A}$ as the verification key. The simulation of signing oracle can be achieved since $\mathcal{B}$ knows x'. When $\mathcal{A}$ outputs a successful forgery $\langle m^*, \sigma^*\rangle$ where $\sigma^* = \langle \varsigma^*, \alpha^*, \beta^*, V_1^*, V_2^*\rangle$, $\mathcal{B}$ computes $z^* \leftarrow \frac{m_i - m^*}{\beta^* - \beta_i}$ for all $i = 1, \ldots, L$ such that $\beta_i \neq \beta^*$, and checks whether $A = g_2^{z^*}$. If it holds, $z^* = x$. $\mathcal{B}$ then outputs $\langle \varsigma, \alpha, d, V_1, V_2\rangle$ where $c, d, \eta \xleftarrow{\mathtt{r}} \mathbb{Z}_p$, $\varsigma \leftarrow (\psi(B))^{\frac{1}{x+c}} g_1^{\frac{d}{x+d}} = (\psi(g_2)^y)^{\frac{1}{x+c}} g_1^{\frac{d}{x+d}} = (g_1^y)^{\frac{1}{x+c}} g_1^{\frac{d}{x+d}} = g_1^{\frac{y+d}{x+c}}$, $\alpha \leftarrow g_2^c$, $V_1 \leftarrow \psi(A) g_1^{\eta}$, $V_2 \leftarrow A^{\eta+c} g_2^{c\eta}$. Note that if $m^* + \beta^* x = m_i + \beta_i x$ and $\beta^* \neq \beta_i$ for some $i \in \{1, \ldots, L\}$. Then $x = \frac{m_i - m^*}{\beta^* - \beta_i}$.

This completes the description of $\mathcal{B}$.

(II) Here we prove the lite blind signature based on Okamoto signature satisfies equivocality defined in Definition 2.3. Generate $(crs, \tau) \leftarrow \mathtt{CRSgen}(1^\lambda)$; here $crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2\rangle$, $\tau = \langle \tau_{u_1}, \tau_{v_1}\rangle$ such that $u_1 = g_1^{\tau_{u_1}}, v_1 = g_1^{\tau_{v_1}}$. Send crs to $\mathcal{A}$.

We first describe the oracle $Users(crs, \cdot)$.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, where $vk = \langle X\rangle$ select $t, s \xleftarrow{\mathtt{r}} \mathbb{Z}_p$ and compute $W \leftarrow g_1^{mt} u_1^t v_1^{st}$, record $(i, m, vk, W, \langle t, s\rangle)$ into memory $history_i$, and return message (i, W) to $\mathcal{A}$.

- Upon receiving message $(i, \langle Y, l, r\rangle, sk)$ from $\mathcal{A}$, where $sk = \langle x\rangle$, if there exists a record $(i, m, vk, W, \langle t, s\rangle)$ in $history_i$ and $g_2^x = X$, then compute $\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2\rangle$ where $\varsigma \leftarrow Y^{\frac{1}{ft}}$, $\alpha \leftarrow X^f g_2^{fr}$, $\beta \leftarrow s + \frac{l}{t}$, $V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h$, $V_2 \leftarrow X^{fh+r} g_2^{frh}$, and $f, h \xleftarrow{\mathtt{r}} \mathbb{Z}_p$. If $\mathtt{verify}(crs, vk, m, \sigma) = 1$, then update $(i, m, vk, W, \langle t, s\rangle)$ in $history_i$ into $(i, m, vk, W, \langle \varsigma, \alpha, \beta, V_1, V_2\rangle, \langle t, s\rangle, \langle f, h\rangle)$, and return $\mathcal{A}$ with message (i, σ); otherwise return $\mathcal{A}$ with message $(i, \perp)$.

- Upon receiving message (i, open), return $\mathcal{A}$ with message $(i, history_i)$.

Next we construct $\mathcal{I}(crs, \tau, \cdot)$.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, run $(W, aux) \leftarrow \mathcal{I}_1(crs, \tau, vk)$ as: select $\widetilde{m}, \widetilde{t}, \widetilde{s} \xleftarrow{\mathtt{r}} \mathbb{Z}_p$, and compute $W \leftarrow g_1^{\widetilde{m}\widetilde{t}} u_1^{\widetilde{t}} v_1^{\widetilde{s}\widetilde{t}}$, record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}, \widetilde{s}\rangle)$ into $temp$, and return message (i, W) to $\mathcal{A}$.

- Upon receiving message $(i, \langle Y, l, r\rangle, sk)$ from $\mathcal{A}$, check whether the following conditions:

  - $Y \neq 1$
  - there exists a record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}, \widetilde{s}\rangle)$ in $temp$ where $g_2^x = X$ and $Y = (W v_1^l)^{\frac{1}{x+r}}$ hold,

  then compute $\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2\rangle$, where $\varsigma \leftarrow (g_1^m u_1 v_1^{\beta})^{\frac{1}{\widehat{f}x+\widehat{r}}}$, $\alpha \leftarrow g_2^{\widehat{f}x+\widehat{r}}$, $V_1 \leftarrow \psi(X)^{\frac{1}{\widehat{f}}} g_1^{\widehat{h}}$, $V_2 \leftarrow X^{\widehat{f}\widehat{h}+\frac{\widehat{r}}{\widehat{f}}} g_2^{\widehat{r}\widehat{h}}$, and $\beta, \widehat{f}, \widehat{r}, \widehat{h} \xleftarrow{\mathtt{r}} \mathbb{Z}_p$. If $\mathtt{verify}(crs, vk, m, \sigma) = 1$ then update $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}, \widetilde{s}\rangle)$ in $temp$ into $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}, \widetilde{s}\rangle; \langle Y, l, r\rangle, sk; \langle \varsigma, \alpha, \beta, V_1, V_2\rangle, \langle \widehat{f}, \widehat{r}, \widehat{h}\rangle)$, and return message (i, σ) to $\mathcal{A}$; otherwise return $\mathcal{A}$ with message $(i, \perp)$.

- Upon receiving message (i, open),

– if there exists a record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}, \widetilde{s} \rangle)$ in $temp$ then obtain $\langle t, s \rangle \leftarrow \mathcal{I}_2(i, temp)$ where $\mathcal{I}_2$ is defined as: select $t \xleftarrow{\mathtt{r}} \mathbb{Z}_p$ and compute s from $(m + \tau_{u_2} + s\tau_{v_2})t = (\widetilde{m} + \tau_{u_2} + \widetilde{s}\tau_{v_2})\widetilde{t}$, i.e. $s \leftarrow \frac{(\widetilde{m}+\tau_{u_2}+\widetilde{s}\tau_{v_2})\widetilde{t}-(m+\tau_{u_2})t}{\tau_{v_2}t}$, and record $(i, m, vk, W, \langle t, s \rangle)$ into $history_i$, and return $\mathcal{A}$ with message $(i, history_i)$;

– if there exists a record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}, \widetilde{s} \rangle; \langle Y, l, r \rangle, sk; \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle, \langle \widehat{f}, \widehat{r}, \widehat{h} \rangle)$ in $temp$, then run $(\langle t, s \rangle, \langle f, h \rangle) \leftarrow \mathcal{I}_2(i, temp)$ as:

  * compute t, s based on equations $(m + \tau_{u_2} + s\tau_{v_2})t = (\widetilde{m} + \tau_{u_2} + \widetilde{s}\tau_{v_2})\widetilde{t}$ and $\beta = s + \frac{l}{t}$; and we have $t \leftarrow \frac{(\widetilde{m}+\tau_{u_2}+\widetilde{s}\tau_{v_2})\widetilde{t}+\tau_{v_2}l}{m+\tau_{u_2}+\beta\tau_{v_2}}$, and $s \leftarrow \beta - l\frac{m+\tau_{u_2}+\beta\tau_{v_2}}{(\widetilde{m}+\tau_{u_2}+\widetilde{s}\tau_{v_2})\widetilde{t}+\tau_{v_2}l}$;
  * compute f based on equation $\widehat{f}x + \widehat{r} = f(x + r)$, and compute h based on equation $\frac{x}{\widehat{f}} + \widehat{h} = \frac{x}{f} + h$; and we have $f \leftarrow \frac{\widehat{f}x+\widehat{r}}{x+r}$ and $h \leftarrow \frac{x}{\widehat{f}} + \widehat{h} - \frac{x(x+r)}{\widehat{f}x+\widehat{r}}$;

  and record $(i, m, vk, W, \sigma, \langle t, s \rangle, \langle f, h \rangle)$ into $history_i$, and return $\mathcal{A}$ with message $(i, history_i)$.

We define $\mathbf{E}$ as the event that upon receiving message (i, open), the coins cannot be reconstructed by the oracle $\mathcal{I}(crs, \tau, \cdot)$. If event $\mathbf{E}$ does not occur, then the adversary cannot distinguish the interaction with the oracle $Users(crs, \cdot)$ and the interaction with the oracle $\mathcal{I}(crs, \tau, \cdot)$ because the adversary's views have the same distribution. Next we still need to argue that the probability of the event $\mathbf{E}$ is negligible.

- In the case that $\mathcal{A}$ expects $(i, m, vk, W, \langle t, s \rangle)$:

  – $\mathcal{I}(crs, \tau, \cdot)$ can reconstruct $\langle t, s \rangle$ except that $t = 0$; note that t is randomly selected from $\mathbb{Z}_p$ and $\Pr[t \xleftarrow{\mathtt{r}} \mathbb{Z}_p : t = 0] = 1/p$ which is negligible.

- In the case that $\mathcal{A}$ expects $(i, m, vk, W, \sigma, \langle t, s \rangle, \langle f, h \rangle)$:

  – t can be reconstructed except that $m + \tau_{u_2} + \beta\tau_{v_2} = 0$; note that β is randomly selected from $\mathbb{Z}_p$ after $\mathcal{I}(crs, \tau, \cdot)$ receiving m from $\mathcal{A}$, and $\Pr[\beta \xleftarrow{\mathtt{r}} \mathbb{Z}_p : m + \tau_{u_2} + \beta\tau_{v_2} = 0] = 1/p$ which is negligible;
  – when t has been reconstructed, s can always be reconstructed because $(\widetilde{m} + \tau_{u_2} + \widetilde{s}\tau_{v_2})\widetilde{t} + \tau_{v_2}l \neq 0$; by contradiction, if $(\widetilde{m} + \tau_{u_2} + \widetilde{s}\tau_{v_2})\widetilde{t} + \tau_{v_2}l = 0$, then $Wv_1^l = 1$ which means $Y = 1$; however when $Y = 1$, message $(i, \perp)$ will be returned, and no effort for constructing the coins now; (the reason of excluding $Y = 1$ is that in the oracle $Users(crs, \cdot)$, when $\varsigma = 1$, no signature will be generated; $\varsigma = Y^{\frac{1}{ft}}$, which means when $Y = 1$ no signature will be produced.)
  – note that $f(x + r) \neq 0$; otherwise in oracle $Users(crs, \cdot)$, $\alpha = X^f g_2^{fr} = g_2^{f(x+r)} = 1$, and no signature will be generated. So $x + r \neq 0$ and $\widehat{f}x + \widehat{r} \neq 0$ and f can always be reconstructed, and h can be reconstructed when $\widehat{f} \neq 0$; notice that $\widehat{f}$ is randomly selected from $\mathbb{Z}_p$, and $\Pr[\widehat{f} \xleftarrow{\mathtt{r}} \mathbb{Z}_p : \widehat{f} = 0] = 1/p$ which is negligible;

Based on the argument above, the event $\mathbf{E}$ can happen with only negligible probability; so the adversary $\mathcal{A}$ cannot distinguish the interactions with the oracle $Users(crs, \cdot)$ and with the oracle $\mathcal{I}(crs, \tau, \cdot)$. □

E.3 Proof for Theorem 3.1

E.3.1 A Lemma

Before the proof of Theorem 3.1, we introduce a useful lemma as below which will help us to organize the proof. Note that we can extend such lemma to general setting.

Lemma E.1. *Assume that*

(1) $\exists \mathcal{S}_1, \forall \mathcal{Z}$, $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{SVZK}}^{R_U}, \mathcal{F}_{\mathrm{SPZK}}^{R_S}}_{\pi_{\Sigma(\mathrm{BSIG})}, \mathcal{Z}} = \mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}$, *where* $\mathcal{F}_1$ *is dummy blind signature functionality (cf. Figure 24);*

(2) for $\mathcal{S}_i$, $\exists \mathcal{S}_{i+1}$, $\forall \mathcal{Z}$, $\left|\mathrm{EXEC}^{\mathcal{F}_i}_{\pi_d, \mathcal{S}_i, \mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_{i+1}}_{\pi_d, \mathcal{S}_{i+1}, \mathcal{Z}}\right| \leq \epsilon_i$, *where* $1 \leq i \leq 4$;

(3) $\mathcal{F}_5 = \mathcal{F}_{\mathrm{BSIG}}$.

Then we have $\exists \mathcal{S}_5$, $\forall \mathcal{Z}$, $\left|\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{SVZK}}^{R_U}, \mathcal{F}_{\mathrm{SPZK}}^{R_S}}_{\pi_{\Sigma(\mathrm{BSIG})}, \mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_{\mathrm{BSIG}}}_{\pi_d, \mathcal{S}_5, \mathcal{Z}}\right| \leq \sum_{i=1}^{4} \epsilon_i$.

Dummy Blind Signature Functionality $\mathcal{F}_1$

Key generation: Upon receiving $(\textsc{KeyGen}, sid)$ from party S, verify that $sid = (S, sid')$ for some sid'. If not, ignore the input. Else, forward $(\textsc{KeyGen}, sid)$ to the adversary $\mathcal{S}$.

Upon receiving $(\textsc{VerificationAlg}, sid, \mathsf{ver})$ from the adversary $\mathcal{S}$, record ver in $history(S)$, and output $(\textsc{VerificationAlg}, sid, \mathsf{ver})$ to party S, where ver is a verification algorithm.

Signature generation: Upon receiving $(\textsc{Sign}, sid, m, \mathsf{ver}')$ from party $U \neq S$, where $sid = (S, sid')$, record $\langle m, \mathsf{ver}' \rangle$ in $history(U)$, and send $(\textsc{Sign}, sid, U, m, \mathsf{ver}')$ to the adversary $\mathcal{S}$.

Upon receiving $(\textsc{SignStatus}, sid, U, \mathsf{completed})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, output $(\textsc{SignStatus}, sid, U, \mathsf{completed})$ to party S, and record $\langle U, \mathsf{completed} \rangle$ in $history(S)$.

Upon receiving $(\textsc{SignStatus}, sid, U, \mathsf{incompleted})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, output $(\textsc{SignStatus}, sid, U, \mathsf{incompleted})$ to party S.

Upon receiving $(\textsc{Signature}, sid, U, m, \sigma)$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, output $(\textsc{Signature}, sid, m, \sigma)$ to party U, and record σ next to $\langle m, \mathsf{ver}' \rangle$ inside $history(U)$.

Upon receiving $(\textsc{Signature}, sid, U, \perp)$ from the adversary$\mathcal{S}$, where U is a user that has requested a signature, output $(\textsc{Signature}, sid, \perp)$ to party U.

Signature verification: Upon receiving $(\textsc{Verify}, sid, m, \sigma, \mathsf{ver}')$ from party V, where $sid = (S, sid')$, send $(\textsc{Verify}, sid, V, m, \sigma, \mathsf{ver}')$ to the adversary $\mathcal{S}$.

Upon receiving $(\textsc{Verified}, sid, V, \phi)$ from the adversary $\mathcal{S}$, output $(\textsc{Verified}, sid, \phi)$ to party V.

Corruption: Upon receiving $(\textsc{Corrupt}, sid, J)$ from $\mathcal{S}$, return $(\textsc{Corrupted}, sid, history(J))$ to $\mathcal{S}$. Here J can be party U or party S. Note that in the dummy functionality, corruption is not necessary because $\mathcal{S}$ obtain all inputs and produce outputs.

Figure 24: Dummy blind signature functionality $\mathcal{F}_1$ in Lemma E.1.

Proof. The proof is straightforward. Consider $\exists \mathcal{S}_1, \forall \mathcal{Z}$, $\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}} = \text{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}$; For this $\mathcal{S}_1$, there exists $\mathcal{S}_2$, for all $\mathcal{Z}$, $\left|\text{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_2}_{\pi_d, \mathcal{S}_2, \mathcal{Z}}\right| \leq \epsilon_1$; So we can have $\exists \mathcal{S}_2$, $\forall \mathcal{Z}$, $\left|\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_2}_{\pi_d, \mathcal{S}_2, \mathcal{Z}}\right| \leq \epsilon_1$. Similarly, $\exists \mathcal{S}_5, \forall \mathcal{Z}$, $\left|\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_5}_{\pi_d, \mathcal{S}_5, \mathcal{Z}}\right| \leq \epsilon_1 + \cdots + \epsilon_4$. Note that $\mathcal{F}_5 = \mathcal{F}_{\text{BSIG}}$. We complete the proof. □

E.3.2 The Proof

Now we turn to the proof of Theorem 3.1.

Proof. In order to prove that $\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}} \approx \text{EXEC}^{\mathcal{F}_{\text{BSIG}}}_{\pi_d, \mathcal{S}, \mathcal{Z}}$, we use the proof strategy explored in Lemma E.1. We develop several bridge hybrid worlds between the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid world and the ideal world, and define the ensemble of random variables of $\mathcal{Z}$'s output of each bridge hybrid worlds as $\text{EXEC}^{\mathcal{F}_i}_{\pi_d, \mathcal{S}_i, \mathcal{Z}}$, $i = 1, 2, \ldots, 5$, where π_d is the dummy protocol same as that in the ideal world. Next we prove $\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}} \approx \text{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}} \approx \cdots \approx \text{EXEC}^{\mathcal{F}_5}_{\pi_d, \mathcal{S}_5, \mathcal{Z}} \approx \text{EXEC}^{\mathcal{F}_{\text{BSIG}}}_{\pi_d, \mathcal{S}, \mathcal{Z}}$. In our sequence of games we introduce a sequence of functionalities called "vault" which gradually becomes from dummy blind signature functionality $\mathcal{F}_1$ into the ideal functionality $\mathcal{F}_{\text{BSIG}} = \mathcal{F}_5$ across a sequence of five steps. At the same time the corresponding simulator becomes from $\mathcal{S}_1$ into ideal world simulator $\mathcal{S} = \mathcal{S}_5$. We also explicitly present the construction of $\mathcal{S}$ after the proof.

Note that we assume the underlying lite blind signature satisfies both lite-unforgeability and equivocality.

$\text{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}$. Here the vault $\mathcal{F}_1$, i.e. the dummy blind signature functionality, is between the dummy parties S, U, V and the simulator $\mathcal{S}_1$; the vault $\mathcal{F}_1$ just forwards the messages between the dummy parties and $\mathcal{S}_1$, and at same time does some "basic" recording. Please refer to Figure 24.

The simulator $\mathcal{S}_1$ simulates exactly the protocol $\pi_{\Sigma(\text{BSIG})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid model except that all inputs/outputs of the parties of protocol $\pi_{\Sigma(\text{BSIG})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid model are from/to the vault $\mathcal{F}_1$ instead of from/to $\mathcal{Z}$.

Analysis:

Note that $\mathcal{S}_1$ restates the whole execution in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid world. So we have $\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}} = \text{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}$.

$\text{EXEC}^{\mathcal{F}_2}_{\pi_d, \mathcal{S}_2, \mathcal{Z}}$. Here the vault $\mathcal{F}_2$, operating like the vault $\mathcal{F}_1$, forwards the messages between the dummy parties and the simulator $\mathcal{S}_2$, and records some basic information. Furthermore, $\mathcal{F}_2$ needs to deal with the patching for m as that in $\mathcal{F}_{\text{BSIG}}$: if there is no $\langle U, \mathsf{completed}\rangle$ in $history(S)$, the vault $\mathcal{F}_2$ will store the patched m into $history(U)$ (note that if there is old $\langle m, \mathsf{ver}'\rangle$ in $history(U)$, then replace the old m with this patched m); now if receives message $(\text{SIGNATURE}, sid, U, \mathsf{completed})$ from $\mathcal{S}_2$, then $\mathcal{F}_2$ stores a mark done into $history(U)$.

$\mathcal{S}_2$ is same as $\mathcal{S}_1$ to simulate the whole $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S})$-hybrid world except: in the case that the user is corrupted, $\mathcal{Z}$ sends $((crs, vk, \mathbf{u}), (m, \rho_1))$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$ on the behalf of the corrupted user U; if $((crs, vk, \mathbf{u}), (m, \rho_1)) \in R_U$, then $\mathcal{S}_2$ patches m into $\mathcal{F}_2$.

Analysis:

This is a preparation step for the next step and the modification has no effect on $\mathcal{Z}$'s output. So $\mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} = \mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}$.

$\mathbf{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Here, we modify the signature verification and we use the same one used in the ideal world. Now the vault $\mathcal{F}_3$ will be in charge of the verification: when receiving $(\textsc{Verify}, sid, m, \sigma, \mathsf{ver}')$ from the dummy verifier V, if $\mathsf{ver}' = \mathsf{ver}$, the signer S is not corrupted, $\mathsf{ver}(m,\sigma) = 1$, and m is not recorded with done in the table, then $\mathcal{F}_3$ halts, otherwise $(\textsc{Verified}, sid, \mathsf{ver}'(m,\sigma))$ will be returned to the dummy V. From now on the VERIFY and VERIFIED messages will not appear between the simulator $\mathcal{S}_3$ and the vault $\mathcal{F}_3$.

Analysis:

We define $\mathbf{E}$ as the event that in the $\mathcal{F}_2$-hybrid world, the signer has generated the verification algorithm ver, and some party V is activated with a verification request $(\textsc{Verify}, sid, m, \sigma, \mathsf{ver})$, where $\mathsf{ver}(m,\sigma) = 1$, and S is honest at this moment, and m has not been signed. Note that event $\mathbf{E}$ can also be defined in the $\mathcal{F}_3$-hybrid world. The only difference between the two worlds, i.e. the $\mathcal{F}_2$-hybrid world and the $\mathcal{F}_3$-hybrid world, is the verification stage. If event $\mathbf{E}$ does not occur, then $\mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} = \mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Based on the difference lemma (refer to [Sho04]), $|\mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}| \leq \Pr[\mathbf{E}]$. Now we still need to argue that $\Pr[\mathbf{E}]$ is negligible.

We now construct an algorithm $\mathcal{B}$ to output $(k+1)$ message-signature pairs after k queries to the signing oracle. Here the algorithm $\mathcal{B}$ is supplied with a verification key vk and is allowed to access to the signing oracle as defined in the lite-unforgeability model.

$\mathcal{B}$ runs a simulated $\mathcal{S}_2$ and a simulated $\mathcal{F}_2$. Note that now the simulator $\mathcal{S}_2$ has to simulate party S without knowing the signing key sk.

1. When $\mathcal{Z}$ activates some party S with input $(\textsc{KeyGen}, sid)$ with $sid = (S, sid')$ for some sid', $\mathcal{B}$ returns ver to $\mathcal{Z}$, where $\mathsf{ver} \stackrel{\mathsf{def}}{=} \mathtt{verify}(crs, vk, \cdot, \cdot)$. Note that vk is from the input of $\mathcal{B}$ as described before.
2. When the simulated S receives $(\textsc{VerifiedSVZK}, sid, U, (crs, vk', \mathbf{u}))$ from $\mathcal{F}^{R_U}_{\mathrm{SVZK}}$, in the case that $vk' = vk$, $\mathcal{S}_2$ needs to simulate $\mathcal{F}^{R_S}_{\mathrm{SPZK}}$ to send $(\textsc{ProveSPZK}, sid, U, (crs, vk, \mathbf{u}, \mathbf{s}))$ to $\mathcal{Z}$. However $\mathcal{S}_2$ cannot produce $\mathbf{s}$ by itself because now $\mathcal{S}_2$ does not have the signing key sk. Note that $\mathcal{S}_2$ simulates $\mathcal{F}^{R_U}_{\mathrm{SVZK}}$ and $\langle m, \rho_1\rangle$ can always be obtained; $\mathcal{S}_2$ queries the signing oracle with $\langle m, \rho_1\rangle$ and obtains $\mathbf{s}$.
3. After finishing the above step, the simulator $\mathcal{S}_2$ lets $\mathcal{F}^{R_S}_{\mathrm{SPZK}}$ send $(\textsc{ProveSPZK}, sid, U, (crs, vk, \mathbf{u}, \mathbf{s}))$ to $\mathcal{Z}$. In the case that the user is honest, the user will produce signature σ for m; $\mathcal{B}$ records such (m,σ) pairs. In the case that the user is corrupted, $\mathcal{B}$ computes $\sigma = \mathsf{lbs}_3(crs, vk, m, \rho_1, \mathbf{u}, \mathbf{s}; \rho_3)$ by randomly selecting ρ_3; $\mathcal{B}$ records (m,σ).
4. When $\mathcal{Z}$ activates some party V with input $(\textsc{Verify}, sid, m, \sigma, \mathsf{ver}')$, $\mathcal{B}$ checks whether (m,σ) is a forgery, i.e. if $\mathsf{ver}' = \mathsf{ver}$, $\mathsf{ver}'(m,\sigma) = 1$, and m has never been queried to the signing oracle. If (m,σ) is a forgery, $\mathcal{B}$ outputs the pair and all recorded pairs, say the number is k, and halts. If $\mathcal{S}_2$ is asked by $\mathcal{Z}$ to corrupt the signer then $\mathcal{B}$ halts.

Note that whenever the event $\mathbf{E}$ occurs, algorithm $\mathcal{B}$ can produce a successful one-more forgery. Therefore $\Pr[\mathbf{E}] = \mathsf{Adv}^{\mathrm{lbs}}_{\mathrm{luf}}$. So, we have $\left|\mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}\right| \leq \mathsf{Adv}^{\mathrm{lbs}}_{\mathrm{luf}}$.

$\mathbf{EXEC}^{\mathcal{F}_4}_{\pi_d,\mathcal{S}_4,\mathcal{Z}}$. Here we let the simulator $\mathcal{S}_4$ send the vault $\mathcal{F}_4$ the pair $\langle \mathsf{sig}, \mathsf{ver}\rangle$ in the key generation, i.e. $\mathcal{S}_4$ sends $(\text{ALGORITHMS}, sid, \mathsf{sig}, \mathsf{ver})$ to $\mathcal{F}_4$. Now the vault $\mathcal{F}_4$ records $\langle \mathsf{sig}, \mathsf{ver}\rangle$ into $history(S)$.

In the case that the signer is corrupted, inside $\mathcal{S}_4$, when $\mathcal{Z}$ sends $((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2))$ to $\mathcal{F}^{R_S}_{\text{SPZK}}$ on behalf of the corrupted signer S, $\mathcal{S}_4$ defines $\mathsf{sig} \stackrel{\mathsf{def}}{=} \mathtt{sign}(crs, vk, sk, \cdot, \cdot)$ and patches sig into $\mathcal{F}_4$. Now $\mathcal{F}_4$ deals with the patching for sig as in $\mathcal{F}_{\text{BSIG}}$: record the patched sig into $history(S)$; if there is old sig, then replace the old sig with this patched sig.

Analysis:

This is a preparation step for the next step and the modification has no effect on $\mathcal{Z}$'s output. So $\text{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}} = \text{EXEC}^{\mathcal{F}_4}_{\pi_d,\mathcal{S}_4,\mathcal{Z}}$.

$\mathbf{EXEC}^{\mathcal{F}_5}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}$. When the vault $\mathcal{F}_5$ receives $(\text{SIGN}, sid, m, \mathsf{ver}')$ from each dummy user U, $\mathcal{F}_5$ "blocks" m and sends $(\text{SIGN}, sid, U, \mathsf{ver}')$ to $\mathcal{S}_5$; now $\mathcal{S}_5$ will simulate the user U without the real m (as opposed to the real m used in $\mathcal{S}_4$). Note that the underlying lite blind signature is equivocal, so now $\mathcal{S}_5$ can obtain some "help" from the machine $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2)$ which is defined in Definition 2.3. Please refer to a full description of the simulator immediately after the proof. Here we only give the difference from the previous simulator $\mathcal{S}_4$:

Once receiving $(\text{SIGNATURE}, sid, U, \mathsf{ver}')$ from $\mathcal{F}_5$, $\mathcal{S}_5$ runs $\mathcal{I}_1$ with trapdoor τ and vk' and obtains $\mathbf{u}$ with some auxiliary information, i.e. $(\mathbf{u}, aux) \leftarrow \mathcal{I}_1(crs, \tau, vk')$, where vk' is obtained from ver'; $\mathcal{S}_5$ records $\langle U, vk', \mathbf{u}, aux\rangle$ into $temp$. Then $\mathcal{S}_5$ will simulate $\mathcal{F}^{R_U}_{\text{SVZK}}$ to send $(\text{PROVESVZK}, sid, U, (crs, vk', \mathbf{u}))$ to $\mathcal{Z}$.

If $\mathcal{Z}$ corrupts U after $\mathcal{Z}$ receives $(\text{PROVESVZK}, sid, U, (crs, vk', \mathbf{u}))$ from $\mathcal{F}^{R_U}_{\text{SVZK}}$, $\mathcal{S}_5$ corrupts the dummy U and obtains its input m from $\mathcal{F}_5$; then $\mathcal{S}_5$ records m into $temp$ together with $\langle U, vk', \mathbf{u}, aux\rangle$, and runs $\rho_1 \leftarrow \mathcal{I}_2(U, temp)$. $\mathcal{S}_5$ returns $\mathcal{Z}$ $\langle m, \rho_1\rangle$ as the internal state of U.

If $\mathcal{Z}$ corrupts U after $\mathcal{Z}$'s receiving $(\text{SIGNATURE}, sid, m, \sigma)$, $\mathcal{S}_5$ corrupts the dummy U and obtains $\langle m, \sigma, \gamma\rangle$ from $\mathcal{F}_5$; then $\mathcal{S}_5$ records $\langle \mathbf{s}, sk, \rho_2\rangle$ and $\langle m, \sigma, \gamma\rangle$ into $temp$ together with $\langle U, vk', \mathbf{u}, aux\rangle$; note that if S is not corrupted, $\langle \mathbf{s}, sk, \rho_2\rangle$ is produced by the simulated S, and if S is corrupted, $\langle \mathbf{s}, sk, \rho_2\rangle$ can be obtained from $\mathcal{Z}$. Now $\mathcal{S}_5$ runs $(\rho_1, \rho_3) \leftarrow \mathcal{I}_2(U, temp)$ and returns $\mathcal{Z}$ $\langle m, \rho_1, \rho_3\rangle$ as the internal state of U.

Analysis:

Assume $\mathcal{Z}$ can distinguish the two worlds, the $\mathcal{F}_4$-hybrid world and the $\mathcal{F}_5$-hybrid world, with non-negligible probability. We can construct an attacker $\mathcal{E}$ to break the equivocality property of the underlying lite blind signature with the same probability.

First, we define $\mathcal{E}^{\mathcal{O}}$ where $\mathcal{E}$ is obtained by modifying the $\mathcal{F}_4$-hybrid world with certain operations with querying oracle $\mathcal{O}$:

- When $\mathcal{Z}$ sends $(\text{SIGN}, sid, m, \mathsf{ver}')$ to a dummy user U, now $\mathcal{E}$ queries $\mathcal{O}$ with (U, m, vk) and receives $(U, \mathbf{u})$; $\mathcal{S}_4$ simulates $\mathcal{F}^{R_U}_{\text{SVZK}}$ to send $(\text{PROVESVZK}, sid, (crs, vk, \mathbf{u}))$ to $\mathcal{Z}$;
- When $\mathcal{Z}$ returns $\mathsf{VerifierComplete}$ to $\mathcal{F}^{R_U}_{\text{SVZK}}$ which is simulated inside $\mathcal{S}_4$, $\mathcal{S}_4$ simulates the honest signer S to compute $\mathbf{s} \leftarrow \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2)$ where ρ_2 is randomly chosen; then $\mathcal{S}_4$ simulates $\mathcal{F}^{R_S}_{\text{SPZK}}$ to send $(\text{PROVESPZK}, sid, (crs, vk, \mathbf{s}))$ to $\mathcal{Z}$; if $\mathcal{Z}$ returns $\mathsf{VerifierComplete}$ to $\mathcal{F}^{R_S}_{\text{SPZK}}$, then $\mathcal{E}$ queries $\mathcal{O}$ with $(U, \mathbf{s}, \rho_2, sk)$ and receives (U, σ); $\mathcal{E}$ returns $(\text{SIGNATURE}, sid, m, \sigma)$ to $\mathcal{Z}$ on behalf of the dummy user.

When the signer is corrupted, $\mathcal{Z}$ may send $(\text{PROVESPZK}, sid, (crs, vk, \mathbf{s}), (sk, \rho_2))$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$ which is simulated inside $\mathcal{S}_4$, if $((crs, vk, \mathbf{s}), (sk, \rho_2)) \in R_S$ then $\mathcal{S}_4$ patches sk to $\mathcal{F}_4$; now $\mathcal{E}$ queries $\mathcal{O}$ with $(U, \mathbf{s}, \rho_2, sk)$ and receives signature σ, and returns the signature to $\mathcal{Z}$ on behalf of the dummy user.

- Once receiving $(\text{CORRUPT}, sid, U)$ from $\mathcal{Z}$, $\mathcal{S}_4$ needs to return the internal state of U to $\mathcal{Z}$; now $\mathcal{E}$ queries $\mathcal{O}$ with (U, open) and obtains the internal state; $\mathcal{S}_4$ then returns the internal state to $\mathcal{Z}$.

Observe that $\mathcal{E}^{Users(crs,\cdot)}$ is exactly the $\mathcal{F}_4$-hybrid world, and $\mathcal{E}^{\mathcal{I}(crs,\tau,\cdot)}$ is exactly the $\mathcal{F}_5$-hybrid world. If $\mathcal{Z}$ can distinguish the two worlds, then $\mathcal{E}$ can break the equivocality of the underlying lite blind signature. So we obtain $\left|\text{EXEC}^{\mathcal{F}_4}_{\pi_d,\mathcal{S}_4,\mathcal{Z}} - \text{EXEC}^{\mathcal{F}_5}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}\right| \leq \mathsf{Adv}^{\text{lbs}}_{\text{eq}}$.

Note that the vault $\mathcal{F}_5$ is exactly the functionality $\mathcal{F}_{\text{BSIG}}$ and $\mathcal{S}_5$ is same as $\mathcal{S}$ in the ideal world. So $\text{EXEC}^{\mathcal{F}_{\text{BSIG}}}_{\pi_d,\mathcal{S},\mathcal{Z}} = \text{EXEC}^{\mathcal{F}_5}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}$.

Based on all discussions above, we obtain $\left|\text{EXEC}^{\mathcal{F}_{\text{BSIG}}}_{\pi_d,\mathcal{S},\mathcal{Z}} - \text{EXEC}^{\mathcal{F}_{\text{CRS}},\mathcal{F}^{R_U}_{\text{SVZK}},\mathcal{F}^{R_S}_{\text{SPZK}}}_{\pi_{\Sigma(\text{BSIG})},\mathcal{Z}}\right| \leq \mathsf{Adv}^{\text{lbs}}_{\text{luf}} + \mathsf{Adv}^{\text{lbs}}_{\text{eq}}$, where $\mathsf{Adv}^{\text{lbs}}_{\text{luf}}$ is the lite-unforgeability advantage, $\mathsf{Adv}^{\text{lbs}}_{\text{eq}}$ is the equivocality advantage. □

E.3.3 The Simulator $\mathcal{S}$

Based on the gradual modification of the simulator in the proof above, we finally obtain the ideal functionality $\mathcal{S}$. Here we explicitly give the description of the simulator $\mathcal{S}$. To make the description clearer, we also give description of the dummy parties and the functionality.

Setup:
The simulator $\mathcal{S}$ generates the CRS for each party internally simulated, and keeps the trapdoor for himself: $(crs, \tau) \leftarrow \mathtt{CRSgen}(1^\lambda)$. Define two relations $R_U = \{((crs, vk, \mathbf{u}), (m, \rho_1)) \mid \mathbf{u} = \mathsf{lbs}_1(crs, vk, m; \rho_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2)) \mid \mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \wedge (vk, sk) \in \mathsf{KeyPair}\}$.

Simulation of key generation:

(1) In the ideal world, when the dummy signer S receives an input (KEYGEN, sid) from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}_{\text{BSIG}}$. The functionality $\mathcal{F}_{\text{BSIG}}$ then forwards (KEYGEN, sid) to the simulator $\mathcal{S}$. Once receiving this message from $\mathcal{F}_{\text{BSIG}}$, $\mathcal{S}$ lets the simulated copy of the party S run $(vk, sk) \leftarrow \mathtt{gen}(crs)$, defines the signing algorithm $\mathsf{sig} := \mathtt{sign}(crs, vk, sk, \cdot, \cdot)$ and the verification algorithm $\mathsf{ver} := \mathtt{verify}(crs, vk, \cdot, \cdot)$; then $\mathcal{S}$ sends message $(\text{ALGORITHMS}, sid, \mathsf{sig}, \mathsf{ver})$ to $\mathcal{F}_{\text{BSIG}}$. Now the functionality $\mathcal{F}_{\text{BSIG}}$ will record $\langle \mathsf{sig}, \mathsf{ver} \rangle$ in $history(S)$, and send $(\text{VERIFICATIONALG}, sid, \mathsf{ver})$ to the dummy signer S, and then to the environment $\mathcal{Z}$.

(1^+) Now if $\mathcal{S}$ receives $(\text{CORRUPT}, sid, S)$ from $\mathcal{Z}$, returns $\langle \mathsf{sig}, \mathsf{ver} \rangle$ to $\mathcal{Z}$. Note that in this stage no signature has been generated, and no involved random coins will be sent to $\mathcal{Z}$ when S is corrupted.

Since this point, $\mathcal{Z}$ may play the corrupted S with different $\langle \overline{vk}, \overline{sk} \rangle$ and some random coins $\overline{\rho}_2$ to respond to each different user request message. In the future when the corrupted S (controlled by $\mathcal{Z}$) wants to respond to the user's request $(\text{VERIFIEDSVZK}, sid, U, (crs, vk, \mathbf{u}))$ from $\mathcal{F}^{R_U}_{\text{SVZK}}$ where $vk = \overline{vk}$, it needs to send $(\text{PROVESPZK}, sid, U, (crs, vk, \mathbf{u}, \overline{\mathbf{s}}), (\overline{sk}, \overline{\rho}_2))$ to $\mathcal{F}^{R_S}_{\text{SPZK}}$, where $\overline{\mathbf{s}} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, \overline{sk}; \overline{\rho}_2)$. When $\mathcal{F}^{R_S}_{\text{SPZK}}$ returns $(\text{VERIFIEDSPZK}, sid, U, (crs, vk, \mathbf{u}, \overline{\mathbf{s}}))$ to the user U, i.e. $((crs, vk, \mathbf{u}, \overline{\mathbf{s}}), (\overline{sk}, \overline{\rho}_2)) \in R_S$, the simulator $\mathcal{S}$ can obtain $(\overline{sk}, \overline{\rho}_2)$ and define $\overline{\mathsf{sig}} := \mathtt{sign}(crs, vk, \overline{sk}, \cdot, \cdot)$ and patch $\overline{\mathsf{sig}}$ into $\mathcal{F}_{\text{BSIG}}$. Note that if $vk \neq \overline{vk}$, then $\mathcal{F}^{R_S}_{\text{SPZK}}$ returns $(\text{VERIFIEDSPZK}, sid, U, \perp)$ to U, i.e. $((crs, vk, \mathbf{u}, \overline{\mathbf{s}}), (\overline{sk}, \overline{\rho}_2)) \notin$

R_S; the reason is that now $(vk, \overline{sk}) \notin \mathsf{KeyPair}$. Note that without this patching, $\mathcal{Z}$ may distinguish the two worlds based on the output of U.

Simulation of signature generation:

Here we need to simulate the user U and the signer S. The simulation of the party U is very complicated, while that of party S is simple. The main reason is that: the real m is withheld by $\mathcal{F}_{\text{BSIG}}$ and the simulator $\mathcal{S}$ has to simulate U without such m, and when party U is corrupted the simulator $\mathcal{S}$ has to equivocate the generated transcripts; while the signing algorithm sig has been known by the simulator, and the party S can be simulated honestly. Notice that though the real m is not given, $\mathcal{S}$ can access to machine $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2)$ defined in Definition 2.3 because the underlying lite blind signature is equivocal. We give details below.

(2) When the dummy user U receives an input $(\text{SIGN}, sid, m, \mathsf{ver}')$ from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}_{\text{BSIG}}$, $\mathcal{F}_{\text{BSIG}}$ records $\langle m, \mathsf{ver}' \rangle$ in $history(U)$ and forwards $(\text{SIGN}, sid, \mathsf{ver}', U)$ to the simulator $\mathcal{S}$. Once receiving this message from $\mathcal{F}_{\text{BSIG}}$, $\mathcal{S}$ obtains vk' from ver' and runs $(\mathbf{u}, aux) \leftarrow \mathcal{I}_1(crs, \tau, vk')$, record $\langle U, vk', \mathbf{u}, aux \rangle$ into $temp$. Then $\mathcal{S}$ simulates $\mathcal{F}_{\text{SVZK}}^{R_U}$ to send the message $(\text{PROVESVZK}, sid, U, (crs, vk', \mathbf{u}))$ to $\mathcal{Z}$.

(2^+) Now if the simulator $\mathcal{S}$ receives $(\text{CORRUPT}, sid, U)$ from $\mathcal{Z}$ (i.e. after $\mathcal{Z}$ received the PROVESVZK message). The simulator $\mathcal{S}$ reconstructs the simulated user U's internal state $\langle m, \rho_1 \rangle$ as follows: $\mathcal{S}$ sends the CORRUPT message to $\mathcal{F}_{\text{BSIG}}$ and obtains the input m of the dummy U, and adds m into $temp$, and then runs $\rho_1 \leftarrow \mathcal{I}_2(U, temp)$. $\mathcal{S}$ returns $\langle m, \rho_1 \rangle$ to $\mathcal{Z}$.

Since this point, $\mathcal{Z}$ may play the corrupted U with different $\overline{m}$ and some random coins $\overline{\rho}_1$, and the corrupted U sends $(\text{PROVESVZK}, sid, (crs, vk', \overline{\mathbf{u}}), (\overline{m}, \overline{\rho}_1))$ to the $\mathcal{F}_{\text{SVZK}}^{R_U}$ where $\overline{\mathbf{u}} = \mathsf{lbs}_1(crs, vk', \overline{m}; \overline{\rho}_1)$. When $\mathcal{F}_{\text{SVZK}}^{R_U}$ returns $(\text{VERIFIEDSVZK}, sid, U, (crs, vk', \overline{\mathbf{u}}))$ to the simulated signer S, i.e. $((crs, vk', \overline{\mathbf{u}}), (\overline{m}, \overline{\rho}_1)) \in R_U$, the simulator $\mathcal{S}$ will patch $\overline{m}$ into $\mathcal{F}_{\text{BSIG}}$. Note that without this patching, $\mathcal{Z}$ may distinguish the two worlds based on the signature verification: valid signature for $\overline{m}$ will be rejected in the ideal world.

(3) When $\mathcal{Z}$ returns $(\text{PROVESVZK}, sid, U, \mathsf{VerifierError})$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$, $\mathcal{F}_{\text{SVZK}}^{R_U}$ sends $(\text{PROVESVZK}, sid, U, \perp)$ to the simulated S, and $\mathcal{S}$ now sends $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerError})$ to $\mathcal{F}_{\text{BSIG}}$; when $\mathcal{F}_{\text{BSIG}}$ receives this message, it will output $(\text{SIGNSTATUS}, sid, U, \mathsf{incompleted})$ to the dummy party S, then to $\mathcal{Z}$. Else when $\mathcal{Z}$ returns $(\text{PROVESVZK}, sid, U, \mathsf{VerifierComplete})$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$, $\mathcal{F}_{\text{SVZK}}^{R_U}$ sends $(\text{PROVESVZK}, sid, U, (crs, vk', \mathbf{u}))$ to the simulated S, now S randomly selects ρ_2 and computes $\mathbf{s} \leftarrow \mathsf{lbs}_2(crs, vk', \mathbf{u}, sk; \rho_2)$, and $\mathcal{S}$ now sends $(\text{SIGNSTATUS}, sid, U, \mathsf{SignerComplete})$ to $\mathcal{F}_{\text{BSIG}}$; when $\mathcal{F}_{\text{BSIG}}$ receives this message, it records $\langle U, \mathsf{completed} \rangle$ in $history(S)$ and outputs $(\text{SIGNSTATUS}, sid, U, \mathsf{completed})$ to the dummy party S, then to $\mathcal{Z}$. At the same time S sends $(\text{PROVESPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}), (sk, \rho_2))$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$ and $\mathcal{F}_{\text{SPZK}}^{R_S}$ sends $(\text{PROVESPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}))$ to $\mathcal{Z}$.

(3^+) Now if the simulator $\mathcal{S}$ receives $(\text{CORRUPT}, sid, S)$ from $\mathcal{Z}$ (i.e. $\mathcal{Z}$ received the PROVESPZK message); $\mathcal{S}$ directly returns the (sk, ρ_2) as the internal state of the simulated signer S to $\mathcal{Z}$.

In the future, $\mathcal{Z}$ may supply the corrupted S with different key-pair $\langle \overline{vk}, \overline{sk} \rangle$, and different random coins $\overline{\rho}_2$ for different user request as discussed in **(1^+)**.

(4) When $\mathcal{Z}$ returns $(\text{PROVESPZK}, sid, U, \mathsf{VerifierError})$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, $\mathcal{F}_{\text{SPZK}}^{R_S}$ sends $(\text{PROVESPZK}, sid, U, \perp)$ to the simulated U, $\mathcal{S}$ will send $(\text{SIGNATURE}, sid, U, \mathsf{UserError})$ to $\mathcal{F}_{\text{BSIG}}$; when $\mathcal{F}_{\text{BSIG}}$ receives this message, it will output $(\text{SIGNATURE}, sid, \perp)$ to the dummy party U, then to $\mathcal{Z}$. Else when $\mathcal{Z}$ returns $(\text{PROVESPZK}, sid, U, \mathsf{VerifierComplete})$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, $\mathcal{F}_{\text{SPZK}}^{R_S}$ sends $(\text{PROVESPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}))$ to the simulated U, then U randomly selects $\widetilde{\rho}_3$ and computes $\widetilde{\sigma} \leftarrow \mathsf{lbs}_3(crs, vk', \mathbf{u}, \mathbf{s}, \widetilde{m}, \widetilde{\rho}_1; \widetilde{\rho}_3)$; if $\mathsf{ver}'(crs, \widetilde{m}, \widetilde{\sigma}) = 1$, then $\mathcal{S}$ now sends $(\text{SIGNATURE}, sid, U, \mathsf{UserComplete})$ to $\mathcal{F}_{\text{BSIG}}$; now $\mathcal{F}_{\text{BSIG}}$ will select γ and use the recorded sig to compute $\sigma \leftarrow \mathsf{sig}(m, \gamma)$, record $\langle \sigma, \gamma, \mathsf{done} \rangle$ next to $\langle m, \mathsf{ver}' \rangle$ inside $history(U)$, and output $(\text{SIGNATURE}, sid, \sigma)$ to the dummy user U, then to $\mathcal{Z}$. If $\mathsf{ver}'(crs, \widetilde{m}, \widetilde{\sigma}) \neq 1$, then

$\mathcal{S}$ now sends $(\text{SIGNATURE}, sid, U, \mathsf{UserError})$ to $\mathcal{F}_{\text{BSIG}}$, and now $\mathcal{F}_{\text{BSIG}}$ sends $(\text{SIGNATURE}, sid, \perp)$ to the dummy U, then to $\mathcal{Z}$.

(4$^+$) Now if the simulator $\mathcal{S}$ receives $(\text{CORRUPT}, sid, U)$ from $\mathcal{Z}$ (i.e. the dummy U has outputted a valid signature σ for m). $\mathcal{S}$ reconstructs the simulated user U's internal state $\langle m, \rho_1, \rho_3 \rangle$ as follows: $\mathcal{S}$ sends the CORRUPT message to $\mathcal{F}_{\text{BSIG}}$ and obtains m, σ, γ of the dummy U, and then records $\langle \mathbf{s}, sk, \rho_2 \rangle$ and $\langle m, \sigma, \gamma \rangle$ into $temp$, and runs $(\rho_1, \rho_3) \leftarrow \mathcal{I}_2(U, temp)$. $\mathcal{S}$ returns $\langle m, \rho_1, \rho_3 \rangle$ to $\mathcal{Z}$.

(5) If the signer is corrupted at the beginning, i.e. no (KEYGEN, sid) was sent out from $\mathcal{Z}$, in future $\mathcal{Z}$ may play the corrupted S with different $\langle \overline{vk}, \overline{sk} \rangle$ and some random coins $\overline{\rho}_2$ to respond to each different user request message as in **(1$^+$)**.

(6) If the user is corrupted at the beginning, i.e. no $(\text{SIGN}, sid, m, \mathsf{ver}')$ was sent out from $\mathcal{Z}$ to U, in future $\mathcal{Z}$ may play the corrupted U with different $\overline{m}$ and some random coins $\overline{\rho}_1$ as in **(2$^+$)**.

Simulation of signature verification:

(7) When the dummy verifier V receives an input $(\text{VERIFY}, sid, m, \sigma, \mathsf{ver}')$ from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}_{\text{BSIG}}$, and $\mathcal{F}_{\text{BSIG}}$ will check if the input consists of forgery. If $\mathsf{ver}' = \mathsf{ver}$ where ver is from $history(S)$, the signer S is not corrupted, $\mathsf{ver}'(m, \sigma) = 1$, and there is no U such that m is recorded with done in $history(U)$, then $\mathcal{F}_{\text{BSIG}}$ halts; Else, it outputs $(\text{VERIFIED}, sid, \mathsf{ver}'(m, \sigma))$ to party V.