Equivocal Blind Signatures and Adaptive UC-Security [*]

Aggelos Kiayias [†] Hong-Sheng Zhou[†]

April 12, 2007

Abstract

We study the design of practical blind signatures in the universal composability (UC) setting against adaptive adversaries. We introduce a new property for blind signature schemes that is fundamental for managing adaptive adversaries: an *equivocal blind signature* is a blind signature protocol where a simulator can construct the internal state of the client so that it matches a simulated transcript even after a signature was released. We present a general construction methodology for building practical 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. We then focus on the exact 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 realize SVZK in a very efficient manner using number theoretic mixed commitments while employing a constant size common reference string and without the need to satisfy non-malleability. 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 important observation simplifies blind signature design substantially as one can realize SPZK very efficiently in a commit-and-prove fashion using merely an extractable commitment.

We instantiate all the building blocks of our design methodology efficiently thus presenting the first practical UC blind signature that is secure against adaptive adversaries in the common reference string model. In particular, we present (1) a lite equivocal blind signature protocol that is based on elliptic curves and the 2SDH assumption of Okamoto, (2) efficient implementations of SPZK, SVZK for the required relations. Our construction also takes advantage of a round optimization method we discuss and it results in a protocol that has an overall communication overhead of as little as 3Kbytes, employing six communication moves and a constant length common reference string. We also present alternative implementations for our equivocal lite blind signature thus demonstrating the generality of our approach.

Finally we count the exact cost of realizing blind signatures with our protocol design by presenting the distance between the $\mathcal{F}_{\mathrm{BSIG}}$-hybrid world and the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world as a function of environment parameters. The distance calculation is facilitated by a basic lemma we prove about structuring UC proofs that may be of independent interest.

[*]Unpublished manuscript.

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

Contents

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 substituted by 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] 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 and public-key encryption) that has 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 modelling 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] 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 practical 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. Moreover, one can construct blind signatures in the CRS model secure against adaptive adversaries using the [CLOS02] secure two party computation compiler. None of these protocols are practical and it is currently unknown whether it is possible to build practical UC blind signature protocols using either methodology.

While generic feasibility results are important in understanding the requirements of a cryptographic primitive, it is equally important to study the exact requirements of building simulation-based provably secure blind signatures that are practical.

1.1 Our Results

In this work we study the design of practical blind signatures in the UC framework against adaptive adversaries. We emphasize that by "practical" we mean the following: (i) protocol design with 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 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) making exact measurements for communication, time complexity as well as tightness of security reductions, (v) 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.

We present the following results:

Equivocal blind signatures. We introduce a new property for blind signatures, called equivocality that is fundamental for proving security against adaptive adversaries. An equivocal blind signature allows a simulator to construct the internal state of a client including all random tapes so that a simulated communication transcript can open to any given message; this capability should hold true *even* after a signature corresponding to the simulated transcript has been released from the protocol. Equivocality strictly strengthens the notion of blindness as typically defined in game-based security formulations of blind signatures.

New methodology for building UC blind signatures. We present a general methodology for designing **practical** UC blind signatures. Our starting point is the new notion of a *lite blind signature*: this is a "light weight" blind signature protocol that we put forth that has a simple game-based security modelling and is intended to be relatively easy to instantiate. 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 present four instantiations of a lite blind signature, one that is based on generic cryptographic primitives and three that are number theoretic: (1) based on the LRSW assumption [LRSW99] and the DLDH assumption [BBS04], (2) the CDH assumption [DH76] and (3) the 2SDH assumption [Oka06]. We then extend lite blind signatures with the equivocality definition and we prove the conditions under which the lite blind signature protocols we presented above are equivocal (with the exception of the DLDH scheme).

Using an equivocal lite blind-signature as a fundamental building block, we illustrate a general protocol design strategy that allows one to produce a blind-signature secure against adaptive adversaries in the UC setting when coupled with an appropriate pair of ZK-functionalities. The formulation of the appropriate ZK-functionalities that are required for building blind signatures is an important part of our design approach. In particular these 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 we adopt the assumption that in each blind signature session there is a single signer and a multitude of users and verifiers (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).

Study of the exact ZK requirements for UC blind signatures. We study the exact requirements for adaptive security of UC blind signatures in terms of realizing the two necessary ZK functionalities. Our findings enable much more practical instantiations of these functionalities compared to realizing them "generically" based on existing UC-ZK formulations. First, we show that $\mathcal{F}_{\mathrm{SVZK}}$ can be realized in a commit-and-prove fashion using a commitment scheme that does not require non-malleability (while such property is essential for general UC commitments). We proceed to realize $\mathcal{F}_{\mathrm{SVZK}}$ using mixed commitments [DN02, Nie03] with only a constant length common reference string (as opposed to linear in the number of parties). Second, rather surprisingly, we find that the functionality $\mathcal{F}_{\mathrm{SPZK}}$ that will be employed by the signer need only be realized against static adversaries for our blind signature scheme to satisfy adaptive security! This enables a much more efficient realization design for $\mathcal{F}_{\mathrm{SPZK}}$ as we can implement it using merely an extractable commitment and a Sigma protocol. 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 [KZ07].

Practical implementation and calculating the cost of UC security. Based on our methodology and the ZK investigation above we present the first *practical* blind signature protocol that is universally composable and secure against adaptive adversaries. In particular, our construction is based on the 2SDH assumption and builds upon the lite blind signature derived from [Oka06]. We first prove that this protocol is equivocal unconditionally. Then, using our general methodology for building adaptively secure UC blind signatures we pair this lite blind signature with two efficient instantiations for $\mathcal{F}_{\mathrm{SVZK}}$ and $\mathcal{F}_{\mathrm{SPZK}}$ realized as described above. Our final protocol employs a CRS that is of constant length, it needs 6 moves of interaction and has a total communication overhead of only 3 KBytes. Our protocol also utilizes a round reduction technique that we introduce that enables us to save two communication moves. We then investigate the concrete security of our blind signature protocol and we show that if one substitutes an instance of an ideal $\mathcal{F}_{\mathrm{BSIG}}$ box with our protocol instantiation, this will incur at most $\mathsf{Adv}_{\mathrm{2sdh}} + (q_1 + 3q_3) \cdot \mathsf{Adv}_{\mathrm{dcr}} + (q_1 + q_2) \cdot (\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}}) + q_1 \cdot 2^{-\ell_{\mathsf{n}}} + q_1 \cdot 2^{-\lambda_U} + q_2 \cdot 2^{-\lambda_S} + (q_1 + 4q_3) \cdot 2^{-\lambda_0}$ computational advantage, where q_1, q_2, q_3 are parameters specified in the adversarial environment such that: q_1 is the number of the users which are corrupted initially, q_2 is the number of valid message-signature pairs obtained from a corrupted signer, and q_3 is the number of the users not corrupted initially. Note that the Adv functions correspond to the computational advantage that the adversary has in breaking the subcomponents of our construction and are defined in Section 2.

Proof strategy. This distance calculation between the $\mathcal{F}_{\mathrm{BSIG}}$-hybrid world and the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world is based on a specific structuring of the UC security proof that relies on a stepwise refinement of an initial dummy functionality that is gradually made to resemble $\mathcal{F}_{\mathrm{BSIG}}$. We present this as a general proof strategy for UC security proofs and we show a general lemma that helps structure such proofs and count their distance in a systematic way (refer to Lemma 5.1); we note that while this has no bearing on the security of the construction, we feel it makes a UC proof easier to understand as the final simulator would be comprised of different "simulator layers" that correspond to the stepwise refinements of the initial dummy functionality to $\mathcal{F}_{\mathrm{BSIG}}$.

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

2 Preliminaries

2.1 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 $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ 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 $\mathrm{EXEC}^{\mathcal{F}}_{\pi_d,\mathcal{S},\mathcal{Z}}$ 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. $\mathrm{EXEC}^{\mathcal{F}}_{\pi_d,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

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 $\mathrm{EXEC}_{\pi,\mathcal{Z}}$ 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 $\mathrm{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ 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. $\mathrm{EXEC}^{\mathcal{G}}_{\pi^{\varrho},\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}$.

2.2 Signatures

2.2.1 Signature Scheme $\Sigma(\mathrm{SIG})$ and Signature Functionality $\mathcal{F}_{\mathrm{SIG}}$

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

Definition 2.1 (EU-CMA Signature Schemes). A signature scheme $\Sigma(\mathrm{SIG}) = \langle \mathtt{gen}, \mathtt{sign}, \mathtt{verify} \rangle$ is called EU-CMA if the following properties hold for any negligible function $\mathsf{negl}(\cdot)$, and all large enough values of the security parameter λ,

Completeness: For any message $m \in \mathcal{M}$,

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

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

Unforgeability: For any PPT forger F,

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

Canetti[Can05] defines the signature functionality $\mathcal{F}_{\mathrm{SIG}}$ in Figure 1 and proves the theorem below, where signature protocol $\pi_{\Sigma(\mathrm{SIG})}$ presented in Figure 2 is transformed from $\Sigma(\mathrm{SIG})$.

Theorem 2.2. $\Sigma(\mathrm{SIG})$ *is EU-CMA* $\Leftrightarrow$ $\pi_{\Sigma(\mathrm{SIG})}$ *securely realizes* $\mathcal{F}_{\mathrm{SIG}}$.

2.2.2 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) $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ is an isomorphism with $\psi(g_2) = g_1$ and (iii) $\hat{\mathsf{e}} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ 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 $\hat{\mathsf{e}}(g_1, g_2) \neq 1$.

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

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

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

Signature generation: Upon receiving $(\texttt{Sign}, sid, m)$ 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 $(\texttt{Signature}, sid, m, \sigma)$ to party S, and record $\langle m, \sigma, rnd \rangle$ into $history(S)$. Else, output $(\texttt{Signature}, sid, \mathsf{error})$ to party S and halt.

Signature verification: Upon receiving $(\texttt{Verify}, sid, m, \sigma, \mathsf{ver}')$ 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 output $(\texttt{Verified}, sid, \mathsf{error})$ to party V and halt. Else, output $(\texttt{Verified}, sid, m, \mathsf{ver}'(m, \sigma))$ to party V.

Corruption: Upon receiving $(\texttt{Corrupt}, sid, J)$ from the adversary $\mathcal{S}$, return $(\texttt{Corrupted}, sid, history(J))$ to $\mathcal{S}$.

Figure 1: Signature functionality $\mathcal{F}_{\mathrm{SIG}}$.

2.2.3 Camenisch-Lysyanskaya Signature

Camenisch and Lysyanskaya [CL04] proposed a digital signature scheme which is EU-CMA secure under the LRSW assumption. The LRSW was first introduced by Lysyanskaya et al. [LRSW99].

Definition 2.3 (LRSW Assumption). Given the bilinear group parameters $(p, g, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}})$. Let $X, Y \in \mathbb{G}_1, X \leftarrow g^x, Y \leftarrow g^y$ and define $\mathcal{O}_{X,Y}()$ to be an oracle that, on input a value $m \in \mathbb{Z}_p$, it outputs a triple $\langle a, b, c \rangle$ such that $b = a^y$, and $c = a^{x+mxy}$ where $a \xleftarrow{r} \mathbb{G}_1$. Then, for all PPT adversaries $\mathcal{A}$,

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{lrsw}} \overset{\mathsf{def}}{=} \Pr\left[\begin{array}{c} x, y \in \mathbb{Z}_p; X = g^x; Y = g^y; (m, a, b, c) \leftarrow \mathcal{A}^{\mathcal{O}_{X,Y}()}(X, Y) : \\ m \notin \mathsf{Q} \wedge m \in \mathbb{Z}_p \wedge m \neq 0 \wedge a \in \mathbb{G}_1 \wedge b = a^y \wedge c = a^{x+mxy} \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{X,Y}()$.

Next we introduce Camenisch-Lysyanskaya signature.

Key generation: generate the bilinear group parameter $(p, \mathbb{G}_1, \mathbb{G}_T, g, \hat{\mathsf{e}})$; then choose $x, y \xleftarrow{r} \mathbb{Z}_p$, and compute $X = g^x$ and $Y = g^y$; set signing key as $sk = \langle x, y \rangle$ and verification key as $vk = \langle p, \mathbb{G}_1, \mathbb{G}_T, g, \hat{\mathsf{e}}; X, Y \rangle$.

Signature generation: on input message m, signing key $sk = \langle x, y \rangle$, and verification key $vk = \langle p, \mathbb{G}, \mathbb{G}_T, g, \hat{\mathsf{e}}; X, Y \rangle$, choose a random $a \in \mathbb{G}_1$, and output the signature $\sigma = \langle a, a^y, a^{x+mxy} \rangle$.

Signature verification: on input verification key $vk = \langle p, \mathbb{G}_1, \mathbb{G}_T, g, \hat{\mathsf{e}}; X, Y \rangle$, message m, and signature $\sigma = \langle a, b, c \rangle$, check whether the verification equations $\hat{\mathsf{e}}(a, Y) = \hat{\mathsf{e}}(g, b)$ and $\hat{\mathsf{e}}(X, a) \cdot \hat{\mathsf{e}}(X, b)^m = \hat{\mathsf{e}}(g, c)$ hold.

2.2.4 Waters Signature

Waters proposed a signature scheme [Wat05] which is EU-CMA secure under the CDH assumption over group $\mathbb{G}_1$.

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

Key generation: When party S is invoked with $(\mathtt{KeyGen}, sid)$ 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 $(\mathtt{VerificationAlg}, sid, \mathsf{ver})$ to $\mathcal{Z}$.

Signature generation: When party S is invoked with $(\mathtt{Sign}, sid, m)$ 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 $(\mathtt{Signature}, sid, m, \sigma)$ to $\mathcal{Z}$.

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

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

Figure 2: Signature protocol $\pi_{\Sigma(\mathrm{SIG})}$.

Definition 2.4 (Computational Diffie-Hellman (CDH) Assumption). Let $\mathbb{G}_1$ be a cyclic group of prime order p. The CDH problem defined as follows: given $g, g^a, g^b \in \mathbb{G}_1$, output $g^{ab} \in \mathbb{G}_1$. The CDH assumption suggests that any PPT algorithm $\mathcal{A}$ solving the CDH problem has negligible probability, i.e.

$$\mathsf{Adv}^{\mathcal{A}}_{\mathrm{cdh}} = \Pr\left[g \in \mathbb{G}_1; a, b \in \mathbb{Z}_p : \mathcal{A}(g, g^a, g^b) = g^{ab}\right] \leq \mathsf{negl}(\lambda).$$

Next, we introduce Waters signature.

Key generation: Randomly select $\alpha \xleftarrow{r} \mathbb{Z}_p$, and select generators $g, g_2, v, u_1, \ldots, u_n \in \mathbb{G}_1$ and set $g_1 \leftarrow g^\alpha$. Set verification key $vk = \langle g, g_1, g_2, v, u_1, \ldots, u_n \rangle$ and secret key $sk = \langle g_2^\alpha \rangle$.

Signature generation: Let m be the n-bit message to be signed, m_j is the j^{th} bit of m. Signer S randomly selects $r \xleftarrow{r} \mathbb{Z}_p$ and compute $\sigma_1 \leftarrow g_2^\alpha (v \prod_{j=1}^n u_j^{m_j})^r$ and $\sigma_2 \leftarrow g^r$. The signature for m is $\sigma \leftarrow \langle \sigma_1, \sigma_2 \rangle$.

Signature verification: Given public key $vk = \langle g, g_1, g_2, v, u_1, \ldots, u_n \rangle$, message m, and signature $\sigma = \langle \sigma_1, \sigma_2 \rangle$, check that $\hat{\mathsf{e}}(\sigma_1, g) / \hat{\mathsf{e}}(\sigma_2, v \prod_{j=1}^n u_j^{m_j}) = \hat{\mathsf{e}}(g_1, g_2)$. If they hold, then the verification is valid; otherwise invalid.

2.2.5 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 2.5 (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.

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

Next we briefly introduce Okamoto signature.

Key generation: Randomly select $g_2, u_2, v_2 \xleftarrow{\text{r}} \mathbb{G}_2$ and set $g_1 \leftarrow \psi(g_2)$, $u_1 \leftarrow \psi(u_2)$ and $v_1 \leftarrow \psi(v_2)$. Randomly select $x \xleftarrow{\text{r}} \mathbb{Z}_p$ 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 $f, r, s, h \xleftarrow{\text{r}} \mathbb{Z}_p$. 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.

2.2.6 Blind Signature Schemes

Definition 2.6 (Blind Signature Scheme). A blind digital signature scheme $\Sigma(\text{BSIG})$ is a five-tuple, consisting of two interactive Turing machines S, U, where S denotes the signer, and U the user, and three algorithms `CRSgen`, `gen`, `verify` as follows:

- `CRSgen`(1^λ) 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.

- `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 $\mathsf{completed}$ or $\perp$ and U outputs either (m, σ) or $\perp$.

- `verify`(crs, vk, m, σ) is a deterministic polynomial time algorithm, which outputs 1 or 0. For any message m, and for all $(crs, \tau) \leftarrow$ `CRSgen`(1^λ) and $(vk, sk) \leftarrow$ `gen`(crs), if both S and U follow the protocol then if S outputs $\mathsf{completed}$ and the output of the user is (m, σ) then `verify`$(crs, vk, m, \sigma) = 1$.

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

When party S and party U are same, i.e. $S = U$, a blind signature scheme $\Sigma(\text{BSIG})$ will collapse to a plain signature scheme $\Sigma(\text{SIG}) = \langle$`gen`, `sign`, `verify`$\rangle$. Here `gen` and `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 $rnd \overset{r}{\leftarrow} \texttt{RND}$ (such algorithm is immediately from the collapse of S, U into a single unit).

A secure blind signature scheme has the following two properties:

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, $\mathsf{Adv}^{\mathcal{A},L}_{\text{unforge}}(\lambda) \leq \mathsf{negl}(\lambda)$.

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 if both user instantiations terminate successfully and output two valid signatures $\langle \sigma_0, \sigma_1\rangle$, then $\mathcal{A}$ is supplied with $\langle \sigma_0, \sigma_1\rangle$, else $\mathcal{A}$ 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 2.7. We will revisit the security model later on in this paper. We only mention the above notions to illustrate the current understanding of the properties of blind signature in a game-based modeling. The definition here is following the definitions in [Oka06] and [HKKL07]. In [JLO97], the unforegeability definition is stronger than the definition above: the adversary is required to supply different message-signature pairs, i.e. $(m_1, \sigma_1), ..., (m_\ell, \sigma_\ell)$ where $\underline{(m_i, \sigma_i) \neq (m_j, \sigma_j)}$, $1 \leq i \neq j \leq \ell$; and the blindness definition is weaker than the definition here: the adversarial signer is not allowed to generate the verification key vk, and the key-pair (vk, sk) is generated honestly, i.e. $(vk, sk) \leftarrow \texttt{gen}(crs)$. In [Fis06], the blindness definition is same as here, and the unforgeability definition is the stronger version as in [JLO97]. In [KZ06], the unforgeability is same as here, and the blindness definition is the weaker version as in [JLO97].

2.3 Encryptions

2.3.1 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 $\mathsf{n} = \mathsf{pq}$, $\mathsf{d} = \text{lcm}(\mathsf{p}-1, \mathsf{q}-1)$, $\mathsf{K} = \mathsf{d}^{-1} \bmod \mathsf{n}$, 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 2.8 (Decisional Composite Residuosity Assumption). There is no PPT distinguisher $\mathcal{A}$ for n-th residues modulo n^2, i.e.

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

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

2.3.2 Linear Encryption

Boneh et al. [BBS04] proposed a variant of ElGamal encryption, called, Linear Encryption that is suitable for groups over which the DDH assumption fails.

Key generation: the public key pk is a triple of generators $t, v, w \in \mathbb{G}_1$ and the secret key sk is the exponents $x, y \in \mathbb{Z}_p$ such that $t^x = v^y = w$.

Encryption: to encrypt a message $m \in \mathbb{G}_1$, choose random values $a, b \in \mathbb{Z}_p$, and output the triple $(t^a, v^b, m \cdot w^{a+b})$.

Decryption: given an encryption (T, V, W), we recover the plaintext m as follows $m = \frac{W}{T^x \cdot V^y}$.

The Linear encryption is based on the Decision Linear Diffie-Hellman assumption, which was first introduced by Boneh et al. [BBS04]. With $g \in \mathbb{G}_1$ as above, along with arbitrary generators t,v, and w of $\mathbb{G}_1$, consider the following problem:

Definition 2.9 (Decision Linear Diffie-Hellman Assumption). Given $t, v, w, t^\alpha, v^\beta,\ w^\gamma \in \mathbb{G}_1$ as input, there is no PPT adversary $\mathcal{A}$ can distinguish with non-negligible probability that $\alpha + \beta = \gamma$, i.e.

$$\mathsf{Adv}^{\mathcal{A}}_{\text{dldh}} \overset{\mathsf{def}}{=} \left| \begin{array}{r} \Pr[t, v, w \in \mathbb{G}_1, \alpha, \beta \in \mathbb{Z}_p : \mathcal{A}(t, v, w, t^\alpha, v^\beta, w^{\alpha+\beta}) = 1] \\ - \Pr[t, v, w, \chi, \in \mathbb{G}_1, \alpha, \beta \in \mathbb{Z}_p : \mathcal{A}(t, v, w, t^\alpha, v^\beta, \chi) = 1] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

2.4 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 2.10 (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 $\zeta \overset{\mathtt{r}}{\leftarrow} \mathtt{RND}$, $\mathtt{ver}(pk, \mathtt{com}(pk, m; \zeta), m, \zeta) = 1$.

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

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

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

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

Next we define equivocal commitment schemes and extractable commitment schemes.

Definition 2.11 (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

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

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 $\zeta \overset{\mathtt{r}}{\leftarrow} \mathtt{RND}$.

Definition 2.12 (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 $\zeta \overset{\mathtt{r}}{\leftarrow} \mathtt{RND}$, there exists an extraction algorithm $\mathtt{extract}$ such that $\mathtt{extract}(pk, xk, \mathtt{com}(pk, m; \zeta)) = m$.

2.5 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. $\mathcal{L}_R \overset{\mathsf{def}}{=} \{x | \exists w \text{ s.t. } (x, w) \in R\}$. 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 2.13 (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 $(x, w) \in R$,

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

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 $\zeta \overset{\mathtt{r}}{\leftarrow} \mathtt{RND}$.

Next we define non-erasure NIZK schemes.

Definition 2.14 (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 $\zeta \xleftarrow{r} \texttt{RND}$.

2.6 Sigma Protocols

In this subsection we introduce Sigma protocols, [CDS94]. Informally, a Sigma-protocol is a three move public randomness protocol (cf. Figure 3) with a 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).

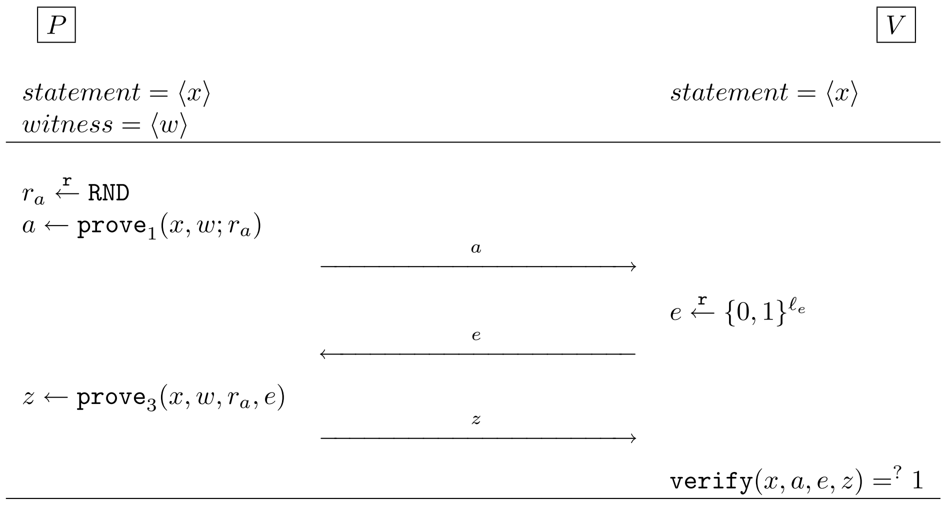

$\boxed{P}$
$\boxed{V}$
$statement = \langle x \rangle$
$witness = \langle w \rangle$
$statement = \langle x \rangle$
$r_a \xleftarrow{r} \texttt{RND}$
$a \leftarrow \texttt{prove}_1(x, w; r_a)$
$\xrightarrow{a}$
$e \xleftarrow{r} \{0,1\}^{\ell_e}$
$\xleftarrow{e}$
$z \leftarrow \texttt{prove}_3(x, w, r_a, e)$
$\xrightarrow{z}$
$\texttt{verify}(x, a, e, z) \stackrel{?}{=} 1$

Figure 3: Three move public randomness protocol for relation R. Here ℓ_e is the length of the challenge e.

Definition 2.15 (Sigma Protocol). A Sigma protocol for the relation R is a tuple $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate}\rangle$, where $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}\rangle$ is a three-move public randomness protocol, $\texttt{simulate}$ is a PPT algorithm, and where the following properties hold:

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

$$\Pr[r_a \xleftarrow{r} \texttt{RND}; a \leftarrow \texttt{prove}_1(x, w; r_a); z \leftarrow \texttt{prove}_3(x, w, r_a, e) : \texttt{verify}(x, a, e, z) = 0] \leq \mathsf{negl}(\lambda).$$

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

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

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

$$\mathsf{Adv}_{\mathrm{zk}}^{\mathcal{A}} \overset{\mathsf{def}}{=} \left| \begin{array}{l} \Pr\left[\begin{array}{l} r_a \overset{\mathrm{r}}{\leftarrow} \mathtt{RND}; a \leftarrow \mathtt{prove}_1(x, w; r_a); e \overset{\mathrm{r}}{\leftarrow} \{0,1\}^{\ell_e}; \\ z \leftarrow \mathtt{prove}_3(x, w, r_a, e) : \mathcal{A}(x, w, a, e, z) = 1 \end{array} \right] \\ \quad - \Pr\left[\begin{array}{l} e \overset{\mathrm{r}}{\leftarrow} \{0,1\}^{\ell_e}; (a, z, aux) \leftarrow \mathtt{simulate}(x, e) : \\ \mathcal{A}(x, w, a, e, z) = 1 \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

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

Definition 2.16 (Non-erasure sigma protocol). A non-erasure Sigma protocol for relation R is a tuple $\langle \mathtt{prove}_1, \mathtt{prove}_3, \mathtt{verify}, \mathtt{simulate}, \mathtt{reconstruct} \rangle$, where $\langle \mathtt{prove}_1, \mathtt{prove}_3, \mathtt{verify}, \mathtt{simulate} \rangle$ 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

$$\mathsf{Adv}_{\mathrm{nezk}}^{\mathcal{A}} \overset{\mathsf{def}}{=} \left| \begin{array}{l} \Pr\left[\begin{array}{l} r_a \overset{\mathrm{r}}{\leftarrow} \mathtt{RND}; a \leftarrow \mathtt{prove}_1(x, w; r_a); e \overset{\mathrm{r}}{\leftarrow} \{0,1\}^{\ell_e}; \\ z \leftarrow \mathtt{prove}_3(x, w, r_a, e) : \mathcal{A}(x, w, a, e, z, r_a) = 1 \end{array} \right] \\ \quad - \Pr\left[\begin{array}{l} e \overset{\mathrm{r}}{\leftarrow} \{0,1\}^{\ell_e}; (a, z, aux) \leftarrow \mathtt{simulate}(x, e); \\ r_a \leftarrow \mathtt{reconstruct}(x, a, z, w, e, aux) : \mathcal{A}(x, w, a, e, z, r_a) = 1 \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

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 a 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 protocol.

2.7 Other Primitives

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

Definition 2.17 (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 $\mathcal{A}$,

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

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

Definition 2.18 (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.

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

3 Equivocal Lite Blind Signatures

3.1 Our Basic Building Block: Lite Blind Signatures

A signature generation protocol is a tuple $\langle \texttt{CRSgen}, \texttt{gen}, \mathsf{lbs}_1, \mathsf{lbs}_2, \mathsf{lbs}_3, \texttt{verify} \rangle$ where CRSgen is a common reference string generation algorithm, 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 4 and verify is a signature verification algorithm. A lite blind signature is a signature generation protocol that satisfies correctness as well as two security properties, *lite-unforgeability* and *lite-blindness*, defined below. The motivation for lite blind signature is to simplify the security properties by requiring the adversary to play the security games with "open tapes".

$$\begin{array}{lcl}
\boxed{U} & & \boxed{S} \\
\text{CRS} = \langle crs \rangle & & \text{CRS} = \langle crs \rangle \\
\text{VerificationKey} = \langle vk \rangle & & \text{VerificationKey} = \langle vk \rangle \\
\text{Plaintext} = \langle m \rangle & & \text{SigningKey} = \langle sk \rangle \\
\hline
\rho_1 \overset{r}{\leftarrow} \texttt{RND} & & \\
\mathbf{u} \leftarrow \mathsf{lbs}_1(crs, vk, m; \rho_1) & \xrightarrow{\quad\quad \mathbf{u} \quad\quad} & \rho_2 \overset{r}{\leftarrow} \texttt{RND} \\
 & & \mathbf{s} \leftarrow \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \\
\rho_3 \overset{r}{\leftarrow} \texttt{RND} & \xleftarrow{\quad\quad \mathbf{s} \quad\quad} & \\
\sigma \leftarrow \mathsf{lbs}_3(crs, vk, m, \rho_1, \mathbf{u}, \mathbf{s}; \rho_3) & & \\
\hline
\end{array}$$

Figure 4: Outline of a two-move signature generation protocol.

We note that CRSgen is not employed in any of the security properties of a lite blind signature and thus it may be just a length parameter in a certain instantiation of the primitive. Nevertheless, we include it, since a lite blind signature is a basic building block in our design methodology and a common reference string will be used by subsequent extensions of the lightweight primitive (in particular in the definition of equivocality that will be given in Section 3.3).

Definition 3.1 (Completeness). A signature generation protocol as in Figure 4 is correct if for all $(crs, \tau) \leftarrow \texttt{CRSgen}(1^\lambda)$, for all $(vk, sk) \leftarrow \texttt{gen}(crs)$, for all $\rho_1, \rho_2, \rho_3 \overset{r}{\leftarrow} \texttt{RND}$, compute $\mathbf{u} \leftarrow \mathsf{lbs}_1(crs, vk, m; \rho_1)$, $\mathbf{s} \leftarrow \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2)$, and $\sigma \leftarrow \mathsf{lbs}_3(crs, vk, m, \rho_1, \mathbf{u}, \mathbf{s}; \rho_3)$, then $\texttt{verify}(crs, vk, m, \sigma) = 1$.

Note that the above completeness is very easy to be generalize 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 in the sense of [GMR88]. Recall that lite-unforgeability is much weaker compared to regular unforgeability of blind signatures (as defined e.g., in [JLO97, Fis06]) since it requires from the adversary to provide the input to an "honest" simulation of the user (as opposed to letting the adversary impersonate user instantiations himself).

Definition 3.2 (Lite-unforgeability). A signature generation protocol as in Figure 4 is lite-unforgeable if for all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and for any $L = \mathsf{poly}(\lambda)$, we have $\mathsf{Adv}_{\text{luf}}^{\mathcal{A},L}(\lambda) \leq \mathsf{negl}(\lambda)$, where $\mathsf{Adv}_{\text{luf}}^{\mathcal{A},L}(\lambda) \overset{\text{def}}{=} \Pr[\mathbf{Exp}_{\mathcal{A},L}^{\text{LUF}}(\lambda) = 1]$ and the experiment $\mathbf{Exp}_{\mathcal{A},L}^{\text{LUF}}(\lambda)$ 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 $k < L$
        $(m_k, \rho_{1,k}, state) \leftarrow \mathcal{A}_1(state, crs, vk)$;
        $\mathrm{s}_k \leftarrow \mathsf{lbs}_2(crs, vk, \mathsf{lbs}_1(crs, vk, m_k; \rho_{1,k}), sk; \rho_{2,k})$; $\rho_{2,k} \xleftarrow{r} \mathtt{RND}$;
        $state \leftarrow state||\mathrm{s}_k$; $k \leftarrow k+1$;
    $(m_1, \sigma_1, \ldots, m_\ell, \sigma_\ell) \leftarrow \mathcal{A}_2(state)$;
    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.

We proceed to the second property of a lite blind signature: lite-blindness; this property is designed similar to the strong blindness property of the primitive (as e.g., in [Oka06, ANN06], adversarial key generations are allowed); but even with the adversary selecting the key-pair, the lite-blindness property is still much weaker than strong blindness since it requires from the adversary to provide the signing key at some point to an "honest" simulation of the signer; also this property is restricted in the two-move setting. Informally in a lite-blindness attack, the adversary selects two messages and verification key; the two messages may be swapped according to a challenge bit b and the adversary receives the communication from two honest users employing the two messages (swapped according to b). The adversary then, may provide some feedback and his private tape used for this feedback including the signing secret and the random coins; in case that the provided signing secret is consistent to the verification key and the private tape is consistent to the feedback, and the adversary's communication to the two honest users results in two valid signatures, the adversary is also provided access to the two signatures (in a predetermined order). The adversary wins the game if he manages to guess the challenge bit.

Definition 3.3 (Lite-blindness). A signature generation protocol as in Figure 4 satisfies lite-blindness if for all PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, we have $\mathsf{Adv}^{\mathcal{A}}_{\mathrm{lb}}(\lambda) \leq \mathsf{negl}(\lambda)$, where $\mathsf{Adv}^{\mathcal{A}}_{\mathrm{lb}}(\lambda) \overset{\mathsf{def}}{=} \left|\Pr[\mathbf{Exp}^{\mathrm{LB}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right|$ and the experiment $\mathbf{Exp}^{\mathrm{LB}}_{\mathcal{A}}(\lambda)$ is defined below:

```
Experiment $\mathbf{Exp}^{\mathrm{LB}}_{\mathcal{A}}(\lambda)$
    $(crs,\tau) \leftarrow \mathtt{CRSgen}(1^\lambda)$;
    $(m_0, m_1, vk, state) \leftarrow \mathcal{A}_1(crs)$;
    $b \xleftarrow{r} \{0,1\}$;
    $\mathbf{u}_0 \leftarrow \mathsf{lbs}_1(crs, vk, m_b; \rho_{1,0})$; $\rho_{1,0} \xleftarrow{r} \mathtt{RND}$;
    $\mathbf{u}_1 \leftarrow \mathsf{lbs}_1(crs, vk, m_{1-b}; \rho_{1,1})$; $\rho_{1,1} \xleftarrow{r} \mathtt{RND}$;
    $(\mathrm{s}_0, \mathrm{s}_1, \rho_{2,0}, \rho_{2,1}, sk, state) \leftarrow \mathcal{A}_2(state, \mathbf{u}_0, \mathbf{u}_1)$;
    if $(vk, sk) \notin$ KEYPAIR or $\mathrm{s}_0 \neq \mathsf{lbs}_2(crs, vk, \mathbf{u}_0, sk; \rho_{2,0})$ or $\mathrm{s}_1 \neq \mathsf{lbs}_2(crs, vk, \mathbf{u}_1, sk; \rho_{2,1})$,
      set $(\sigma_0, \sigma_1) \leftarrow (\perp, \perp)$;
    $\sigma_b \leftarrow \mathsf{lbs}_3(crs, vk, m_b, \rho_{1,0}, \mathbf{u}_0, \mathrm{s}_0; \rho_{3,0})$; $\rho_{3,0} \xleftarrow{r} \mathtt{RND}$;
    $\sigma_{1-b} \leftarrow \mathsf{lbs}_3(crs, vk, m_{1-b}, \rho_{1,1}, \mathbf{u}_1, \mathrm{s}_1; \rho_{3,1})$; $\rho_{3,1} \xleftarrow{r} \mathtt{RND}$;
    if $\mathtt{verify}(crs, vk, m_0, \sigma_0) \neq 1$ or $\mathtt{verify}(crs, vk, m_1, \sigma_1) \neq 1$, set $(\sigma_0, \sigma_1) \leftarrow (\perp, \perp)$;
    $b^* \leftarrow \mathcal{A}_3(state, \sigma_0, \sigma_1)$;
    if $b^* = b$, then return $1$;
```

Here KEYPAIR denotes the relation of verification-key and signing-key as defined by the key generation algorithm.

3.2 Lite Blind Signature Constructions

Lite blind signatures are simpler than full-fledged blind signatures (in the sense e.g., of [PS00, JLO97]) and thus can be more readily instantiated. In this subsection, first we present a generic construction, and then we present three concrete number theoretic constructions.

3.2.1 Generic Construction

For the first scheme, refer to the signature generation protocol in Figure 5, the CRSgen algorithm produces $crs = \langle pk_{\text{eqc}}, pk_{\text{exc}}, crs_{\text{nizk}}\rangle$; EQC is a commitment scheme with committing key pk_{eqc} and EQCcom is its committing algorithm; EXC is a commitment scheme with committing key pk_{exc} and EXCcom is its committing algorithm; NIZK is an NIZK argument scheme with CRS crs_{nizk} where NIZKprove is the proof generation algorithm and NIZKverify is the proof verification algorithm. The gen algorithm produces a key-pair $\langle vk, sk\rangle$ for a signature scheme SIG where SIGsign is the signature generation algorithm and SIGverify is the corresponding verification algorithm. The language $\mathcal{L}_R \overset{\text{def}}{=} \{x | (x, w) \in R\}$ where $R \overset{\text{def}}{=} \{(crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \rho_1, \rho_3) \,|\, \mathbf{u} = \texttt{EQCcom}(pk_{\text{eqc}}, m; \rho_1) \wedge \texttt{SIGverify}(vk, \mathbf{u}, \mathbf{s}) = 1 \wedge E = \texttt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)\}$. The verify algorithm given a message m and signature σ operates as follow: parse σ into E and ϖ, and check that $\texttt{NIZKverify}((crs, vk, E, m), \varpi) \overset{?}{=} 1$.

$$crs = \langle pk_{\text{eqc}}, pk_{\text{exc}}, crs_{\text{nizk}}\rangle$$

$$\begin{array}{lcl}
\boxed{U} & & \boxed{S} \\
\text{VerificationKey} = \langle vk\rangle & & \text{VerificationKey} = \langle vk\rangle \\
\text{Plaintext} = \langle m\rangle & & \text{SigningKey} = \langle sk\rangle \\
\hline
\rho_1 \overset{r}{\leftarrow} \texttt{RND};\ \mathbf{u} \leftarrow \texttt{EQCcom}(pk_{\text{eqc}}, m; \rho_1) & & \\
 & \xrightarrow{\quad \mathbf{u} \quad} & \rho_2 \overset{r}{\leftarrow} \texttt{RND} \\
\texttt{SIGverify}(vk, \mathbf{u}, \mathbf{s}) \overset{?}{=} 1 & \xleftarrow{\quad \mathbf{s} \quad} & \mathbf{s} \leftarrow \texttt{SIGsign}(vk, sk, \mathbf{u}; \rho_2) \\
\rho_3, \rho_4 \overset{r}{\leftarrow} \texttt{RND};\ E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3) & & \\
\varpi \leftarrow \texttt{NIZKprove}((crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \rho_1, \rho_3); \rho_4 & & \\
\quad :\ \mathbf{u} = \texttt{EQCcom}(pk_{\text{eqc}}, m; \rho_1)\ \wedge\ \texttt{SIGverify}(vk, \mathbf{u}, \mathbf{s}) = 1 & & \\
\quad \wedge E = \texttt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)) & & \\
\sigma \leftarrow E || \varpi & & \\
\texttt{verify}(crs, vk, m, \sigma) \overset{?}{=} 1 & & \\
\hline
\text{output } (m; \sigma) & &
\end{array}$$

Figure 5: A generic signature generation protocol.

Theorem 3.4. *The two-move signature generation protocol in Figure 5 is a lite blind signature as follows: it satisfies lite-unforgeability provided that* SIG *is EU-CMA secure,* EQC *is binding,* EXC *is extractable, and* NIZK *satisfies soundness; and it satisfies lite-blindness provided that* EQC *and* EXC *are hiding, and* NIZK *is zero-knowledge.*

Proof. **(I)** (sketch) Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to attack the signature scheme 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 $\texttt{SIGverify}(vk, \mathbf{u}_i, \mathbf{s}_i) = 1$. $\mathcal{B}$'s goal is to obtain a 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 $\mathbf{u}_i = \texttt{EQCcom}(pk_{\text{eqc}}, m_i; \rho_{1,i})$; 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 $\texttt{SIGverify}(vk, \mathbf{u}_i, \mathbf{s}_i) = 1$. At some point, $\mathcal{A}$ produces a pair $\langle m^*, \sigma^* \rangle$ where $\sigma^* = E^* || \varpi^*$, and $\texttt{NIZKverify}(crs, vk, E^*, m^*; \varpi^*) = 1$. 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 $\texttt{EQC}$ is binding, we have only negligible probability to find $m_i' \neq m_i$ such that $\mathbf{u}_i = \texttt{EQCcom}(pk_{\text{eqc}}, m_i; \rho_{1,i}) = \texttt{EQCcom}(pk_{\text{eqc}}, m_i'; \rho_{1,i}')$. Therefore m^* cannot be based on any queried $\mathbf{u}_i$. By using the extractable trapdoor xk_{exc} of commitment scheme $\texttt{EXC}$, $\mathcal{B}$ can extract $\mathbf{u}^*$ and $\mathbf{s}^*$ from E. Note that $\texttt{SIGverify}(vk, \mathbf{u}^*, \mathbf{s}^*) = 1$, and $\mathbf{u}^*$ has never been queried, $\mathcal{B}$ can obtain a forgery $\langle \mathbf{u}^*, \mathbf{s}^* \rangle$ for $\texttt{SIG}$.

(II) Note that the lite-blindness is implied by Proposition 3.10 and Theorem 3.11 below. □

3.2.2 Construction based on Camenisch-Lysyanskaya Signature and Linear Encryption

Lite blind signature construction that we present in Figure 6 uses the LRSW assumption [LRSW99] and the DLDH assumption [BBS04] and is based on the blind signature that appeared in [KZ06]. The $\texttt{gen}$ algorithm produces $vk = \langle p, g, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}; X, Y \rangle$ and $sk = \langle x, y \rangle$, and the $\texttt{verify}$ algorithm given a message m and signature $\sigma = \langle a, b, c \rangle$, responds as follows: check that $\hat{\mathsf{e}}(a, Y) = \hat{\mathsf{e}}(g, b)$ and $\hat{\mathsf{e}}(X, a)\hat{\mathsf{e}}(X, b)^m = \hat{\mathsf{e}}(g, c)$.

$\boxed{U}$
$vk = \langle p, g, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}; X, Y \rangle$
$msg = \langle m \rangle$, $m \in \mathbb{Z}_p$

$\boxed{S}$
$vk = \langle p, g, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}; X, Y \rangle$
$sk = \langle x, y \rangle$

$\zeta \xleftarrow{\text{r}} \texttt{RND}$
$(PK_U, SK_U) \leftarrow \texttt{gen}_{\text{LE}}(1^\lambda, \zeta)$
$PK_U = \langle t, v, w \rangle$, $SK_U = \langle \delta, \xi \rangle$
$k, l \xleftarrow{\text{r}} \mathbb{Z}_p$; $\theta \xleftarrow{\text{r}} \mathbb{G}_1$
$T \leftarrow t^k$; $V \leftarrow v^l$; $W \leftarrow \theta^m w^{k+l}$

$$\xrightarrow{PK_U, \langle \theta, T, V, W \rangle}$$

$\alpha', k', l' \xleftarrow{\text{r}} \mathbb{Z}_p$
$a' \leftarrow \theta^{\alpha'}$; $b' \leftarrow \theta^{y\alpha'}$
$T' \leftarrow T^{xy\alpha'} t^{k'\alpha'}$; $V' \leftarrow V^{xy\alpha'} v^{l'\alpha'}$
$W' \leftarrow W^{xy\alpha'} \theta^{x\alpha'} w^{k'\alpha' + l'\alpha'}$

$$\xleftarrow{a', b', T', V', W'}$$

$\alpha \xleftarrow{\text{r}} \mathbb{Z}_p$
$a \leftarrow (a')^\alpha$; $b \leftarrow (b')^\alpha$; $c \leftarrow \left(\frac{W'}{T'^\delta V'^\xi} \right)^\alpha$
$\sigma \leftarrow \langle a, b, c \rangle$

---

output $(m; \sigma)$

Figure 6: Lite blind signature protocol based on [KZ06]. Here $\texttt{gen}_{\text{LE}}$ is Linear Encryption key generation.

Theorem 3.5. *The two-move protocol of Figure 6 is a lite blind signature as follows: it satisfies lite-unforgeability under the LRSW assumption; and it satisfies lite-blindness under the DLDH assumption.*

Proof. **(I)** (sketch) For lite-unforgeability, we can follow the unforgeability proof in [KZ06]. Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to break the LRSW assumption.

Note that $\mathcal{B}$ is given $vk = \langle p, g, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}; X, Y \rangle$, and is allowed to query the oracle $\mathcal{O}_{X,Y}$ (as defined in the LRSW assumption) with m_i and obtain $\sigma_i = \langle a_i, b_i, c_i \rangle$ such that $\texttt{verify}(vk, m_i, \sigma_i) = 1$. $\mathcal{B}$'s goal is to obtain a pair $\langle m^*, \sigma^* \rangle$ where m^* is not queried. $\mathcal{B}$ runs a copy of $\mathcal{A}$ inside, and supplies $\mathcal{A}$ with vk; note that $\mathcal{A}$ now may query with $(m_i, \rho_{1,i})$ where $\rho_{1,i} = \langle \zeta_i, k_i, l_i, \theta_i \rangle$; $\mathcal{B}$ can "recover" the key pair $\langle PK_U, SK_U \rangle$. Given such query, $\mathcal{B}$ queries his oracle $\mathcal{O}_{X,Y}$ with m_i, obtains $\sigma_i = \langle a_i, b_i, c_i \rangle$ where $\mathtt{verify}(vk, m_i, \sigma_i) = 1$, and then compute $a'_i \leftarrow a_i$, $b'_i \leftarrow b_i$, $W'_i \leftarrow c_i w^{k'+l'}$, $T'_i \leftarrow t^{k'}$, $V'_i \leftarrow v^{l'}$, where $k', l' \xleftarrow{\mathtt{r}} \mathbb{Z}_p$. $\mathcal{B}$ returns $\langle a'_i, b'_i, W'_i, T'_i, V'_i \rangle$ to $\mathcal{A}$. At some point, $\mathcal{A}$ produces a pair $\langle m^*, \sigma^* \rangle$ where m^* has never been queried to $\mathcal{O}_{X,Y}$, and $\sigma^* = \langle a^*, b^*, c^* \rangle$, $\mathtt{verify}(vk, m^*, \sigma^*) = 1$. $\mathcal{B}$ outputs such pair, and this breaks the LRSW assumption.

(II) (sketch) For lite-blindness, we follow the blindness proof in [KZ06]. The proof idea is when the adversary initials two user instantiations with two messages and verification key (m_0, m_1, vk), the simulator as in the lite-blindness definition, gives the verification key to two users, and randomly selects $b \xleftarrow{\mathtt{r}} \{0,1\}$ and also supplies, say, the left user with m_b and the right user with m_{1-b}.

Notice that the involved Linear Encryption is CPA-secure based on the DLDH assumption, there is only negligible probability that the adversary can detect if the simulator simulates the two user instantiations with random messages, i.e. the two messages m_0, m_1 will not be used, or not.

At some point, the adversary will reply the two users following signer response function; and the adversary also returns the random coins used and the signing key. If the simulator verify the random coins and the signing key are consistent, then the simulator uses the signing key to produces two valid signatures $\langle \sigma_0, \sigma_1 \rangle$ for $\langle m_0, m_1 \rangle$ and returns $\langle \sigma_0, \sigma_1 \rangle$ to the adversary; otherwise simulator returns $\langle \perp, \perp \rangle$ to the adversary. At this point the simulator finishes the simulation with the adversary and finally the adversary is required to guess the coins b. Note that in the interactions between the user instantiations and the adversary, coins b is not used. So the adversary cannot win the game under the DLDH assumption. $\square$

3.2.3 Construction based on Waters Signature

In Figure 7 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 CDH assumption and is based on Waters' digital signature scheme [Wat05]. Note that our lite blind signature construction here is based on the blind signature construction in [Oka06] (refer to section 10, page 31 in [Oka06]). In this setting the $\mathtt{CRSgen}$ algorithm produces $crs = \langle p, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}, g, g_2, v, u_1, \ldots, u_n \rangle$, the $\mathtt{gen}$ algorithm produces a key-pair $vk = \langle g_1 \rangle$ where $g_1 = g^{\alpha}$, $sk = \langle g_2^{\alpha} \rangle$, and the $\mathtt{verify}$ algorithm given an n-bit message m and signature $\sigma = \langle \sigma_1, \sigma_2 \rangle$, responds as follow: check that $\hat{\mathsf{e}}(\sigma_1, g)/\hat{\mathsf{e}}(\sigma_2, v\prod_{j=1}^{n} u_j^{m_j}) = \hat{\mathsf{e}}(g_1, g_2)$.

Remark 3.6. We remark that Waters' signature scheme is not secure without any restriction of message m. For some m satisfies $v\prod_{j=1}^{n} u_j^{m_j} = 1$, the signature for m is $\sigma = \langle \sigma_1, \sigma_2 \rangle$, where $\sigma_1 = g_2^{\alpha}$ and $\sigma_2 = g^r$ for some randomly selected $r \xleftarrow{\mathtt{r}} \mathbb{Z}_p$. Note that $\sigma_1 = sk$, and a signature for such m will reveal the signing key!

The blind signature scheme based on Waters signature in [Oka06] also suffers similar attack. An adversarial user may select an m such that $v\prod_{j=1}^{n} u_j^{m_j} = 1$ and the user based on such m computes $W = 1$ and sends W to an honest signer; the signer will return $Y_1 = g_2^{\alpha} W^r$ and $Y_2 = g^r$ where r is randomly selected. Note that $Y_1 = g_2^{\alpha} W^r = g_2^{\alpha}$ which is the signing key!

A simple way to avoid such attack is to check if m satisfies $v\prod_{j=1}^{n} u_j^{m_j} = 1$, and refuse to produce signature for such "bad" message. But this will change the message space of the signature scheme.

An alternative way is to require that $p > 2^{\lambda_p + n}$ where $\lambda_p = \Omega(\log^2 \lambda)$. This requirement will not allow the adversary to select m such that $v\prod_{j=1}^{n} u_j^{m_j} = 1$ except negligible probability. From $v\prod_{j=1}^{n} u_j^{m_j} = 1$, we can have $\tau_v + \sum_{j=1}^{n} \tau_{u_j} m_j = 0$, where $v = g^{\tau_v}, u_1 = g^{\tau_{u_1}}, \ldots, u_n = g^{\tau_{u_n}}$. Notice that given the length of m is n bits, there are at most 2^n potential m satisfies $\tau_v + \sum_{j=1}^{n} \tau_{u_j} m_j = 0$. Also notice that $\tau_v \in \mathbb{Z}_p$ has 2^{λ_p+n} potential values. So the probability that $\tau_v + \sum_{j=1}^{n} \tau_{u_j} m_j = 0$ holds is $\frac{2^n}{2^{\lambda_p+n}}$, i.e. $\frac{1}{2^{\lambda_p}}$, which is negligible.

In our lite blind signature construction based on Waters' signature, we require that $p > 2^{\lambda_p+n}$. Still we require that $W \neq 1$ to avoid the adversarial selection of $t = 0$.

$$crs = \langle p, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}, g, g_2, v, u_1, \ldots, u_n \rangle$$

$\boxed{U}$		$\boxed{S}$
$vk = \langle g_1 = g^\alpha \rangle$		$vk = \langle g_1 = g^\alpha \rangle$
$msg = \langle m \rangle$		$sk = \langle g_2^\alpha \rangle$
$t \xleftarrow{r} \mathbb{Z}_p$; $W \leftarrow (v \prod_{j=1}^n u_j^{m_j})^t$	$\xrightarrow{W}$	$W \neq^? 1$
		$r \xleftarrow{r} \mathbb{Z}_p$
$s \xleftarrow{r} \mathbb{Z}_p$	$\xleftarrow{Y_1, Y_2}$	$Y_1 \leftarrow g_2^\alpha W^r$; $Y_2 \leftarrow g^r$
$\sigma_1 \leftarrow Y_1 (v \prod_{j=1}^n u_j^{m_j})^s$; $\sigma_2 \leftarrow Y_2^t g^s$		
$\sigma \leftarrow \langle \sigma_1, \sigma_2 \rangle$		
$\mathtt{verify}(crs, vk, m, \sigma) =^? 1$		
output $(m; \sigma)$		

Figure 7: Signature generation protocol based on Waters digital signature [Wat05]. Here m is n-bit message, and m_j is the jth bit of m, and $p > 2^{\lambda_p + n}$ where $\lambda_p = \Omega(\log^2 \lambda)$.

Theorem 3.7. *The two-move protocol of Figure 7 is a lite blind signature as follows: it satisfies lite-unforgeability under the CDH assumption; and it satisfies lite-blindness unconditionally.*

Proof. **(I)** The proof idea of lite-unforgeability is similar to the proof for Waters IBE (refer to page 5, in [Wat05]). Note that here the lite blind signature will not suffer the attack in Remark 3.6. Assume $\mathcal{A}$ is a lite-unforgeability adversary. We construct algorithm $\mathcal{B}$ to break the CDH assumption by inputting $\langle g, A = g^a, B = g^b \rangle$ and outputting g^{ab}.

Assume $\mathcal{A}$ at most queries q times. $\mathcal{B}$ first sets $l = 4q$, randomly selects $k \xleftarrow{r} \{0, \ldots, n\}$, and randomly selects $w, x_1, \ldots, x_n \xleftarrow{r} \mathbb{Z}_l$, $z, y_1, \ldots, y_n \xleftarrow{r} \mathbb{Z}_p$. For n-bit message m, define $F(m) = (p - kl) + w + \sum_{j=1}^n x_j m_j$ and $J(m) = z + \sum_{j=1}^n y_j m_j$, where m_j is the jth bit of m. Now $\mathcal{B}$ gives $\langle g, g_1, g_2, v, u_1, \ldots, u_n \rangle$ to $\mathcal{A}$ where $g_1 = A$, $g_2 = B$, $v = g_2^{p-kl+w} g^z$, $u_j = g_2^{x_j} g^{y_j}$ for $j = 1, \ldots, n$.

Now $\mathcal{A}$ queries the lite-unforgeability signing oracle with $\langle m, t \rangle$. Here $W = (v \prod_{j=1}^n u_j^{m_j})^t$. $\mathcal{B}$ should reply with $\langle Y_1, Y_2 \rangle$ in the form of $\langle g_2^a (v \prod_{j=1}^n u_j^{m_j})^{rt}, g^r \rangle$, or in the form of $\langle g_2^a (v \prod_{j=1}^n u_j^{m_j})^r, g^{r/t} \rangle$, where r is randomly selected. Note that now $v \prod_{j=1}^n u_j^{m_j} = 1$ holds with only negligible probability based on the discussion in Remark 3.6. So, we can compute $Y_1 = g_1^{-J(m)/F(m)} (v \prod_{j=1}^n u_j^{m_j})^r$, and $Y_2 = g_1^{-1/(F(m)t)} g^{r/t}$. Let $\widehat{r} = r - a/F(m)$. Then $Y_1 = g_2^a (v \prod_{j=1}^n u_j^{m_j})^{\widehat{r}}$, and $Y_2 = g^{\widehat{r}/t}$ which is a valid response. This finishes the simulation of signing oracle.

When the adversary outputs a valid message-signature pair, m^* and $\sigma^* = \langle \sigma_1^*, \sigma_2^* \rangle$, where $\sigma_1^* = g_2^a (v \prod_{j=1}^n u_j^{m_j^*})^{\widetilde{r}}$ and $\sigma_2^* = g^{\widetilde{r}}$ for some $\widetilde{r}$. If $F(m^*) \neq 0$, aborts. Otherwise $F(m^*) = 0$, and we can extract g_2^a which is g^{ab}; this finishes the proof.

(II) We ignore the proof of lite-blindness which is implied by Proposition 3.10 and Theorem 3.12 below. $\square$

3.2.4 Construction based on Okamoto Signature

In Figure 8 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 digital signature scheme [Oka06]. Note that our lite blind signature construction here is based on the blind signature construction in [Oka06] (refer to section 6, page 16 in

[Oka06]). In this setting the CRSgen algorithm produces $crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$, 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 follow: 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 $\hat{\mathsf{e}}(\varsigma, \alpha) = \hat{\mathsf{e}}(g_1, g_2^m u_2 v_2^\beta)$, $\hat{\mathsf{e}}(V_1, \alpha) = \hat{\mathsf{e}}(\psi(X), X) \cdot \hat{\mathsf{e}}(g_1, V_2)$.

$$crs = \langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2 \rangle$$

$$\begin{array}{lcl}
\boxed{U} & & \boxed{S} \\
vk = \langle X = g_2^x \rangle & & vk = \langle X = g_2^x \rangle \\
msg = \langle m \rangle, m \in \mathbb{Z}_p & & sk = \langle x \rangle \\
\hline
t, s \xleftarrow{r} \mathbb{Z}_p; W \leftarrow g_1^{mt} u_1^t v_1^{st} & \xrightarrow{\quad W \quad} & \\
 & & r, l \xleftarrow{r} \mathbb{Z}_p \text{ s.t. } x + r \neq 0 \\
f, h \xleftarrow{r} \mathbb{Z}_p; \varsigma \leftarrow Y^{\frac{1}{ft} \bmod p} & \xleftarrow{\quad Y,l,r \quad} & Y \leftarrow (W v_1^l)^{\frac{1}{x+r}} \\
\alpha \leftarrow X^f g_2^{fr}; \beta \leftarrow s + \frac{l}{t} \bmod p & & \\
V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h; V_2 \leftarrow X^{fh+r} g_2^{frh} & & \\
\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle & & \\
\mathtt{verify}(crs, vk, m, \sigma) \overset{?}{=} 1 & & \\
\hline
\text{output } (m; \sigma) & &
\end{array}$$

Figure 8: Signature generation protocol based on Okamoto digital signature [Oka06].

Theorem 3.8. *The two-move protocol of Figure 8 is a lite blind signature as follows: it satisfies lite-unforgeability under the 2SDH assumption; and it satisfies lite-blindness unconditionally.*

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 $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2, X \rangle$ 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 `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) We ignore the proof of lite-blindness which is implied by Proposition 3.10 and Theorem 3.13.

□

3.3 Equivocal Blind Signatures

In this subsection, we introduce a new property, *equivocality*, for blind signature schemes. For simplicity we will only consider the property over two-move protocols following the skeleton of Figure 4. We call a lite blind signature scheme with the equivocality property as an *equivocal lite blind signature scheme*.

Informally an equivocal blind signature scheme is accompanied by a simulator procedure $\mathcal{I}$ which can produce communication transcripts without using the plaintext m and furthermore can "explain" the communication transcripts to any adversarially selected m even after the signature σ for m has been generated. Considering that in the equivocality definition, m need not be used to produce communication transcripts, we can prove that equivocality implies lite-blindness. Thus an equivocal lite blind signature needs to satisfy only two properties: equivocality and lite-unforgeability. 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 9).

Definition 3.9 (Equivocality). We say a lite blind signature scheme 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 $\mathsf{Adv}_{\mathsf{eq}}^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$,

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

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

- Upon receiving message (i, m, vk) from $\mathcal{A}$, select $\rho_1 \xleftarrow{\mathtt{r}} \mathtt{RND}$ and compute $\mathbf{u} \leftarrow \mathsf{lbs}_1(crs, vk, m; \rho_1)$, record $\langle i, m, vk, \mathbf{u}, \rho_1\rangle$ into $history$, and return message $(i, \mathbf{u})$ to $\mathcal{A}$.
- Upon receiving message $(i, \mathrm{s}, \rho_2, sk)$ from $\mathcal{A}$, if there exists a record $\langle i, m, vk, \mathbf{u}, \rho_1\rangle$ in $history$ and $(vk, sk) \in$ KeyPair and $\mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2)$, then select $\rho_3 \xleftarrow{\mathtt{r}} \mathtt{RND}$ and compute $\sigma \leftarrow \mathsf{lbs}_3(crs, vk, m, \rho_1, \mathbf{u}, \mathbf{s}; \rho_3)$; if $\mathtt{verify}(crs, vk, m, \sigma) = 1$, then update $\langle i, m, vk, \mathbf{u}, \rho_1\rangle$ in $history$ into $\langle i, m, vk, \mathbf{u}, \sigma, \rho_1, \rho_3\rangle$, 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 $(i, history)$.

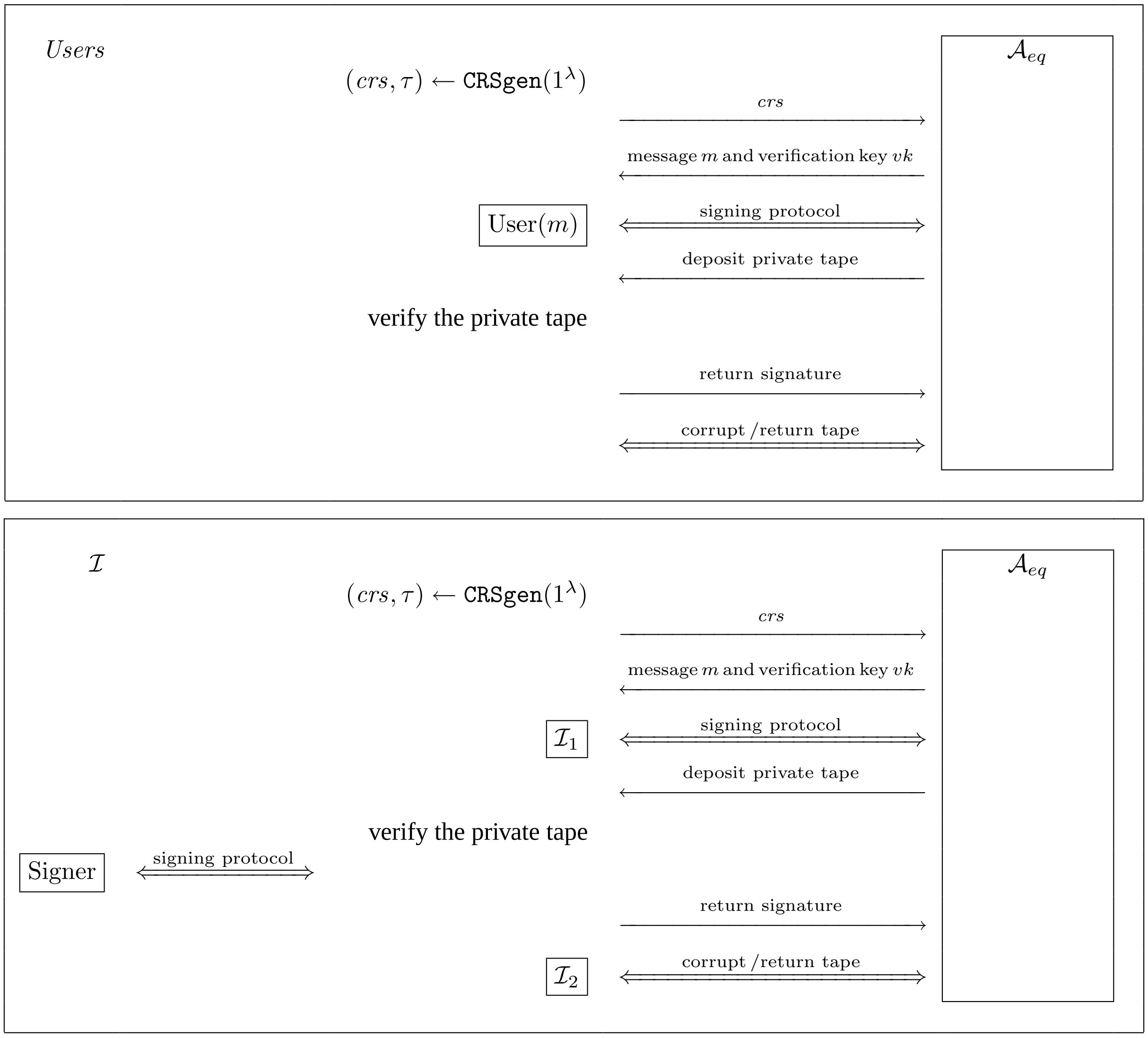


Figure 9: The two worlds an equivocality adversary is asked to distinguish in Definition 3.9.

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 $(vk,sk) \in$ KEYPAIR and $\mathbf{s} = \mathsf{lbs}_2(crs,vk,\mathbf{u},sk;\rho_2)$, then select $\gamma \xleftarrow{r}$ $\texttt{RND}$ and compute $\sigma \leftarrow \texttt{sign}(crs,vk,sk,m,\gamma)$, and 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 message (i,σ) to $\mathcal{A}$; otherwise return $\mathcal{A}$ with message $(i,\perp)$.
- Upon receiving message (i,open), if there exists a record $\langle i,m,vk,\mathbf{u},aux\rangle$ in $temp$ then run $\rho_1 \leftarrow \mathcal{I}_2(i,temp)$ and record $\langle i,m,vk,\mathbf{u},\rho_1\rangle$ into $history$, and return $\mathcal{A}$ with message $(i,history)$; if there exists a record $\langle i,m,vk,\mathbf{u},aux;\mathbf{s},sk,\rho_2;\sigma,\gamma\rangle$ in $temp$, then run $(\rho_1,\rho_3) \leftarrow \mathcal{I}_2(i,temp)$ and record $\langle i,m,vk,\mathbf{u},\sigma,\rho_1,\rho_3\rangle$ into $history$, and return $\mathcal{A}$ with message $(i,history)$.

Proposition 3.10. *Consider a tuple* $\langle \texttt{CRSgen}, \texttt{gen}, \mathsf{lbs}_1, \mathsf{lbs}_2, \mathsf{lbs}_3, \texttt{verify}\rangle$*, cf. Figure 4, that satisfies equivocality. Then it necessarily also satisfies lite-blindness.*

Proof. (sketch) Here we show equivocality implies lite-blindness, which means if there is no equivocality attacker, then there is no lite-blindness attacker. So we need to construct an equivocality attacker $\mathcal{A}_{eq}$ based on a lite-blindness attacker $\mathcal{A}_{lb}$. Next we give a description of such construction in Figure 10.

$\mathcal{A}_{eq}$ obtains crs from his oracle, i.e. the oracle $Users(crs, \cdot)$ or the oracle $\mathcal{I}(crs, \tau, \cdot)$; then $\mathcal{A}_{eq}$ gives such crs to $\mathcal{A}_{lb}$ and obtains two messages m_0, m_1 and verification key vk from $\mathcal{A}_{lb}$. Now $\mathcal{A}_{eq}$ flips a coin b and initials two query/response sessions with his oracle: query with m_b and obtain $\mathbf{u}_0$, and query with m_{1-b} and obtain $\mathbf{u}_1$.

Then $\mathcal{A}_{eq}$ sends $\mathcal{A}_{lb}$ with $\mathbf{u}_0$ and $\mathbf{u}_1$ and obtains response $(\mathbf{s}_0, \mathbf{s}_1, \rho_{2,0}, \rho_{2,1}, sk)$. $\mathcal{A}_{eq}$ verifies the response, and if the response from $\mathcal{A}_{lb}$ is valid, $\mathcal{A}_{eq}$ initials two sessions with his oracle: query with $(\mathbf{s}_0, \rho_{2,0}, sk)$ and obtain σ_b, and query with $(\mathbf{s}_1, \rho_{2,1}, sk)$ and obtain σ_{1-b}.

If both σ_0 and σ_1 are produced, then $\mathcal{A}_{eq}$ sends them to $\mathcal{A}_{lb}$ and obtains $\mathcal{A}_{lb}$'s guess of random bit b. If $\mathcal{A}_{lb}$'s guess is correct, then $\mathcal{A}_{eq}$ outputs 1, otherwise 0.

Notice that in the case that $\mathcal{A}_{eq}$ is interacting with his oracle $\mathcal{I}(crs, \tau, \cdot)$, $\mathbf{u}_0$ and $\mathbf{u}_1$ are not computed based on m_0 or m_1, so $\mathcal{A}_{lb}$ has only probability $1/2$ to figure out the bit b. In the case that $\mathcal{A}_{eq}$ is interacting with his oracle $Users(crs, \cdot)$, $\mathcal{A}_{lb}$ can figure out the bit b with probability $1/2 + \epsilon$ where ϵ is non-negligible because $\mathcal{A}_{lb}$ is a successful lite-blindness attacker. Now we can compute $\mathsf{Adv}_{\mathrm{eq}}^{\mathcal{A}_{eq}} = \epsilon$ which is non-negligible.

$\square$

Equivocality is in fact a strict strengthening of the lite-blindness property: for example the lite blind signature in Section 3.2.2 satisfies lite-blindness but is not equivocal due to the employment of an encryption of the message in the first step of the signature generation protocol. Next we show the lite blind signature constructions in Section 3.2.1, Section 3.2.3 and Section 3.2.4 are equivocal, respectively.

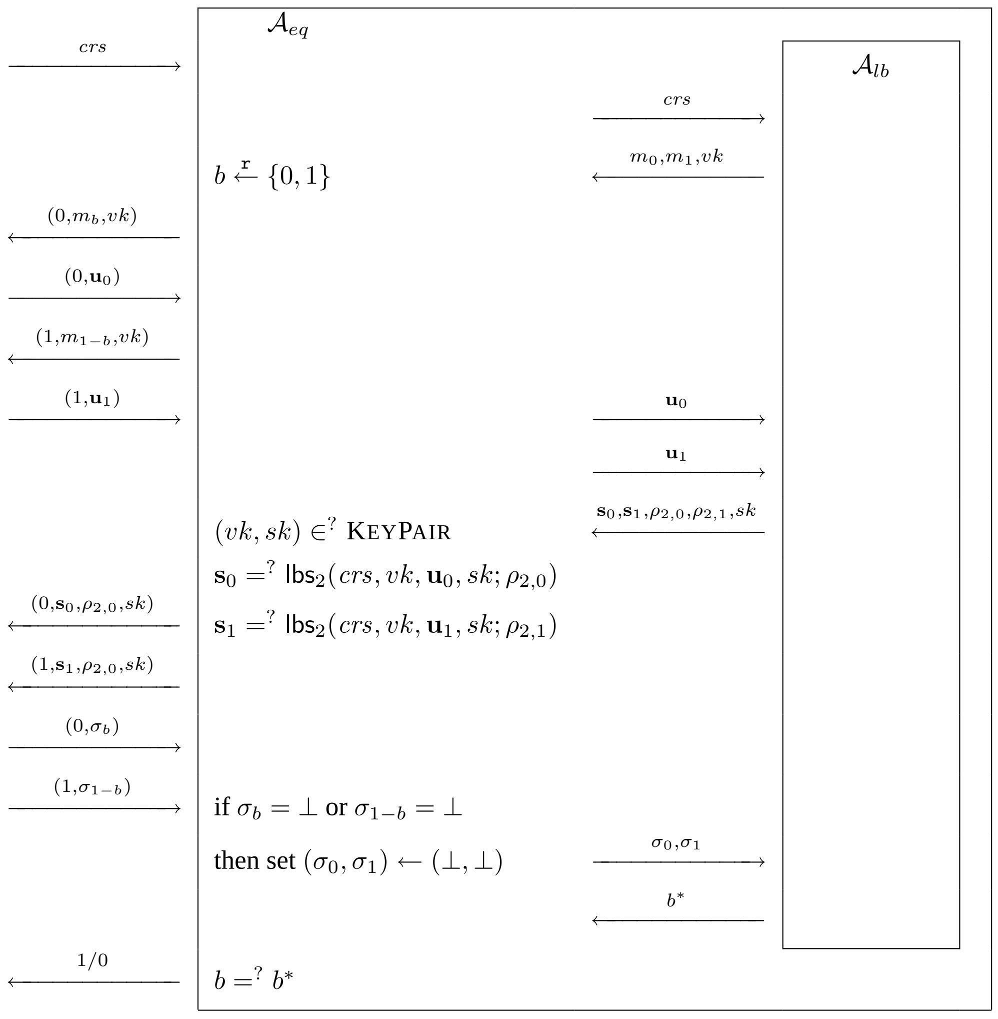
Theorem 3.11. *The two-move signature generation protocol described in Figure 5 is equivocal provided that* $\texttt{EQC}$ *is equivocal,* $\texttt{EXC}$ *is hiding, and* $\texttt{NIZK}$ *is non-erasure zero-knowledge.*

Proof. Here we prove the lite blind signature in Figure 3.11 satisfies equivocality defined in Definition 3.9. Generate $(crs, \tau) \leftarrow \texttt{CRSgen}(1^\lambda)$; here $crs = \langle pk_{\mathrm{eqc}}, pk_{\mathrm{exc}}, crs_{\mathrm{nizk}}\rangle$, $\tau = \langle ek_{\mathrm{eqc}}, \tau_{\mathrm{nizk}}\rangle$ where ek_{eqc} is the equivocal key for commitment scheme $\texttt{EQC}$ and τ_{nizk} is the trapdoor for state reconstruction of $\texttt{NIZK}$. Send crs to $\mathcal{A}$.

For completeness, we first describe how oracle $Users(crs, \cdot)$ operates.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, select $\rho_1 \stackrel{r}{\leftarrow} \texttt{RND}$ and compute $\mathbf{u} \leftarrow \texttt{EQCcom}(pk_{\mathrm{eqc}}, m; \rho_1)$, record $\langle i, m, vk, \mathbf{u}, \rho_1\rangle$ into $history$, 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 $history$ and $(vk, sk) \in \textsc{KeyPair}$ and $\mathbf{s} = \texttt{SIGsign}(vk, sk, \mathbf{u}; \rho_2)$, then select $\rho_3, \rho_4 \stackrel{r}{\leftarrow} \texttt{RND}$ and compute $E \leftarrow \texttt{EXCcom}(pk_{\mathrm{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)$, and $\varpi \leftarrow \texttt{NIZKprove}((crs, vk, E, m), (\mathbf{u}, \mathbf{s}, \rho_1, \rho_3); \rho_4 : \mathbf{u} = \texttt{EQCcom}(pk_{\mathrm{eqc}}, m; \rho_1) \wedge \texttt{SIGverify}(vk, \mathbf{u}, \mathbf{s}) = 1 \wedge E = \texttt{EXCcom}(pk_{\mathrm{exc}}, \mathbf{u}, \mathbf{s}; \rho_3))$ and set $\sigma \leftarrow E||\varpi$. If $\texttt{verify}(crs, vk, m, \sigma) = 1$, then update $\langle i, m, vk, \mathbf{u}, \rho_1\rangle$ in $history$ 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 $(i, history)$.

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



$\mathcal{A}_{eq}$
$\mathcal{A}_{lb}$
crs
crs
m_0, m_1, vk
$b \xleftarrow{r} \{0,1\}$
$(0, m_b, vk)$
$(0, \mathbf{u}_0)$
$(1, m_{1-b}, vk)$
$(1, \mathbf{u}_1)$
$\mathbf{u}_0$
$\mathbf{u}_1$
$\mathbf{s}_0, \mathbf{s}_1, \rho_{2,0}, \rho_{2,1}, sk$
$(vk, sk) \in^? \text{KeyPair}$
$\mathbf{s}_0 =^? \mathsf{lbs}_2(crs, vk, \mathbf{u}_0, sk; \rho_{2,0})$
$\mathbf{s}_1 =^? \mathsf{lbs}_2(crs, vk, \mathbf{u}_1, sk; \rho_{2,1})$
$(0, \mathbf{s}_0, \rho_{2,0}, sk)$
$(1, \mathbf{s}_1, \rho_{2,0}, sk)$
$(0, \sigma_b)$
$(1, \sigma_{1-b})$
if $\sigma_b = \perp$ or $\sigma_{1-b} = \perp$
then set $(\sigma_0, \sigma_1) \leftarrow (\perp, \perp)$
σ_0, σ_1
b^*
1/0
$b =^? b^*$

Figure 10: Construction of equivocality attacker from lite-blindness attacker.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, run $(\mathbf{u}, aux) \leftarrow \mathcal{I}_1(crs, \tau, vk)$, i.e. compute $(\mathbf{u}, state_{\text{eqc}}) \leftarrow \texttt{EQCfake}(pk_{\text{eqc}}, ek_{\text{eqc}})$, and record $\langle i, m, vk, \mathbf{u}, state_{\text{eqc}} \rangle$ into $temp$, and return message $(i, \mathbf{u})$ to $\mathcal{A}$; here `EQCfake` is an algorithm of producing a fake commitment based on `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 $(vk, sk) \in \text{KeyPair}$ and $\mathbf{s} = \texttt{SIGsign}(vk, sk, \mathbf{u}; \rho_2)$; then compute $\sigma \leftarrow E || \varpi$, where $E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, \mathbf{u}, \mathbf{s}; \rho_3)$ and $(\varpi, state_{\text{nizk}}) \leftarrow \texttt{NIZKsimulate}((crs, vk, E, m), \tau_{\text{nizk}})$ and $\rho_3 \xleftarrow{r} \texttt{RND}$. 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 `NIZKsimulate` is used to produce a simulated proof for `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 $history$, and return $\mathcal{A}$ with message $(i, history)$; here `EQCequivocate` is equivocation algorithm associated with the equivocal commitment `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*, and return $\mathcal{A}$ with message $(i, history)$; here NIZKreconstruct is state reconstruction algorithm associated with 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 NIZK and hiding property of EXC respectively. Therefore $\mathcal{A}$ will output 1 with the same probability except negligible probability in the two interactions. $\square$

Theorem 3.12. *The two-move signature generation protocol described in Figure 7 is equivocal (unconditionally).*

Proof. Here we prove the lite blind signature based on Waters signature satisfies equivocality defined in Definition 3.9. Generate $(crs, \tau) \leftarrow \texttt{CRSgen}(1^\lambda)$; here $crs = \langle p, \mathbb{G}_1, \mathbb{G}_T, \hat{\mathsf{e}}, g, g_2, v, u_1, \ldots, u_n\rangle$, $\tau = \langle \tau_v, \tau_{u_1}, \ldots, \tau_{u_n}\rangle$ such that $v = g^{\tau_v}$, $u_1 = g^{\tau_{u_1}}, \ldots, u_n = g^{\tau_{u_n}}$. Send crs to $\mathcal{A}$.

For completeness, we first describe oracle $Users(crs, \cdot)$.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, where $vk = \langle g_1 = g^\alpha\rangle$ select $t \xleftarrow{\text{r}} \mathbb{Z}_p$ and compute $W \leftarrow (v \prod_{j=1}^n u_j^{m_j})^t$, record (i, m, vk, W, t) into memory *history*, and return message (i, W) to $\mathcal{A}$.
- Upon receiving message $(i, \langle Y_1, Y_2\rangle, sk, r)$ from $\mathcal{A}$, where $sk = \langle g_2^\alpha\rangle$, if there exists a record (i, m, vk, W, t) in *history* and $\hat{\mathsf{e}}(sk, g) = \hat{\mathsf{e}}(g_2, g_1)$, then compute $\sigma \leftarrow \langle \sigma_1, \sigma_2\rangle$ where $\sigma_1 \leftarrow Y_1(v \prod_{j=1}^n u_j^{m_j})^s$, $\sigma_2 \leftarrow Y_2^t g^s$, and $s \xleftarrow{\text{r}} \mathbb{Z}_p$. If $\texttt{verify}(crs, vk, m, \sigma) = 1$, then update (i, m, vk, W, t) in *history* into $(i, m, vk, W, \langle \sigma_1, \sigma_2\rangle, \langle t, s\rangle)$, 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 $(i, history)$.

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 n-bit random message $\widetilde{m}$, select $\widetilde{t} \xleftarrow{\text{r}} \mathbb{Z}_p$, compute $W = (v \prod_{j=1}^n u_j^{\widetilde{m}_j})^{\widetilde{t}}$, and record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}\rangle)$ into *temp*, and return message (i, W) to $\mathcal{A}$.
- Upon receiving message $(i, \langle Y_1, Y_2\rangle, sk, r)$ from $\mathcal{A}$, if there exists a record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}\rangle)$ in *temp* and $\hat{\mathsf{e}}(sk, g) = \hat{\mathsf{e}}(g_1, g_2)$ and $Y_1 = sk \cdot W^r$ and $Y_2 = g^r$, then compute $\sigma \leftarrow \langle \sigma_1, \sigma_2\rangle$, where $\sigma_1 \leftarrow sk \cdot (v \prod_{j=1}^n u_j^{m_j})^{\widehat{t}}$ and $\sigma_2 \leftarrow g^{\widehat{t}}$ and $\widehat{t} \xleftarrow{\text{r}} \mathbb{Z}_p$. Update $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}\rangle)$ in *temp* into $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}\rangle; \langle Y_1, Y_2\rangle, sk, r; \langle \sigma_1, \sigma_2\rangle, \widehat{t})$, 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 $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}\rangle)$ in *temp* then run $t \leftarrow \mathcal{I}_2(i, temp)$ as: compute t from $(\tau_v + \sum_{j=1}^n \tau_{u_j} m_j)t = (\tau_v + \sum_{j=1}^n \tau_{u_j} \widetilde{m}_j)\widetilde{t}$ and record (i, m, vk, W, t) into *history*, and return $\mathcal{A}$ with message $(i, history)$;

– if there exists a record $(i, m, vk, W, \tau, \langle \widetilde{m}, \widetilde{t}\rangle; \langle Y_1, Y_2\rangle, sk, r; \langle \sigma_1, \sigma_2\rangle, \widehat{t})$ in *temp*, then run $(t, s) \leftarrow \mathcal{I}_2(i, temp)$ as: compute t based on equation $(\tau_v + \sum_{j=1}^{n} \tau_{u_j} m_j)t = (\tau_v + \sum_{j=1}^{n} \tau_{u_j} \widetilde{m}_j)\widetilde{t}$, compute s based on equation $\widehat{t} = rt + s$, and we have $t \leftarrow \frac{(\tau_v + \sum_{j=1}^{n} \tau_{u_j} \widetilde{m}_j)\widetilde{t}}{(\tau_v + \sum_{j=1}^{n} \tau_{u_j} m_j)}$, and $s \leftarrow \widehat{t} - rt$, and record $(i, m, vk, W, \sigma, \langle t, s\rangle)$ into *history*, and return $\mathcal{A}$ with message $(i, history)$.

We define **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 **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 **E** is negligible.

Based on the discussion in Remark 3.6, there is only negligible probability that $v \prod_{j=1}^{n} u_j^{m_j} = 1$, i.e. $\tau_v + \sum_{j=1}^{n} \tau_{u_j} m_j = 0$. So $\mathcal{I}(crs, \tau, \cdot)$ can reconstruct t, s except negligible probability, which means the event **E** can happen with only negligible probability. Therefore the adversary $\mathcal{A}$ cannot distinguish the interaction with the oracle $Users(crs, \cdot)$ and with the oracle $\mathcal{I}(crs, \tau, \cdot)$. This completes the proof.

□

Theorem 3.13. *The two-move signature generation protocol described in Figure 8 is equivocal (unconditionally).*

Proof. Here we prove the lite blind signature based on Okamoto signature satisfies equivocality defined in Definition 3.9. 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 $\mathcal{A}$.

For completeness we first describe the oracle $Users(crs, \cdot)$.

- Upon receiving message (i, m, vk) from $\mathcal{A}$, where $vk = \langle X\rangle$ select $t, s \xleftarrow{r} \mathbb{Z}_p$ 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*, 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* 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 $f, h \xleftarrow{r} \mathbb{Z}_p$. If $\mathtt{verify}(crs, vk, m, \sigma) = 1$, then update $(i, m, vk, W, \langle t, s\rangle)$ in *history* 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 $(i, history)$.

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 $\widetilde{m}, \widetilde{t}, \widetilde{s} \xleftarrow{r} \mathbb{Z}_p$, 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 $\texttt{verify}(crs, vk, m, \sigma) = 1$ 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 $history$, and return $\mathcal{A}$ with message $(i, history)$;

  – 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 $history$, and return $\mathcal{A}$ with message $(i, history)$.

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

□

4 UC Blind Signatures Definition

We formulate our blind signature ideal functionality $\mathcal{F}_{\text{BSIG}}$ in Figure 11. A previous formalization of the blind signature primitive in the UC setting was given by [Fis06]. We would like to point out that our formalization is based on $\mathcal{F}_{\text{SIG}}$ as given in [Can05] while Fischlin's is based on $\mathcal{F}_{\text{SIG}}$ in [Can04]; there is a number of other (small) differences that we review in this section. 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. We explain in Remark 4.4 how it is possible to readily modify our $\mathcal{F}_{\text{BSIG}}$ to cover the strong unforgeability property.

As defined in Definition 2.6, 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 $\texttt{sign}$ 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 4.1. 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 $\mathcal{F}_{\text{BSIG}}$ of Figure 11 in the $\mathcal{F}_{\text{CRS}}$-hybrid world, where $\mathcal{F}_{\text{CRS}}$ is an implementation of the $\texttt{CRSgen}$ 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 2.2.6 and Remark 2.7. Recall that the definitions cover regular version of unforgeability and strong version of blindness.).

The theorem below is a sanity-check that shows that the ideal functionality we propose is consistent with some of the previous modelling 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 4.2. *If $\Sigma(\text{BSIG})$ can realize our $\mathcal{F}_{\text{BSIG}}$ in Figure 11, 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 $(\texttt{KeyGen}, sid)$, and gives the returned verification algorithm ver to F.

When the simulated F outputs $\texttt{user}_1$ message on behalf of some party U, $\mathcal{Z}$ creates party U; it corrupts party U and forces it to send an outgoing $\texttt{user}_1$ message (through $\mathcal{Z}$) to party S; when F outputs a $\texttt{user}_i$ message where $i > 1$, $\mathcal{Z}$ forces party U to send the $\texttt{user}_i$ message to party S; when the corrupted party U receives an incoming $\texttt{signer}_j$ message, $\mathcal{Z}$ forwards it to the simulated F. $\mathcal{Z}$ uses $counter_1$ to count the number of successful final $\texttt{signer}$ messages from the party S. $\mathcal{Z}$ uses $counter_2$ to count the number

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

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

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

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

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

Upon receiving $(\texttt{Signature}, sid, U, \mathsf{UserComplete})$ from the adversary $\mathcal{S}$, where U is a user that has requested a signature, do: if S is honest and $\mathsf{ver} \neq \mathsf{ver}'$, output $(\texttt{Signature}, sid, \perp)$ to party U and halt. Else, use the information in both $history(U)$ and $history(S)$ to obtain a valid signature: compute $\sigma \leftarrow \mathsf{sig}(m, rnd)$ with the required random coins rnd, and verify that $\mathsf{ver}'(m, \sigma) = 1$; if so, output $(\texttt{Signature}, sid, m, \sigma)$ to party U, and record $\langle \sigma, rnd, \mathsf{done} \rangle$ into $history(U)$ together with $\langle m, \mathsf{ver}' \rangle$ inside $history(U)$. Else, output $(\texttt{Signature}, sid, \mathsf{error})$ to party U and halt.

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

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

Signature verification: Upon receiving $(\texttt{Verify}, sid, m, \sigma, \mathsf{ver}')$ 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 output $(\texttt{Verified}, sid, \mathsf{error})$ to party V and halt. Else, output $(\texttt{Verified}, sid, m, \mathsf{ver}'(m, \sigma))$ to party V.

Corruption: Upon receiving $(\texttt{Corrupt}, sid, J)$ from $\mathcal{S}$, return $(\texttt{Corrupted}, sid, history(J))$ to $\mathcal{S}$. Here J can be party U or party S. Furthermore:

- after receiving $(\texttt{Corrupt}, sid, U)$, upon receiving $(\texttt{Patch}, sid, U, \overline{m})$ from the adversary $\mathcal{S}$, and no $\langle U, \mathsf{completed} \rangle$ was recorded in $history(S)$, then replace the old message $\langle m, \mathsf{ver}' \rangle$ in $history(U)$ with $\langle \overline{m}, \mathsf{ver}' \rangle$; once a subsequent $(\texttt{Signature}, sid, U, \mathsf{SignerComplete})$ is received from $\mathcal{S}$, record done in $history(U)$ and $\langle U, \mathsf{completed} \rangle$ in $history(S)$;
- after receiving $(\texttt{Corrupt}, sid, S)$, upon receiving $(\texttt{Patch}, sid, S, \overline{\mathsf{sig}})$ from the adversary $\mathcal{S}$, then replace sig in $history(S)$ with $\overline{\mathsf{sig}}$.

Figure 11: Blind signature functionality $\mathcal{F}_{\mathrm{BSIG}}$.

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

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

Key generation: When party S is invoked with input $(\mathtt{KeyGen}, sid)$ 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 $\mathsf{ver} = \mathtt{verify}(crs, vk, \cdot, \cdot)$, records $\langle sk, vk\rangle$ in $history(S)$, and sends output $(\mathtt{VerificationAlg}, sid, \mathsf{ver})$ to $\mathcal{Z}$.

Signature generation: When party U is invoked with input $(\mathtt{Sign}, sid, m, \mathsf{ver}')$ by $\mathcal{Z}$ where $sid = (S, sid')$ and $U \neq S$, it computes $\mathsf{UserMsg}_1$ where some random coins $rnd_{U,1}$ may be used, records $\langle m, \mathsf{ver}', rnd_{U,1}\rangle$ in $history(U)$, and sends outgoing $(\mathtt{user}_1, sid, \mathsf{UserMsg}_1)$ to party S (through $\mathcal{Z}$). If party U cannot compute $\mathsf{UserMsg}_1$, then sends output $(\mathtt{Signature}, sid, \perp)$ to $\mathcal{Z}$.

When party U is invoked with incoming $(\mathtt{signer}_i, sid, \mathsf{SignerMsg}_i)$ from party S, where $sid = (S, sid')$, it computes $\mathsf{UserMsg}_{i+1}$ where some random coins $rnd_{U,i+1}$ may be used, records $\langle rnd_{U,i+1}\rangle$ in $history(U)$, and sends outgoing $(\mathtt{user}_{i+1}, sid, \mathsf{UserMsg}_{i+1})$ to party S. If U cannot compute $\mathsf{UserMsg}_{i+1}$, then sends output $(\mathtt{Signature}, sid, \perp)$ to $\mathcal{Z}$.

When party S is invoked with incoming $(\mathtt{user}_i, sid, \mathsf{UserMsg}_i)$ from party U, where $sid = (S, sid')$, it computes $\mathsf{SignerMsg}_i$ where some random coins $rnd_{S,i}$ may be used, records $\langle rnd_{S,i}, U\rangle$ in $history(S)$, and sends outgoing $(\mathtt{signer}_i, sid, \mathsf{SignerMsg}_i)$ to party U. If S cannot compute $\mathsf{SignerMsg}_i$, then sends output $(\mathtt{Signature}, sid, U, \perp)$ to $\mathcal{Z}$.

When party S is invoked with incoming $(\mathtt{user}_{last}, sid, \mathsf{UserMsg}_{last})$, which is the last user message from party U, where $sid = (S, sid')$, it computes $\mathsf{SignerMsg}_{last}$ where some random coins $rnd_{S,last}$ may be used, records $\langle rnd_{U,last}, U\rangle$ in $history(S)$, and sends outgoing $(\mathtt{signer}_{last}, sid, \mathsf{SignerMsg}_{last})$ to party U, and sends output $(\mathtt{Signature}, sid, U, \mathsf{completed})$ to $\mathcal{Z}$. If party S cannot compute $\mathsf{SignerMsg}_{last}$, then sends output $(\mathtt{Signature}, sid, U, \perp)$ to $\mathcal{Z}$.

When party U is invoked with incoming $(\mathtt{signer}_{last}, sid, \mathsf{SignerMsg}_{last})$, which is the last signer message from party S, where $sid = (S, sid')$, it computes signature σ for m where some random coins $rnd_{U,last+1}$ may be used, records $\langle \sigma, rnd_{U,last+1}\rangle$ in $history(U)$. It verifies that $\mathsf{ver}'(m, \sigma) = 1$; if so, sends output $(\mathtt{Signature}, sid, m, \sigma)$ to $\mathcal{Z}$; if not, sends output $(\mathtt{Signature}, sid, \perp)$ to $\mathcal{Z}$.

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

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

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

of $(\texttt{Signature}, \mathit{sid}, U, \textsf{completed})$ 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 $(\texttt{Verify}, \mathit{sid}, \widehat{m}_i, \widehat{\sigma}_i, \textsf{ver})$, where $i = 1, \ldots, L$; the verifiers will return 1 for the successful forged pairs. $\mathcal{Z}$ uses $\mathit{counter}_3$ to count the number of 1's. If $\mathcal{Z}$ finds that $\mathit{counter}_3 \leq \mathit{counter}_2$ it returns 0 otherwise it returns 1.

In the real world, because F is a successful forger, with non-negligible probability, $\mathcal{Z}$ will observe $\mathit{counter}_3 > \mathit{counter}_1$. Moreover, note that $\mathit{counter}_1 = \mathit{counter}_2$ will hold in the real world, which is based on the fact: when party S sends his last outgoing $\texttt{signer}$ message to party U, he also sends output $(\texttt{Signature}, \mathit{sid}, U, \textsf{completed})$ 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 $(\texttt{Verify}, \mathit{sid}, \widehat{m}_i, \widehat{\sigma}_i, \textsf{ver})$, he will forward such message to $\mathcal{F}_{\mathrm{BSIG}}$, and $\mathcal{F}_{\mathrm{BSIG}}$ will check if the message is a forgery using the information he possesses. Based on the definition of $\mathcal{F}_{\mathrm{BSIG}}$, such check will return $\textsf{ver}(\widehat{m}_i, \widehat{\sigma}_i)$ to the verifier, as long $\widehat{m}_i$ is recorded with done. In case the message is not recorded with done the two possible answers from $\mathcal{F}_{\mathrm{BSIG}}$ would be $(\texttt{Verified}, \mathit{sid}, \widehat{m}_i, \widehat{\sigma}_i, 0)$ and $(\texttt{Verified}, \mathit{sid}, \widehat{m}_i, \widehat{\sigma}_i, \textsf{error})$. It follows that $\mathit{counter}_1$ 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 $\mathcal{F}_{\mathrm{BSIG}}$: $\mathcal{S}$ has patched the particular message into some corrupted user instance inside $\mathcal{F}_{\mathrm{BSIG}}$. Note that while unlimited patching is allowed (and observe that $\mathcal{S}$ may be capable of obtaining unlimited numbers of forged signatures) a message is recorded with done by $\mathcal{F}_{\mathrm{BSIG}}$ only after a message $(\texttt{Signature}, \mathit{sid}, U, \textsf{SignerComplete})$ is received by $\mathcal{S}$. The environment tracks the $(\texttt{Signature}, \mathit{sid}, U, \textsf{SignerComplete})$ messages into $\mathit{counter}_2$ thus we can be certain that in the ideal world it would be impossible to have $\mathit{counter}_3 > \mathit{counter}_2$. It follows that in the ideal world, $\mathit{counter}_3 \leq \mathit{counter}_2$; 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 $\Sigma(\mathrm{BSIG})$. 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 $(\texttt{Sign}, \mathit{sid}, m_b, \textsf{ver})$ and $(\texttt{Sign}, \mathit{sid}, m_{1-b}, \textsf{ver})$, respectively, where $b \xleftarrow{\mathrm{r}} \{0,1\}$ 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 $(\texttt{Signature}, \mathit{sid}, m_b, \sigma_b)$, $(\texttt{Signature}, \mathit{sid}, m_{1-b}, \sigma_{1-b})$ from the two parties U_L and U_R respectively, $\mathcal{Z}$ returns the two message-signature pairs $\langle m_0, \sigma_0; m_1, \sigma_1\rangle$ to D. Otherwise $\mathcal{Z}$ returns $\langle \perp; \perp\rangle$ to D. 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 $\Sigma(\mathrm{BSIG})$; 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 $\mathcal{F}_{\mathrm{BSIG}}$ 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 $\langle \perp; \perp\rangle$ to the distinguisher; it follows that the coins b are independent from D and $\mathcal{S}$, 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. □

Remark 4.3. We can easily modify the functionality $\mathcal{F}_{\mathrm{BSIG}}$ in Figure 11 to incorporate Canetti's $\mathcal{F}_{\mathrm{SIG}}$ in [Can05] as a special case (refer to Figure 1 in page 7) by adding the following subitem into "Signature generation" item in $\mathcal{F}_{\mathrm{BSIG}}$.

Upon receiving $(\texttt{Sign}, \mathit{sid}, m)$ from party S where $\mathit{sid} = (S, \mathit{sid}')$, let $\sigma = \textsf{sig}(m, \mathit{rnd})$ where some random coins rnd may be used, and verify that $\textsf{ver}(m, \sigma) = 1$. If so, output $(\texttt{Signature}, \mathit{sid}, m, \sigma)$

to party S, and record $\langle m, \sigma, rnd, \mathsf{done}\rangle$ into $history(S)$. Else, output $(\mathtt{Signature}, sid, \bot)$ to party S and halt.

Remark 4.4. The ideal functionality $\mathcal{F}_{\text{BSIG}}$ in Figure 11 does not cover strong unforgeability. It is simple to modify it so that the property can be achieved.

We have to incorporate the following modifications in the program of the ideal functionality in Figure 11. First, we need stricter check in the signature verification stage, i.e. (m, σ) is not recorded with done:

> Upon receiving $(\mathtt{Verify}, sid, m, \sigma, \mathsf{ver}')$ 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, σ) is not recorded with done, then output $(\mathtt{Verified}, sid, \mathsf{error})$ to party V and halt. Else, output $(\mathtt{Verified}, sid, m, \mathsf{ver}'(m, \sigma))$ to party V.

Second, when the user is corrupted, the functionality expects that the ideal adversary will patch $(\overline{m}, \overline{\sigma})$, not only $\overline{m}$ as before.

- after receiving $(\mathtt{Corrupt}, sid, U)$, upon receiving $(\mathtt{Patch}, sid, U, \overline{m}, \overline{\sigma})$ from the adversary $\mathcal{S}$, and no $\langle U, \mathsf{completed}\rangle$ was recorded in $history(S)$, then replace the old message $\langle m, \mathsf{ver}'\rangle$ in $history(U)$ with $\langle \overline{m}, \overline{\sigma}, \mathsf{ver}'\rangle$; once a subsequent $(\mathtt{Signature}, sid, U, \mathsf{SignerComplete})$ is received from the $\mathcal{S}$, record done in $history(U)$ and $\langle U, \mathsf{completed}\rangle$ in $history(S)$;

5 Design Methodology for Adaptively Secure UC Blind Signatures

In this section we will 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 $\mathcal{F}_{\text{BSIG}}$ (refer to Figure 11). Our design reveals the exact components required for designing UC blind signatures in the adaptive security setting. In our construction we will employ a lite blind signature and we will operate in a hybrid world where the following ideal functionalities exist: $\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}$; $\mathcal{F}_{\text{CRS}}$ 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: (1) $\mathcal{F}_{\text{SVZK}}^{R_U}$ is the "single verifier zero-knowledge functionality for the relation R_U" defined in Figure 13 and, (2) $\mathcal{F}_{\text{SPZK}}^{R_S}$ is the "single-prover zero-knowledge functionality for the relation R_S" defined in Figure 14.

These two functionalities differ from the multi-session zero-knowledge ideal functionality $\mathcal{F}_{\text{MZK}}$ (i.e. $\hat{\mathcal{F}}_{\text{ZK}}$ 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. Moreover, recall that in a single blind signature session we have a single signer interacting with many users.

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

We proceed next to describe our generic construction. In Figure 15, 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 $R_U = \{((crs, vk, \mathbf{u}), (m, \rho_1)) \mid \mathbf{u} =$

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

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

Proof stage: Upon receiving $(\texttt{ProveSVZK}, sid, P_i, x, w)$ 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 $(\texttt{ProveSVZK}, sid, P_i, x)$ to the adversary $\mathcal{S}$.

Upon receiving $(\texttt{ProveSVZK}, sid, P_i, \mathsf{VerifierComplete})$ from the adversary $\mathcal{S}$, output $(\texttt{VerifiedSVZK}, sid, P_i, x)$ to party V. Else, if receiving $(\texttt{ProveSVZK}, sid, P_i, \mathsf{VerifierError})$ from the adversary $\mathcal{S}$, output $(\texttt{VerifiedSVZK}, sid, P_i, \perp)$ to party V and halt.

Corruption: Upon receiving $(\texttt{CorruptProverSVZK}, sid, P_i)$ from the adversary $\mathcal{S}$, return $\mathcal{S}$ $(\texttt{CorruptedProverSVZK}, sid, history(P_i))$.

After the successful corruption of P_i, upon receiving $(\texttt{PatchSVZK}, sid, P_i, x', w')$ from the adversary $\mathcal{S}$, if $(x', w') \in R$ and no output $(\texttt{VerifiedSVZK}, sid, P_i, ...)$ was returned to party V yet, then output $(\texttt{VerifiedSVZK}, sid, P_i, x')$ to party V. Else, if $(x', w') \notin R$ and no output $(\texttt{VerifiedSVZK}, sid, P_i, ...)$ was returned to party V yet, then output $(\texttt{VerifiedSVZK}, sid, P_i, \mathsf{error})$ to party V and halt.

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

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

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

Proof stage: Upon receiving $(\texttt{ProveSPZK}, sid, V_i, x, w)$ 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 $(\texttt{ProveSPZK}, sid, V_i, x)$ to the adversary $\mathcal{S}$.

Upon receiving $(\texttt{ProveSPZK}, sid, V_i, \mathsf{VerifierComplete})$ from the adversary $\mathcal{S}$, output $(\texttt{VerifiedSPZK}, sid, V_i, x)$ to party V_i. Else, if receiving $(\texttt{ProveSPZK}, sid, V_i, \mathsf{VerifierError})$ from the adversary $\mathcal{S}$, output $(\texttt{VerifiedSPZK}, sid, V_i, \perp)$ to party V_i and halt.

Corruption: Upon receiving $(\texttt{CorruptProverSPZK}, sid)$ from the adversary $\mathcal{S}$, return $\mathcal{S}$ $(\texttt{CorruptedProverSPZK}, sid, history(P))$.

After the corruption has occurred successfully, upon receiving $(\texttt{PatchSPZK}, sid, V_i, x', w')$ from the adversary $\mathcal{S}$, if $(x', w') \in R$ and no output $(\texttt{VerifiedSPZK}, sid, V_i, ...)$ was returned to party V_i yet, then output $(\texttt{VerifiedSPZK}, sid, V_i, x')$ to party V_i. Else, if $(x', w') \notin R$ and no output $(\texttt{VerifiedSPZK}, sid, V_i, ...)$ was returned to party V_i yet, then output $(\texttt{VerifiedSPZK}, sid, V_i, \mathsf{error})$ to party V_i and halt.

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

$\mathsf{lbs}_1(crs, vk, m; \rho_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2)) \mid \mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \wedge (vk, sk) \in \textsc{KeyPair}\}$. We remark that the protocol in Figure 15 is non-interactive; due to the usage of the ideal functionalities $\mathcal{F}_{\text{SVZK}}^{R_U}$ and $\mathcal{F}_{\text{SPZK}}^{R_S}$ all the communication the users and the signer is relayed through the ideal functionalities.

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

Key generation: When party S is invoked with input $(\texttt{KeyGen}, sid)$ 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 $\mathsf{ver} = \texttt{verify}(crs, vk, \cdot, \cdot)$, and sends output $(\texttt{VerificationAlg}, sid, \mathsf{ver})$ to $\mathcal{Z}$.

Signature generation: On input $(\texttt{Sign}, sid, m, \mathsf{ver}')$ 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 $(\texttt{ProveSVZK}, sid, U, (crs, vk', \mathbf{u}), (m, \rho_1))$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$ where $\mathcal{F}_{\text{SVZK}}^{R_U}$ is parameterized by the relation R_U.

Upon receiving $(\texttt{VerifiedSVZK}, sid, U, (crs, vk', \mathbf{u}))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, the party S verifies $vk' = vk$. If not, then the party S outputs $(\texttt{Signature}, sid, U, \perp)$ 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 $(\texttt{ProveSPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}), (sk, \rho_2))$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$ where $\mathcal{F}_{\text{SPZK}}^{R_S}$ is parameterized by the relation R_S; and outputs $(\texttt{Signature}, sid, U, \mathsf{completed})$ to $\mathcal{Z}$.

Upon receiving $(\texttt{VerifiedSVZK}, sid, U, \perp)$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, the party S outputs $(\texttt{Signature}, sid, U, \perp)$ to $\mathcal{Z}$.

Upon receiving $(\texttt{VerifiedSPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}))$ from $\mathcal{F}_{\text{SPZK}}^{R_S}$, 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 verifies $\mathsf{ver}'(m, \sigma) = 1$; if so, outputs $(\texttt{Signature}, sid, m, \sigma)$ to $\mathcal{Z}$; if not, outputs $(\texttt{Signature}, sid, \perp)$ to $\mathcal{Z}$.

Upon receiving $(\texttt{VerifiedSPZK}, sid, U, \perp)$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, the party U outputs $(\texttt{Signature}, sid, \perp)$ to $\mathcal{Z}$.

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

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

Figure 15: 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 $\langle \texttt{CRSgen}, \texttt{gen}, \mathsf{lbs}_1, \mathsf{lbs}_2, \mathsf{lbs}_3, \texttt{verify} \rangle$.

Next we will prove a theorem that the blind signature protocol in Figure 15 can realize functionality $\mathcal{F}_{\text{BSIG}}$. Before we prove the theorem, 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 5.1. *Assume that*

(1) $\exists \mathcal{S}_1, \forall \mathcal{Z}$, $\text{EXEC}_{\pi_{\Sigma(\text{BSIG})}, \mathcal{Z}}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{SVZK}}^{R_U}, \mathcal{F}_{\text{SPZK}}^{R_S}} = \text{EXEC}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}^{\mathcal{F}_1}$, *where* $\mathcal{F}_1$ *is dummy blind signature functionality (cf. Figure 16);*

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

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

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

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

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

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

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

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

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

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

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

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

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

Corruption: Upon receiving $(\texttt{Corrupt}, sid, J)$ from $\mathcal{S}$, return $(\texttt{Corrupted}, sid, history(J))$ to $\mathcal{S}$. Here J can be party U or party S.

Figure 16: Dummy blind signature functionality $\mathcal{F}_1$ in Lemma 5.1.

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

Theorem 5.2. *Given a signature generation protocol that is an equivocal lite blind signature, the protocol $\pi_{\Sigma(\mathrm{BSIG})}$ in Figure 15 securely realizes $\mathcal{F}_{\mathrm{BSIG}}$ in the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}^{R_U}_{\mathrm{SVZK}}, \mathcal{F}^{R_S}_{\mathrm{SPZK}})$-hybrid model with advantage $\mathsf{Adv}^{\mathrm{lbs}}_{\mathrm{eq}} + \mathsf{Adv}^{\mathrm{lbs}}_{\mathrm{luf}}$, where $\mathsf{Adv}^{\mathrm{lbs}}_{\mathrm{eq}}$ and $\mathsf{Adv}^{\mathrm{lbs}}_{\mathrm{luf}}$ are the equivocality distance and lite-unforgeability distance for the underlying lite blind signature.*

Proof. In order to prove that $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}^{R_U}_{\mathrm{SVZK}}, \mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BSIG})}, \mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}_{\mathrm{BSIG}}}_{\pi_d, \mathcal{S}, \mathcal{Z}}$, we use the proof strategy explored in Lemma 5.1. We develop several bridge hybrid worlds between the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}^{R_U}_{\mathrm{SVZK}}, \mathcal{F}^{R_S}_{\mathrm{SPZK}})$-hybrid world and the ideal world, and define the ensemble of random variables of $\mathcal{Z}$'s output of each bridge hybrid worlds as $\mathrm{EXEC}^{\mathcal{F}_i}_{\pi_d, \mathcal{S}_i, \mathcal{Z}}$, $i = 1, 2, \ldots, 5$, where π_d is the dummy protocol same as that in the ideal world. Next we prove $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}^{R_U}_{\mathrm{SVZK}}, \mathcal{F}^{R_S}_{\mathrm{SPZK}}}_{\pi_{\Sigma(\mathrm{BSIG})}, \mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d, \mathcal{S}_1, \mathcal{Z}} \approx \cdots \approx \mathrm{EXEC}^{\mathcal{F}_5}_{\pi_d, \mathcal{S}_5, \mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}_{\mathrm{BSIG}}}_{\pi_d, \mathcal{S}, \mathcal{Z}}$. 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 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.

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 16.

The simulator $\mathcal{S}_1$ simulates exactly the protocol $\pi_{\Sigma(\text{BSIG})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}^{R_U}_{\text{SVZK}}, \mathcal{F}^{R_S}_{\text{SPZK}})$-hybrid model except that all inputs/outputs of the parties of protocol $\pi_{\Sigma(\text{BSIG})}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}^{R_U}_{\text{SVZK}}, \mathcal{F}^{R_S}_{\text{SPZK}})$-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 $(\mathcal{F}_{\text{CRS}}, \mathcal{F}^{R_U}_{\text{SVZK}}, \mathcal{F}^{R_S}_{\text{SPZK}})$-hybrid world. So we have $\text{EXEC}^{\mathcal{F}_{\text{CRS}}, \mathcal{F}^{R_U}_{\text{SVZK}}, \mathcal{F}^{R_S}_{\text{SPZK}}}_{\pi_{\Sigma(\text{BSIG})},\mathcal{Z}} = \text{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}}$.

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 $\mathit{history}(S)$, the vault $\mathcal{F}_2$ will store the patched m into $\mathit{history}(U)$ (note that if there is old $\langle m, \mathsf{ver}'\rangle$ in $\mathit{history}(U)$, then replace the old m with this patched m); now if receives message $(\texttt{Signature}, \mathit{sid}, U, \mathsf{completed})$ from $\mathcal{S}_2$, then $\mathcal{F}_2$ stores a mark done into $\mathit{history}(U)$.

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

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 $(\texttt{Verify}, \mathit{sid}, m, \sigma, \mathsf{ver}')$ 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 $\mathcal{F}_3$ outputs $(\texttt{Verified}, \mathit{sid}, \mathsf{error})$ to the dummy party V and halts, otherwise $(\texttt{Verified}, \mathit{sid}, m, \mathsf{ver}'(m, \sigma))$ 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 $(\texttt{Verify}, \mathit{sid}, m, \sigma, \mathsf{ver})$, 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 $\text{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} = \text{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Based on the difference lemma (refer to [Sho04]), $|\text{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}} - \text{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}| \leq \Pr[\mathbf{E}]$. 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 $(\texttt{KeyGen}, sid)$ with $sid = (S, sid')$ for some sid', $\mathcal{B}$ returns ver to $\mathcal{Z}$, where $\mathsf{ver} \overset{\text{def}}{=} \texttt{verify}(crs, vk, \cdot, \cdot)$. Note that vk is from the input of $\mathcal{B}$ as described before.
2. When the simulated S receives $(\texttt{VerifiedSVZK}, sid, U, (crs, vk', \mathbf{u}))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, in the case that $vk' = vk$, $\mathcal{S}_2$ needs to simulate $\mathcal{F}_{\text{SPZK}}^{R_S}$ to send $(\texttt{ProveSPZK}, sid, U, (crs, vk, \mathbf{u}, \mathbf{s}))$ 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 $\mathcal{F}_{\text{SVZK}}^{R_U}$ 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 $\mathcal{F}_{\text{SPZK}}^{R_S}$ send $(\texttt{ProveSPZK}, sid, U, (crs, vk, \mathbf{u}, \mathbf{s}))$ 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 $(\texttt{Verify}, sid, m, \sigma, \mathsf{ver}')$, $\mathcal{B}$ 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 $\Pr[\mathbf{E}] = \mathsf{Adv}_{\text{luf}}^{\text{lbs}}$. So, we have $\left| \text{EXEC}_{\pi_d, \mathcal{S}_2, \mathcal{Z}}^{\mathcal{F}_2} - \text{EXEC}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}^{\mathcal{F}_3} \right| \le \mathsf{Adv}_{\text{luf}}^{\text{lbs}}$.

$\text{EXEC}_{\pi_d, \mathcal{S}_4, \mathcal{Z}}^{\mathcal{F}_4}$. 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 $(\texttt{Algorithms}, sid, \mathsf{sig}, \mathsf{ver})$ 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 $\mathcal{F}_{\text{SPZK}}^{R_S}$ on behalf of the corrupted signer S, $\mathcal{S}_4$ defines $\mathsf{sig} \overset{\text{def}}{=} \texttt{sign}(crs, vk, sk, \cdot, \cdot)$ 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 $\text{EXEC}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}^{\mathcal{F}_3} = \text{EXEC}_{\pi_d, \mathcal{S}_4, \mathcal{Z}}^{\mathcal{F}_4}$.

$\text{EXEC}_{\pi_d, \mathcal{S}_5, \mathcal{Z}}^{\mathcal{F}_5}$. When the vault $\mathcal{F}_5$ receives $(\texttt{Sign}, sid, m, \mathsf{ver}')$ from each dummy user U, $\mathcal{F}_5$ "blocks" m and sends $(\texttt{Sign}, sid, U, \mathsf{ver}')$ 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 3.9. 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 $(\texttt{Signature}, sid, U, \mathsf{ver}')$ 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 $\mathcal{F}_{\text{SVZK}}^{R_U}$ to send $(\texttt{ProveSVZK}, sid, U, (crs, vk', \mathbf{u}))$ to $\mathcal{Z}$.

If $\mathcal{Z}$ corrupts U after $\mathcal{Z}$ receives $(\texttt{ProveSVZK}, sid, U, (crs, vk', \mathbf{u}))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, $\mathcal{S}_5$ 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 $(\texttt{Signature}, sid, m, \sigma)$, $\mathcal{S}_5$ 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 $(\texttt{Sign}, sid, m, \mathsf{ver}')$ to a dummy user U, now $\mathcal{E}$ queries $\mathcal{O}$ with (U, m, vk) and receives $(U, \mathbf{u})$; $\mathcal{S}_4$ simulates $\mathcal{F}_{\text{SVZK}}^{R_U}$ to send $(\texttt{ProveSVZK}, sid, (crs, vk, \mathbf{u}))$ to $\mathcal{Z}$;
- When $\mathcal{Z}$ returns $\mathsf{VerifierComplete}$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$ 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 $\mathcal{F}_{\text{SPZK}}^{R_S}$ to send $(\texttt{ProveSPZK}, sid,\ (crs, vk, \mathbf{s}))$ to $\mathcal{Z}$; if $\mathcal{Z}$ returns $\mathsf{VerifierComplete}$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, then $\mathcal{E}$ queries $\mathcal{O}$ with $(U, \mathbf{s}, \rho_2, sk)$ and receives (U, σ); $\mathcal{E}$ returns $(\texttt{Signature}, sid, m, \sigma)$ to $\mathcal{Z}$ on behalf of the dummy user.

  When the signer is corrupted, $\mathcal{Z}$ may send $(\texttt{ProveSPZK}, sid,\ (crs, vk, \mathbf{s}), (sk, \rho_2))$ 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 $(\texttt{Corrupt}, sid, U)$ 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 $\left|\text{EXEC}_{\pi_d,\mathcal{S}_4,\mathcal{Z}}^{\mathcal{F}_4} - \text{EXEC}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}^{\mathcal{F}_5}\right| \le \mathsf{Adv}_{\text{eq}}^{\text{lbs}}$.

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 $\text{EXEC}_{\pi_d,\mathcal{S},\mathcal{Z}}^{\mathcal{F}_{\text{BSIG}}} = \text{EXEC}_{\pi_d,\mathcal{S}_5,\mathcal{Z}}^{\mathcal{F}_5}$.

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

□

The construction of the ideal world 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: $(crs, \tau) \leftarrow \texttt{CRSgen}(1^\lambda)$. Define two relations $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$ and $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (W v_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$.

Simulation of key generation:

(1) In the ideal world, when the dummy signer S receives an input $(\texttt{KeyGen}, sid)$ from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}_{\text{BSIG}}$. The functionality $\mathcal{F}_{\text{BSIG}}$ then forwards $(\texttt{KeyGen}, sid)$ 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 $(vk, sk) \leftarrow \texttt{gen}(crs)$, defines the signing algorithm $\mathsf{sig} \overset{\text{def}}{=} \texttt{sign}(crs, vk, sk, \cdot, \cdot)$ and the verification algorithm $\mathsf{ver} \overset{\text{def}}{=} \texttt{verify}(crs, vk, \cdot, \cdot)$; then $\mathcal{S}$ sends message $(\texttt{Algorithms}, sid, \mathsf{sig}, \mathsf{ver})$ 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 $(\texttt{VerificationAlg}, sid, \mathsf{ver})$ to the dummy signer S, and then to the environment $\mathcal{Z}$.
(1$^+$) Now if $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, S)$ 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. Note that in the future when the corrupted S (controlled by $\mathcal{Z}$) wants to respond to the user's request $(\texttt{VerifiedSVZK}, sid, U, (crs, vk, \mathbf{u}))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, it needs to send $(\texttt{ProveSPZK}, sid, U, (crs, vk, \mathbf{u}, \overline{\mathrm{s}}), (\overline{sk}, \overline{\rho}_2))$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, where $\overline{\mathrm{s}} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, \overline{sk}; \overline{\rho}_2)$. When $\mathcal{F}_{\text{SPZK}}^{R_S}$ returns $(\texttt{VerifiedSPZK}, sid, U, (crs, vk, \mathbf{u}, \overline{\mathrm{s}}))$ to the user U, i.e. $((crs, vk, \mathbf{u}, \overline{\mathrm{s}}), (\overline{sk}, \overline{\rho}_2)) \in R_S$, the simulator $\mathcal{S}$ can obtain $(\overline{sk}, \overline{\rho}_2)$ and define $\overline{\mathsf{sig}} \overset{\text{def}}{=} \texttt{sign}(crs, vk, \overline{sk}, \cdot, \cdot)$ and patch $\overline{\mathsf{sig}}$ into $\mathcal{F}_{\text{BSIG}}$. 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 3.9 because the underlying lite blind signature is equivocal. We give details below.

(2) When the dummy user U receives an input $(\texttt{Sign}, sid, m, \mathsf{ver}')$ 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 $(\texttt{Sign}, sid, \mathsf{ver}', U)$ 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 $(\texttt{ProveSVZK}, sid, U, (crs, vk', \mathbf{u}))$ to $\mathcal{Z}$.
(2$^+$) Now if the simulator $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, U)$ from $\mathcal{Z}$ (i.e. after $\mathcal{Z}$ received the $\texttt{ProveSVZK}$ message). The simulator $\mathcal{S}$ reconstructs the simulated user U's internal state $\langle m, \rho_1\rangle$ as follows: $\mathcal{S}$ sends the $\texttt{Corrupt}$ 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 $(\texttt{ProveSVZK}, sid, (crs, vk', \overline{\mathbf{u}}), (\overline{m}, \overline{\rho}_1))$ 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 $(\texttt{VerifiedSVZK}, sid, U, (crs, vk', \overline{\mathbf{u}}))$ 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 $(\texttt{ProveSVZK}, sid, U, \mathsf{VerifierError})$ to $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$, $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$ sends $(\texttt{ProveSVZK}, sid, U, \perp)$ to the simulated S, and $\mathcal{S}$ now sends $(\texttt{Signature}, sid, U, \mathsf{SignerError})$ to $\mathcal{F}_{\mathrm{BSIG}}$; when $\mathcal{F}_{\mathrm{BSIG}}$ receives this message, it will output $(\texttt{Signature}, sid, U, \perp)$ to the dummy party S, then to $\mathcal{Z}$. Else when $\mathcal{Z}$ returns $(\texttt{ProveSVZK}, sid, U, \mathsf{VerifierComplete})$ to $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$, $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$ sends $(\texttt{ProveSVZK}, sid, U, (crs, vk', \mathbf{u}))$ 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 $(\texttt{Signature}, sid, U, \mathsf{SignerComplete})$ to $\mathcal{F}_{\mathrm{BSIG}}$; when $\mathcal{F}_{\mathrm{BSIG}}$ receives this message, it records $\langle U, \mathsf{completed}\rangle$ in $history(S)$ and outputs $(\texttt{Signature}, sid, U, \mathsf{completed})$ to the dummy party S, then to $\mathcal{Z}$. At the same time S sends $(\texttt{ProveSPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}), (sk, \rho_2))$ to $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$ and $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$ sends $(\texttt{ProveSPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}))$ to $\mathcal{Z}$.

(3$^+$) Now if the simulator $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, S)$ from $\mathcal{Z}$ (i.e. $\mathcal{Z}$ received the $\texttt{ProveSPZK}$ 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 $(\texttt{ProveSPZK}, sid, U, \mathsf{VerifierError})$ to $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$, $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$ sends $(\texttt{ProveSPZK}, sid, U, \perp)$ to the simulated U, $\mathcal{S}$ will send $(\texttt{Signature}, sid, U, \mathsf{UserError})$ to $\mathcal{F}_{\mathrm{BSIG}}$; when $\mathcal{F}_{\mathrm{BSIG}}$ receives this message, it will output $(\texttt{Signature}, sid, \perp)$ to the dummy party U, then to $\mathcal{Z}$. Else when $\mathcal{Z}$ returns $(\texttt{ProveSPZK}, sid, U, \mathsf{VerifierComplete})$ to $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$, $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$ sends $(\texttt{ProveSPZK}, sid, U, (crs, vk', \mathbf{u}, \mathbf{s}))$ 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 $(\texttt{Signature}, sid, U, \mathsf{UserComplete})$ to $\mathcal{F}_{\mathrm{BSIG}}$; now $\mathcal{F}_{\mathrm{BSIG}}$ 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 $(\texttt{Signature}, sid, m, \sigma)$ to the dummy user U, then to $\mathcal{Z}$. If $\mathsf{ver}'(crs, \widetilde{m}, \widetilde{\sigma}) \neq 1$, then $\mathcal{S}$ now sends $(\texttt{Signature}, sid, U, \mathsf{UserError})$ to $\mathcal{F}_{\mathrm{BSIG}}$, and now $\mathcal{F}_{\mathrm{BSIG}}$ sends $(\texttt{Signature}, sid, \perp)$ to the dummy U, then to $\mathcal{Z}$.

(4$^+$) Now if the simulator $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, U)$ 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 $\texttt{Corrupt}$ message to $\mathcal{F}_{\mathrm{BSIG}}$ 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 $(\texttt{KeyGen}, sid)$ 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 $(\texttt{Sign}, sid, m, \mathsf{ver}')$ 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 $(\texttt{Verify}, sid, m, \sigma, \mathsf{ver}')$ from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}_{\mathrm{BSIG}}$, and $\mathcal{F}_{\mathrm{BSIG}}$ will check if the input consists of forgery. If $\mathsf{ver}' = \mathsf{ver}$ where ver is from $history(U)$, the signer S is not corrupted, $\mathsf{ver}'(m, \sigma) = 1$, and m is not marked with a done, then $\mathcal{F}_{\mathrm{BSIG}}$ outputs $(\texttt{Signature}, sid, \perp)$ to party U and halts; Else, it outputs $(\texttt{Signature}, sid, \mathsf{ver}'(m, \sigma))$ to party U.

5.2 Implementation Strategies for $\mathcal{F}_{\mathrm{SVZK}}$ and $\mathcal{F}_{\mathrm{SPZK}}$

In this section, we discuss the special circumstances that apply in realizing our ZK functionalities.

Realizing $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$. The functionality $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$ will be realized against adaptive adversaries; we will proceed as follows: first given $(x, w) \in R_U$, we will have the prover commit the witness w into C, and then we design 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. The commitment scheme that we will employ is tuned to our "single-verifier" setting and is based on the mixed commitment primitive of [DN02, Nie03].

Please refer to an SVZK protocol in Figure 18 where $\texttt{EQC}$ and $\texttt{COM}$ are two equivocal commitment schemes, and $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate}, \texttt{reconstruct} \rangle$ is a non-erasure Sigma protocol for relation R'_U defined as follows: $R'_U = \{((crs, x, C_0, E), (w, \eta, \zeta)) \mid (x, w) \in R_U \wedge E = K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2 \wedge C_0 = \texttt{EQCcom}(pk_{\text{eqc}}, w; \eta)\}$. The combination of the commitment with the above Sigma protocol is shown in Figure 17.

$$crs = \langle \mathsf{n}, \mathsf{g}; \mathsf{K}; pk_{\text{eqc}} \rangle$$

$$\begin{array}{lcl}
\boxed{P} & & \boxed{V} \\
statement = \langle x \rangle & & statement = \langle x \rangle \\
witness = \langle w \rangle & & \\
\hline
\eta \overset{\text{r}}{\leftarrow} \texttt{RND}; & & \\
C_0 \leftarrow \texttt{EQCcom}(pk_{\text{eqc}}, w; \eta) & & \\
K_1 \overset{\text{r}}{\leftarrow} \mathbb{Z}^*_{\mathsf{n}^2}; \mu_1 \overset{\text{r}}{\leftarrow} \mathbb{Z}^*_{\mathsf{n}}; & & \\
C_1 \leftarrow \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3 & \xrightarrow{C_0, C_1} & \\
 & & K_2 \overset{\text{r}}{\leftarrow} \mathbb{Z}^*_{\mathsf{n}^2} \\
K \leftarrow K_1 K_2 \bmod \mathsf{n}^2 & \xleftarrow{K_2} & \\
\zeta \overset{\text{r}}{\leftarrow} \mathbb{Z}^*_{\mathsf{n}}; & & \\
E \leftarrow K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2 & & \\
r_a \overset{\text{r}}{\leftarrow} \texttt{RND} & & \\
a \leftarrow \texttt{prove}_1((x, C_0, E), (w, \eta, \zeta); r_a) & & \\
 & \xrightarrow{K_1, \mu_1, E, a} & C_1 \overset{?}{=} \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3 \\
 & & e \overset{\text{r}}{\leftarrow} \{0,1\}^{\ell_e} \\
 & \xleftarrow{e} & \\
z \leftarrow \texttt{prove}_3((x, C_0, E), (w, \eta, \zeta), r_a, e) & & \\
 & \xrightarrow{z} & \\
 & & \texttt{verify}((x, C_0, E), a, e, z) \overset{?}{=} 1 \\
\hline
\end{array}$$

Figure 17: A combination of a committing step with a non-erasure Sigma protocol for relation $R'_U = \{((crs, x, C_0, E), (w, \eta, \zeta)) \mid (x, w) \in R_U \wedge E = K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2 \wedge C_0 = \texttt{EQCcom}(pk_{\text{eqc}}, w; \eta)\}$ in the single-verifier setting. Here $\texttt{EQCcom}$ is the committing algorithm for equivocal commitment scheme $\texttt{EQC}$.

Theorem 5.3. *Given two equivocal commitment schemes $\texttt{EQC}$ and $\texttt{COM}$, and a non-erasure Sigma protocol $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate}, \texttt{reconstruct} \rangle$, the SVZK protocol $\pi_{\Sigma(\text{SVZK})}$ in Figure 18 securely realizes $\mathcal{F}^{R_U}_{\text{SVZK}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model with advantage $q_3 \cdot (\mathsf{Adv}^{\text{eqc}}_{\text{eq}} + \mathsf{Adv}^{\text{com}}_{\text{eq}} + \mathsf{Adv}^{\text{sigma}}_{\text{nezk}} + 2 \cdot \mathsf{Adv}_{\text{dcr}}) + q_1 \cdot (\mathsf{Adv}^{\text{eqc}}_{\text{binding}} + \nu(\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}}) + 2^{-\ell_{\mathsf{n}}})$, where q_3 is the number of the provers which are not corrupted initially, q_1 is the number of the provers corrupted initially, $\mathsf{Adv}^{\text{eqc}}_{\text{eq}}$ and $\mathsf{Adv}^{\text{eqc}}_{\text{binding}}$ are equivocality distance and binding distance for the equivocal commitment $\texttt{EQC}$, $\mathsf{Adv}^{\text{com}}_{\text{eq}}$ and $\mathsf{Adv}^{\text{com}}_{\text{binding}}$ are equivocality distance and binding distance for the equivocal commitment $\texttt{COM}$, $\mathsf{Adv}^{\text{sigma}}_{\text{nezk}}$ and $\mathsf{Adv}^{\text{sigma}}_{\text{sound}}$ are non-erasure honest-verifier zero-knowledge distance and soundness distance for the non-erasure Sigma protocol.*

Proof. In order to prove that $\text{EXEC}^{\mathcal{F}_{\text{CRS}}}_{\pi_{\Sigma(\text{SVZK})}, \mathcal{Z}} \approx \text{EXEC}^{\mathcal{F}^{R_U}_{\text{SVZK}}}_{\pi_d, \mathcal{S}, \mathcal{Z}}$, we use the similar proof strategy explored in Lemma 5.1. We develop several bridge hybrid worlds between the $\mathcal{F}_{\text{CRS}}$-hybrid world and the ideal world,

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

Proof stage: When party P is invoked with input $(\texttt{ProveSPZK}, sid, P, x, w)$ 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 $C_0 \leftarrow \texttt{EQCcom}(pk_{\mathrm{eqc}}, w; \eta)$ 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 $(\texttt{prover}_1, sid, C_0, C_1)$ to party V (through $\mathcal{Z}$).

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

When party P is invoked with incoming $\texttt{verifier}_1$ message, it computes $K \leftarrow K_1K_2 \bmod \mathsf{n}^2$, $E \leftarrow K^w\zeta^{\mathsf{n}} \bmod \mathsf{n}^2$, $a \leftarrow \texttt{prove}_1((x, C_0, E), (w, \eta, \zeta); r_a)$ and $c \leftarrow \texttt{COMcom}(pk_{\mathrm{com}}, a; r)$ where $\zeta \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}}$, $r_a, r \xleftarrow{r} \texttt{RND}$, and sends message $(\texttt{prover}_2, sid, K_1, \mu_1, x, E, c)$ to party P (through $\mathcal{Z}$).

When party V is invoked with incoming $\texttt{prover}_2$ 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 $(\texttt{verifier}_2, sid, e)$ to party P (through $\mathcal{Z}$).

When party P is invoked with incoming $\texttt{verifier}_2$ message, it computes $z \leftarrow \texttt{prove}_3((x, C_0, E), (w, \eta, \zeta), r_a, e)$, and sends message $(\texttt{prover}_3, sid, a, r, z)$ to party V (through $\mathcal{Z}$).

When party V is invoked with incoming $\texttt{prover}_3$ message, it verifies $c = \texttt{COMcom}(pk_{\mathrm{com}}, a; r)$ and $\texttt{verify}((x, C_0, E), a, e, z) = 1$; if both hold, then it returns message $(\texttt{VerifiedSVZK}, sid, x)$ to $\mathcal{Z}$.

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

Figure 18: Single-verifier zero-knowledge protocol $\pi_{\Sigma(\mathrm{SVZK})}$ for relation R_U in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world.

and define the ensemble of random variables of $\mathcal{Z}$'s output of each bridge hybrid worlds as $\mathrm{EXEC}^{\mathcal{F}_i}_{\pi_d,\mathcal{S}_i,\mathcal{Z}}$, $i = 1, 2, 3$, where π_d is the dummy protocol same as that in the ideal world. Next we prove $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{CRS}}}_{\pi_{\Sigma(\mathrm{SVZK})},\mathcal{Z}}$ $\approx \mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} \approx \cdots \approx \mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}} \approx \mathrm{EXEC}^{\mathcal{F}^{R_U}_{\mathrm{SVZK}}}_{\pi_d,\mathcal{S},\mathcal{Z}}$. In our sequence of games we introduce a functionality called the "vault" which gradually becomes from dummy SVZK functionality $\mathcal{F}_1$ into the ideal functionality $\mathcal{F}^{R_U}_{\mathrm{SVZK}}$ across a sequence of three steps. Note that we assume the protocol is based on a single-verifier mixed commitment scheme which is based on the DCR assumption, and the underlying commitments $\texttt{EQC}$ and $\texttt{COM}$ are equivocal, and $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate}, \texttt{reconstruct}\rangle$ is a non-erasure Sigma protocol.

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

The vault $\mathcal{F}_1$ is between the dummy parties P, V and the simulator $\mathcal{S}_1$; the vault $\mathcal{F}_1$ receives the outputs from the dummy parties and forwards them to $\mathcal{S}_1$ (and $\mathcal{S}_1$ will supply the messages as inputs of the simulated parities of protocol $\pi_{\Sigma(\mathrm{SVZK})}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model); $\mathcal{F}_1$ forwards the outputs of the simulated parties in $\mathcal{S}_1$ to the dummy parties which are finally returned to the environment $\mathcal{Z}$.

Analysis:

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

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 $\mathcal{S}_2$. Furthermore, $\mathcal{F}_2$ records some information, like the ideal functionality $\mathcal{F}^{R_U}_{\text{SVZK}}$, into $history(P)$: once receiving an input $(\texttt{ProveSVZK}, sid, P, x, w)$ from the dummy user P, if $(x, w) \in R_U$ the vault $\mathcal{F}_2$ records $\langle x, w\rangle$ in $history(P)$. The change in this step is that $\mathcal{F}_2$ will "block" the real witness w and only send the statement x to $\mathcal{S}_2$, i.e. send message $(\texttt{ProveSVZK}, sid, P, x)$ to $\mathcal{S}_2$.

$\mathcal{S}_2$ is same as $\mathcal{S}_1$ to simulate the whole $\mathcal{F}_{\text{CRS}}$-hybrid world except: $\mathcal{S}_2$ simulates the party P without using the real witness which has been described in the ideal world simulator. But $\mathcal{S}_2$ simulates party V same as in the $\mathcal{F}_{\text{CRS}}$-hybrid world; be more explicit when P is corrupted, $\mathcal{S}_2$ will not extract the witness for patching.

Analysis:

In this step, $\mathcal{S}_2$ can make perfect equivocation except negligible distance, which means $\mathcal{Z}$ cannot distinguish the two worlds except negligible distance. Next we calculate the distance. The commitments C_0 and c can be equivocated except probability $\mathsf{Adv}^{\text{eqc}}_{\text{eq}} + \mathsf{Adv}^{\text{com}}_{\text{eq}}$ given $\texttt{EQC}$ and $\texttt{COM}$ are equivocal commitment schemes, where $\mathsf{Adv}^{\text{eqc}}_{\text{eq}}$ and $\mathsf{Adv}^{\text{com}}_{\text{eq}}$ are equivocality distances for the two commitments. Under DCR assumption, the commitment E can always be equivocated, so here the distance introduced is $\mathsf{Adv}_{\text{dcr}}$. The underlying zero-knowledge proof of membership will introduce $\mathsf{Adv}^{\text{sigma}}_{\text{nezk}} + \mathsf{Adv}_{\text{dcr}}$ where $\mathsf{Adv}^{\text{sigma}}_{\text{nezk}}$ is the non-erasure honest-verifier zero-knowledge distance for the non-erasure Sigma protocol, and $\mathsf{Adv}_{\text{dcr}}$ is due to the fact that we use Damgård-Jurik's encryption to encrypt the witness. So $|\text{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} - \text{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}| \leq q_3 \cdot (\mathsf{Adv}^{\text{eqc}}_{\text{eq}} + \mathsf{Adv}^{\text{com}}_{\text{eq}} + \mathsf{Adv}^{\text{sigma}}_{\text{nezk}} + 2 \cdot \mathsf{Adv}_{\text{dcr}})$ where q_3 is the number of the provers which are not corrupted initially.

EXEC$^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Here $\mathcal{F}_3$ is same as $\mathcal{F}^{R_U}_{\text{SVZK}}$ and $\mathcal{S}_3$ is same as the ideal simulator $\mathcal{S}$ which will be described immediately after the proof. The difference between the $\mathcal{F}_2$-hybrid world and the $\mathcal{F}_3$-hybrid world is: in the case that some party P is corrupted, in the $\mathcal{F}_2$-hybrid world, when party V verifies all the equations, message $(\texttt{VerifiedSVZK}, sid, P, x)$ will be sent to dummy V (then to $\mathcal{Z}$); while in the $\mathcal{F}_3$-hybrid world, when party V verifies all the equations, $\mathcal{S}_3$ needs to extract the witness w and patch $\langle x, w\rangle$ to $\mathcal{F}_3$, after verifying $(x, w) \in R_U$, message $(\texttt{VerifiedSVZK}, sid, P, x)$ will be sent to dummy V (then to $\mathcal{Z}$).

Analysis:

In the case that prover P is corrupted, the simulator $\mathcal{S}_3$ will extract the witness and patch it into the functionality after verifying the proof. Assume the underlying Sigma protocol with the equivocal commitment $\texttt{COM}$ have no error, if the prover is corrupted by $\mathcal{Z}$ before the prover sends out the $\texttt{prover}_2$ message, the environment $\mathcal{Z}$ cannot obtain an E-key K except negligible probability $2^{-\ell_{\mathsf{n}}}$, which can be viewed as extraction error for the underlying single-verifier mixed commitment. Note that if $\mathcal{Z}$ let the K be an E-key, then $\mathcal{Z}$ cannot figure out K_1 to satisfy the two conditions as follows at the same time: *(i)* consistent with the commitment C_1; *(ii)* $K_1 = K/K_2 \bmod \mathsf{n}^2$, where K_2 is randomly selected. Still we need to consider a special case: when an honest prover P sends out the $\texttt{prover}_2$ message, $\mathcal{Z}$ can "capture" the used E-key K by computing $K = K_1 K_2 \bmod \mathsf{n}^2$; and then $\mathcal{Z}$ corrupts the prover P and computes a commitment E for a different $\widehat{w}$, and sends out the modified $\texttt{prover}_2$ message (note that $\mathcal{Z}$ can do this in the name of the corrupted user). Note that C_0 is produced by P when it is honest, which means C_0 is a commitment based the real witness w; and also note that the commitment is binding, and the Sigma protocol has no soundness error. The probability to produce a valid proof for the "fake" E and the real C_0 is $\mathsf{Adv}^{\text{eqc}}_{\text{binding}}$.

Now we still need to consider that error of the underlying Sigma protocol with the equivocal commitment $\texttt{COM}$. The proof idea here follows that in [Dam00]; please also refer to section 5 in [Nie03].

Define the event $\mathbf{E}$ that the pair $\langle \widehat{x}, \widehat{w} \rangle$ patched by $\mathcal{S}_3$ such that $(\widehat{x}, \widehat{w}) \notin R_U$. Notice that, if such event $\mathbf{E}$ does not occur, then $\mathcal{Z}$ cannot distinguish the two worlds because the views of them have the same distribution. Now we investigate the probability of the event $\mathbf{E}$.

When simulator $\mathcal{S}_3$ accepts a proof for statement x, then $\mathcal{S}_3$ receives $\langle a, r, z \rangle$ for a corrupted prover where $c = \texttt{COMcom}(pk_{\text{com}}, a; r)$ and $\texttt{verify}(x, a, e, z) = 1$. Now rewind the state of the simulation and let the simulator $\mathcal{S}_3$ send a new challenge e'. Repeat this until receiving $\langle a', r', z' \rangle$, where $c = \texttt{COMcom}(pk_{\text{com}}, a'; r')$ and $\texttt{verify}(x, a', e', z') = 1$. When this happens, if $a \neq a'$, then there exists a double opening of commitment value c, i.e. $c = \texttt{COMcom}(pk_{\text{com}}, a; r) = \texttt{COMcom}(pk_{\text{com}}, a'; r')$. Else if $a = a'$, and $e \neq e'$, then the membership soundness will not be satisfied. Else, if $a = a'$ and $e \neq e'$, then $\mathcal{S}_3$ gives up. So for one successful protocol for each corrupted prover, the two worlds can be distinguished with probability $\nu \cdot (\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}})$.

So $|\text{EXEC}^{\mathcal{F}_2}_{\pi_d, \mathcal{S}_2, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_3}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}| \leq q_1 \cdot (\mathsf{Adv}^{\text{eqc}}_{\text{binding}} + \nu \cdot (\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}}) + 2^{-\ell_{\mathsf{n}}})$ where q_1 is the number of the provers which are corrupted initially.

Note that the vault $\mathcal{F}_3$ is exactly the functionality $\mathcal{F}^{R_U}_{\text{SVZK}}$ and $\mathcal{S}_3$ is same as $\mathcal{S}$ in the ideal world. So $\text{EXEC}^{\mathcal{F}^{R_U}_{\text{SVZK}}}_{\pi_d, \mathcal{S}, \mathcal{Z}} = \text{EXEC}^{\mathcal{F}_3}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}$.

Based on all discussions above, we obtain $\left| \text{EXEC}^{\mathcal{F}^{R_U}_{\text{SVZK}}}_{\pi_d, \mathcal{S}, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_{\text{CRS}}}_{\pi_{\Sigma(\text{SVZK})}, \mathcal{Z}} \right| \leq q_3 \cdot (\mathsf{Adv}^{\text{eqc}}_{\text{eq}} + \mathsf{Adv}^{\text{com}}_{\text{eq}} + \mathsf{Adv}^{\text{sigma}}_{\text{nezk}} + 2 \cdot \mathsf{Adv}_{\text{dcr}}) + q_1 \cdot (\mathsf{Adv}^{\text{eqc}}_{\text{binding}} + \nu \cdot (\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}}) + 2^{-\ell_{\mathsf{n}}})$.

□

The ideal world simulator $\mathcal{S}$ proceeds as follows. Here we give a full description of the ideal world simulator $\mathcal{S}$.

Setup:
The simulator $\mathcal{S}$ generates the CRS for each party, and keeps the trapdoor for himself: generate a Paillier public key $\langle \mathsf{n}, \mathsf{g} \rangle$ along with the X-trapdoor $\langle \mathsf{p}, \mathsf{q} \rangle$; generate an equivocal key K with the E-trapdoor τ_{K} where $\mathsf{K} = (\tau_{\mathsf{K}})^{\mathsf{n}^2} \bmod \mathsf{n}^3$; run $(pk_{\text{eqc}}, ek_{\text{eqc}}) \leftarrow \texttt{EQCgen}(1^\lambda)$ and $(pk_{\text{com}}, ek_{\text{com}}) \leftarrow \texttt{COMgen}(1^\lambda)$. Let $crs = \langle \mathsf{n}, \mathsf{g}; \mathsf{K}; pk_{\text{eqc}}, pk_{\text{com}} \rangle$ and $\tau = \langle \mathsf{p}, \mathsf{q}; \tau_{\mathsf{K}}; ek_{\text{eqc}}, ek_{\text{com}} \rangle$.

Simulation of the proof stage:

(1) When the dummy prover P receives an input $(\texttt{ProveSVZK}, sid, P, x, w)$ from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}^{R_U}_{\text{SVZK}}$; if $(x, w) \in R_U$, then $\mathcal{F}^{R_U}_{\text{SVZK}}$ will record $\langle x, w \rangle$ in $history(P)$ and send $(\texttt{ProveSVZK}, sid, P, x)$ to $\mathcal{S}$. Once receiving this message from $\mathcal{F}^{R_U}_{\text{SVZK}}$, $\mathcal{S}$ simulates the party P as follows:

- randomly select $\widetilde{K}_1 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}^2}$, $\widetilde{\mu}_1 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}}$, and compute $C_1 = \mathsf{K}^{\widetilde{K}_1} (\widetilde{\mu}_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$;
- run $(C_0, aux_{\text{eqc}}) \leftarrow \texttt{EQCfake}(pk_{\text{eqc}}, ek_{\text{eqc}})$ without using the equivocal trapdoor ek_{eqc} for producing C_0, i.e. randomly select $\widetilde{w}, \widetilde{\eta} \xleftarrow{\mathsf{r}} \texttt{RND}$, compute $C_0 \leftarrow \texttt{EQCcom}(pk_{\text{eqc}}, \widetilde{w}; \widetilde{\eta})$, and set $aux_{\text{eqc}} \leftarrow \langle ek_{\text{eqc}}, \widetilde{w}, \widetilde{\eta} \rangle$

Then $\mathcal{S}$ simulates the party P to send message $(\texttt{prover}_1, sid, C_0, C_1)$ to the simulated verifier V (through the simulated $\mathcal{Z}$).

(1$^+$) If the simulator $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, P)$ from $\mathcal{Z}$ after the party P sending out the message $(\texttt{prover}_1, sid, C_0, C_1)$, but before receiving the message $(\texttt{verifier}_1, sid, K_2)$, $\mathcal{S}$ sends $(\texttt{Corrupt}, sid, P)$ to $\mathcal{F}^{R_U}_{\text{SVZK}}$, and receives $(\texttt{Corrupted}, sid, P, history(P))$ from $\mathcal{F}^{R_U}_{\text{SVZK}}$, where $history(P)$ as defined in

$\mathcal{F}_{\text{SVZK}}^{R_U}$ includes w. Then $\mathcal{S}$ runs $\eta \leftarrow \texttt{EQCequivocate}(pk_{\text{eqc}}, C_0, aux_{\text{eqc}}, w)$, and returns $\langle w, \widetilde{K}_1, \widetilde{\mu}_1, \eta\rangle$ as the internals to $\mathcal{Z}$.

(2) When $\mathcal{Z}$ delivers message $(\texttt{prover}_1, sid, C_0, C_1)$ from P to V, $\mathcal{S}$ simulates the party V easily: randomly selects $K_2 \xleftarrow{\mathsf{r}} \mathbb{Z}_{\mathsf{n}^2}^*$ and returns it to party P.

Note that there is no secret coins involved for the party V. So when $\mathcal{S}$ receives the message of corrupting V, no internals will be returned to $\mathcal{Z}$.

(3) When $\mathcal{Z}$ delivers message $(\texttt{verifier}_1, sid, K_2)$ from V to P, $\mathcal{S}$ simulates the party P:

- generate K with equivocal trapdoor τ_K as follows: randomly select $\tau_K \xleftarrow{\mathsf{r}} \mathbb{Z}_{\mathsf{n}}^*$ and compute $K \leftarrow (\tau_K)^{\mathsf{n}} \bmod \mathsf{n}^2$; then randomly select $\widetilde{\zeta} \xleftarrow{\mathsf{r}} \mathbb{Z}_{\mathsf{n}}^*$, and compute $E \leftarrow K^{\widetilde{w}}\widetilde{\zeta}^{\mathsf{n}} \bmod \mathsf{n}^2$; then compute $K_1 \leftarrow K/K_2 \bmod \mathsf{n}^2$, and open C_1 into $\mu_1 \leftarrow \tau_{\mathsf{K}}^{\widetilde{K}_1 - K_1}\widetilde{\mu}_1 \bmod \mathsf{n}$.

- randomly select $\widetilde{e} \xleftarrow{\mathsf{r}} \{0,1\}^{\ell_e}$, and run $(\widetilde{a}, \widetilde{z}, aux_{\text{sigma}}) \leftarrow \texttt{simulate}((x, C_0, E), \widetilde{e})$.

- run $(c, aux_{\text{com}}) \leftarrow \texttt{COMfake}(pk_{\text{com}}, ek_{\text{com}})$ without using equivocal trapdoor ek_{com} for producing c, i.e. compute $c \leftarrow \texttt{COMcom}(pk_{\text{com}}, \widetilde{a}; \widetilde{r})$, and set $aux_{\text{com}} \leftarrow \langle ek_{\text{com}}, \widetilde{a}, \widetilde{r}\rangle$.

Then $\mathcal{S}$ simulates the party P to return message $(\texttt{prover}_2, sid, K_1, \mu_1, x, E, c)$ to the party V (through $\mathcal{Z}$).

(3$^+$) If the simulator $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, P)$ from $\mathcal{Z}$ after the party P sending out the $\texttt{prover}_2$ message, and before receiving the $\texttt{verifier}_2$ message, $\mathcal{S}$ sends $(\texttt{Corrupt}, sid, P)$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$, and receives $(\texttt{Corrupted},\ sid, P, history(P))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, where $history(P)$ includes w. As in **(1$^+$)**, $\mathcal{S}$ runs $\eta \leftarrow \texttt{EQCequivocate}(pk_{\text{eqc}}, C_0, aux_{\text{eqc}}, w)$. Furthermore $\mathcal{S}$ uses the trapdoor τ_K to compute ζ such that $E = K^w\zeta^{\mathsf{n}} \bmod \mathsf{n}^2$ as below: $\zeta = (\tau_K)^{\widetilde{w}-w}\widetilde{\zeta} \bmod \mathsf{n}$ where $\widetilde{w}, \widetilde{\zeta}$ as used in **(3)**; then $\mathcal{S}$ runs $r_a \leftarrow \texttt{reconstruct}((x, C_0, E), \widetilde{a}, \widetilde{z}, aux_{\text{sigma}}, \widetilde{e}, w)$; $\mathcal{S}$ returns $\langle w, \eta, \zeta, r_a, \widetilde{r}\rangle$ as the internals to $\mathcal{Z}$.

(4) When $\mathcal{Z}$ delivers message $(\texttt{prover}_2, sid, K_1, \mu_1, x, E, c)$ from P to V, $\mathcal{S}$ simulates the party V as follows: verify if $C_1 = \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$ and return $(\texttt{verifier}_2, sid, \widetilde{e})$ to the party P. Note that the simulated V is an honest verifier and $\widetilde{e}$ is the one selected in **(3)**.

Same as in **(2)** when $\mathcal{S}$ receives the message of corrupting V, no internals will be returned to $\mathcal{Z}$.

(5) When $\mathcal{Z}$ delivers message $(\texttt{verifier}_2, sid, e)$ from V to P, $\mathcal{S}$ simulates the party P:

- if $e = \widetilde{e}$, which means V is honest, then let $a = \widetilde{a}$, $z = \widetilde{z}$, $r = \widetilde{r}$;

- else if $e \neq \widetilde{e}$, which means V is corrupted, then run $(a, z, aux_{\text{sigma}}) \leftarrow \texttt{simulate}((x, C_0, E), e)$; then run $r \leftarrow \texttt{COMequivocate}(pk_{\text{com}}, c, aux_{\text{com}}, a)$.

Then $\mathcal{S}$ simulates P to send message $(\texttt{prover}_3, sid, a, r, z)$ to V (through $\mathcal{Z}$).

(5$^+$) If the simulator $\mathcal{S}$ receives $(\texttt{Corrupt}, sid, P)$ from $\mathcal{Z}$ after party P sending out the $\texttt{prover}_3$ message, $\mathcal{S}$ sends $(\texttt{Corrupt}, sid, P)$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$, and receives $(\texttt{Corrupted}, sid, U, history(P))$ from $\mathcal{F}_{\text{SVZK}}^{R_U}$, where $history(P)$ includes w. Same as in **(3$^+$)**, $\mathcal{S}$ obtains ζ and η. But $\mathcal{S}$ still needs to return r_a. If verifier V is honest, $\mathcal{S}$ uses the same method in **(3$^+$)**; if verifier V is not honest, $\mathcal{S}$ runs $r_a \leftarrow \texttt{reconstruct}((x, C_0, E), a, z, aux_{\text{sigma}}, e, w)$. Then $\mathcal{S}$ returns $\langle w, \eta, \zeta, r_a\rangle$ as the internals to $\mathcal{Z}$.

(6) When $\mathcal{Z}$ delivers message $(\texttt{prover}_3, sid, a, r, z)$ from P to V, $\mathcal{S}$ simulates the party V to verify the equations $c = \texttt{COMcom}(pk_{\text{com}}, a; r)$ and $\texttt{verify}((x, C_0, E), a, e, z) = 1$. If both hold, then the simulator $\mathcal{S}$ returns $(\texttt{ProveSVZK}, sid, P, \mathsf{VerifierComplete})$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$, and $\mathcal{F}_{\text{SVZK}}^{R_U}$ returns $(\texttt{VerifiedSVZK}, sid, P, x)$ to dummy V, then to $\mathcal{Z}$. If one of the two equations does not hold, then $\mathcal{S}$ returns $(\texttt{ProveSVZK}, sid, P, \mathsf{VerifierError})$ to $\mathcal{F}_{\text{SVZK}}^{R_U}$, and $\mathcal{F}_{\text{SVZK}}^{R_U}$ returns $(\texttt{VerifiedSVZK}, sid, P, \perp)$ to dummy V, then to $\mathcal{Z}$.

In the case that P is corrupted, $\mathcal{S}$ uses the extractable trapdoor $\langle \mathsf{p}, \mathsf{q} \rangle$ to decrypt K and E. If K is an X-key, then w can be extracted from E and the pair $\langle x, w \rangle$ will be "patched" into $\mathcal{F}_{\text{SVZK}}^{R_U}$; If $\mathcal{F}_{\text{SVZK}}^{R_U}$ verifies $(x, w) \in R_U$, then it returns message $(\texttt{Verified}, sid, P, x)$ to dummy V, then to $\mathcal{Z}$; If $(x, w) \notin R_U$, then it returns message $(\texttt{Verified}, sid, P, \mathsf{error})$ to dummy V, then to $\mathcal{Z}$. If K is an E-key, then $\mathcal{S}$ halts.

Remark 5.4. 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 commitment C_0 that is made to w by the prover in the very first communication flow of the protocol in Figure 17 is unnecessary. Thus the commitment scheme $\texttt{EQC}$ may be dropped entirely from the realization; in this setting, x will be playing the role of C_0. 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 in our efficient protocol instantiation in Section 6).

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 general context of universally composable zero-knowledge in [KZ07]; 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.

Similar to the realization of $\mathcal{F}_{\text{SVZK}}^{R_U}$, for $(x, w) \in R_S$, we have the prover commit the witness w into C, and then develop a Sigma protocol to show the consistency between the commitment C and the statement x by performing a proof of language membership. But here we only need employ an extractable commitment considering we only need to realize $\mathcal{F}_{\text{SPZK}}^{R_S}$ against static adversary. Please refer to Figure 20 for an SPZK protocol against static adversary, where $\texttt{EXC}$ is an extractable commitment, $\texttt{COM}$ is an equivocal commitment, and $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate} \rangle$ is a Sigma protocol for relation R'_S as defined follows: $R'_S = \{(crs, x, E), (w, \zeta) \mid (x, w) \in R_S \wedge E = \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)\}$. The combination of the extractable commitment with the above Sigma protocol is shown in Figure 19.

Theorem 5.5. *Given an extractable commitment scheme* $\texttt{EXC}$*, an equivocal commitment scheme* $\texttt{COM}$*, and a Sigma protocol* $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate} \rangle$*, the SPZK protocol* $\pi_{\Sigma(\text{SPZK})}$ *in Figure 20 securely realizes* $\mathcal{F}_{\text{SPZK}}^{R_S}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model against static adversary with advantage* $(q_1 + q_3) \cdot (\mathsf{Adv}_{\text{hiding}}^{\text{exc}} + \mathsf{Adv}_{\text{eq}}^{\text{com}} + \mathsf{Adv}_{\text{zk}}^{\text{sigma}}) + q_2 \cdot \nu \cdot (\mathsf{Adv}_{\text{binding}}^{\text{com}} + \mathsf{Adv}_{\text{sound}}^{\text{sigma}})$*, where* q_3 *is the number of the verifiers which are not corrupted initially,* q_1 *is the number of the verifiers corrupted initially,* q_2 *is the number of successful protocols from the corrupted prover,* $\mathsf{Adv}_{\text{binding}}^{\text{exc}}$ *is binding distance for the extractable commitment* $\texttt{EXC}$*,* $\mathsf{Adv}_{\text{eq}}^{\text{com}}$ *and* $\mathsf{Adv}_{\text{binding}}^{\text{com}}$ *are equivocality distance and binding distance for the equivocal commitment* $\texttt{COM}$*,* $\mathsf{Adv}_{\text{zk}}^{\text{sigma}}$ *and* $\mathsf{Adv}_{\text{sound}}^{\text{sigma}}$ *are honest-verifier zero-knowledge distance and soundness distance for the Sigma protocol.*

Proof. In order to prove that $\text{EXEC}_{\pi_{\Sigma(\text{SPZK})}, \mathcal{Z}}^{\mathcal{F}_{\text{CRS}}} \approx \text{EXEC}_{\pi_d, \mathcal{S}, \mathcal{Z}}^{\mathcal{F}_{\text{SPZK}}^{R_S}}$, we use the similar proof strategy explored in Lemma 5.1. We develop several bridge hybrid worlds between the $\mathcal{F}_{\text{CRS}}$-hybrid world and the ideal world, and define the ensemble of random variables of $\mathcal{Z}$'s output of each bridge hybrid worlds as $\text{EXEC}_{\pi_d, \mathcal{S}_i, \mathcal{Z}}^{\mathcal{F}_i}$, $i = 1, 2, 3$, where π_d is the dummy protocol same as that in the ideal world. Next we prove $\text{EXEC}_{\pi_{\Sigma(\text{SPZK})}, \mathcal{Z}}^{\mathcal{F}_{\text{CRS}}}$ $\approx \text{EXEC}_{\pi_d, \mathcal{S}_1, \mathcal{Z}}^{\mathcal{F}_1} \approx \cdots \approx \text{EXEC}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}^{\mathcal{F}_3} \approx \text{EXEC}_{\pi_d, \mathcal{S}, \mathcal{Z}}^{\mathcal{F}_{\text{SPZK}}^{R_S}}$. In our sequence of games we introduce a functionality called the "vault" which gradually becomes from dummy SPZK functionality $\mathcal{F}_1$ into the

$$crs = \langle pk_{\text{exc}} \rangle$$

$\boxed{P}$ $\boxed{V}$

$\boxed{P}$		$\boxed{V}$
$statement = \langle x \rangle$ $witness = \langle w \rangle$		$statement = \langle x \rangle$
$\zeta \xleftarrow{r} \texttt{RND};$ $E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)$ $r_a, r \xleftarrow{r} \texttt{RND}$ $a \leftarrow \texttt{prove}_1((x, E), (w, \zeta); r_a)$		
	$\xrightarrow{E,a}$	
		$e \xleftarrow{r} \{0,1\}^{\ell_e}$
	$\xleftarrow{e}$	
$z \leftarrow \texttt{prove}_3((x, E), (w, \zeta), r_a, e)$		
	$\xrightarrow{z}$	
		$\texttt{verify}((x, E), a, e, z) =^? 1$

Figure 19: 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.

ideal functionality $\mathcal{F}_{\text{SPZK}}^{R_S}$ across a sequence of three steps. Note that we assume the protocol is based on an extractable commitment scheme, and the underlying commitments EQC and COM are equivocal, and $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate} \rangle$ is a Sigma protocol.

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

The vault $\mathcal{F}_1$ is between the dummy parties P, V and the simulator $\mathcal{S}_1$; the vault $\mathcal{F}_1$ receives the outputs from the dummy parties and forwards them to $\mathcal{S}_1$ (and $\mathcal{S}_1$ will supply the messages as inputs of the simulated parities of protocol $\pi_{\Sigma(\text{SPZK})}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model); $\mathcal{F}_1$ forwards the outputs of the simulated parties in $\mathcal{S}_1$ to the dummy parties which are finally returned to the environment $\mathcal{Z}$.

Analysis:

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

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 $\mathcal{S}_2$. Furthermore, $\mathcal{F}_2$ records some information, like the ideal functionality $\mathcal{F}_{\text{SPZK}}^{R_S}$, into $history(P)$: upon receiving an input $(\texttt{ProveSPZK}, sid, V, x, w)$ from the dummy party P, if $(x, w) \in R_S$ the vault $\mathcal{F}_2$ records $\langle V, x, w \rangle$ in $history(P)$. The change in this step is that $\mathcal{F}_2$ will "block" the real witness w and only send the statement x to $\mathcal{S}_2$, i.e. send message $(\texttt{ProveSPZK}, sid, V, x)$ to $\mathcal{S}_2$.

$\mathcal{S}_2$ is same as $\mathcal{S}_1$ to simulate the whole $\mathcal{F}_{\text{CRS}}$-hybrid world except: $\mathcal{S}_2$ simulates the party P without using the real witness which has been described in the ideal world simulator. But $\mathcal{S}_2$ simulates party

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

Proof stage: When party P is invoked with input $(\texttt{ProveSPZK}, sid, V, x, w)$ 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 $E \leftarrow \texttt{EXCcom}(pk_{\mathrm{exc}}, w; \zeta)$, $a \leftarrow \texttt{prove}_1((x, E), (w, \zeta); r_a)$ and $c \leftarrow \texttt{COMcom}(pk_{\mathrm{com}}, a; r)$ where $\zeta, r_a, r \overset{\mathrm{r}}{\leftarrow} \texttt{RND}$, and then sends message $(\texttt{prover}_1, sid, x, E, c)$ to party V (through $\mathcal{Z}$).

When party V is invoked with incoming $\texttt{prover}_1$ message, it randomly selects $e \overset{\mathrm{r}}{\leftarrow} \{0,1\}^{\ell_e}$, and sends message $(\texttt{verifier}_1, sid, e)$ to party P (through $\mathcal{Z}$).

When party P is invoked with incoming $\texttt{verifier}_1$ message, it computes $z \leftarrow \texttt{prove}_3((x, E), (w, \zeta), r_a, e)$, and sends message $(\texttt{prover}_2, sid, a, r, z)$ to party V (through $\mathcal{Z}$).

When party V is invoked with incoming $\texttt{prover}_2$ message, it verifies $c = \texttt{COMcom}(pk_{\mathrm{com}}, a; r)$ and $\texttt{verify}((x, E), a, e, z) = 1$; if both hold, then it returns message $(\texttt{VerifiedSPZK}, sid, x)$ to $\mathcal{Z}$.

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

Figure 20: Single-prover zero-knowledge protocol $\pi_{\Sigma(\mathrm{SPZK})}$ for relation R_S in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world.

V same as in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world; be more explicit when P is corrupted, $\mathcal{S}_2$ will not extract the witness for patching.

Analysis:

In this step, $\mathcal{S}_2$ can make perfect simulation except negligible distance: the commitments E is hiding except probability $\mathsf{Adv}^{\mathrm{exc}}_{\mathrm{hiding}}$ given $\texttt{EXC}$ is an extractable commitment, where $\mathsf{Adv}^{\mathrm{exc}}_{\mathrm{hiding}}$ is hiding distance; and commitment c is equivocal except probability $\mathsf{Adv}^{\mathrm{com}}_{\mathrm{eq}}$ given $\texttt{COM}$ is an equivocal commitment schemes, and $\mathsf{Adv}^{\mathrm{com}}_{\mathrm{eq}}$ are equivocality distance; the underlying zero-knowledge proof of membership will introduce distance $\mathsf{Adv}^{\mathrm{sigma}}_{\mathrm{zk}}$ where $\mathsf{Adv}^{\mathrm{sigma}}_{\mathrm{zk}}$ is the honest-verifier zero-knowledge distance for the Sigma protocol. So $|\mathrm{EXEC}^{\mathcal{F}_1}_{\pi_d,\mathcal{S}_1,\mathcal{Z}} - \mathrm{EXEC}^{\mathcal{F}_2}_{\pi_d,\mathcal{S}_2,\mathcal{Z}}| \le (q_1 + q_3) \cdot (\mathsf{Adv}^{\mathrm{exc}}_{\mathrm{hiding}} + \mathsf{Adv}^{\mathrm{com}}_{\mathrm{eq}} + \mathsf{Adv}^{\mathrm{sigma}}_{\mathrm{zk}})$ where q_1 is the number of the verifiers which are not corrupted initially, and q_3 the number of the verifiers corrupted initially.

$\mathrm{EXEC}^{\mathcal{F}_3}_{\pi_d,\mathcal{S}_3,\mathcal{Z}}$. Here $\mathcal{F}_3$ is same as $\mathcal{F}^{R_S}_{\mathrm{SPZK}}$ and $\mathcal{S}_3$ is same as the ideal simulator $\mathcal{S}$ described before. The difference between the $\mathcal{F}_2$-hybrid world and the $\mathcal{F}_3$-hybrid world is: in the case that party P is corrupted, in the $\mathcal{F}_2$-hybrid world, when party V verifies all the equations, message $(\texttt{VerifiedSPZK}, sid, P, x)$ will be sent to dummy V (then to $\mathcal{Z}$); while in the $\mathcal{F}_3$-hybrid world, when party V verifies all the equations, $\mathcal{S}_3$ needs to extract the witness w and patch it to $\mathcal{F}_3$, after verifying $(x, w) \in R_S$, message $(\texttt{VerifiedSPZK}, sid, P, x)$ will be sent to dummy V (then to $\mathcal{Z}$).

Analysis:

The proof idea here follows that in [Dam00]; please also refer to section 5 in [Nie03]. In the case that prover P is corrupted, the simulator $\mathcal{S}_3$ will extract the witness and patch it into the functionality after verifying the proof. Define the event $\mathbf{E}$ that the pair $(\widehat{x}, \widehat{w})$ patched by $\mathcal{S}_3$ such that $(\widehat{x}, \widehat{w}) \notin R_S$. Notice that, if such event $\mathbf{E}$ does not occur, then $\mathcal{Z}$ cannot distinguish the two worlds because the views of them have the same distribution. Now we investigate the probability of the event $\mathbf{E}$.

When simulator $\mathcal{S}_3$ accepts a proof for statement x, then $\mathcal{S}_3$ receives $\langle a, r, z \rangle$ for a corrupted prover where $c = \texttt{COMcom}(pk_{\mathrm{com}}, a; r)$ and $\texttt{verify}(x, a, e, z) = 1$. Now rewind the state of the simulation and let the simulator $\mathcal{S}_3$ send a new challenge e'. Repeat this until receiving $\langle a', r', z' \rangle$, where $c = \texttt{COMcom}(pk_{\text{com}}, a'; r')$ and $\texttt{verify}(x, a', e', z') = 1$. When this happens, if $a \neq a'$, then there exists a double opening of commitment value c, i.e. $c = \texttt{COMcom}(pk_{\text{com}}, a; r) = \texttt{COMcom}(pk_{\text{com}}, a'; r')$. Else if $a = a'$, and $e \neq e'$, then the membership soundness will not be satisfied. Else, if $a = a'$ and $e \neq e'$, then $\mathcal{S}_3$ gives up. So for one successful protocol for the corrupted prover, the two worlds can be distinguished with probability $\nu \cdot (\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}})$. Therefore, $|\text{EXEC}^{\mathcal{F}_2}_{\pi_d, \mathcal{S}_2, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_3}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}| \leq q_2 \cdot \nu \cdot (\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}})$ where q_2 is the number of successful protocols from the corrupted prover.

Note that the vault $\mathcal{F}_3$ is exactly the functionality $\mathcal{F}^{R_S}_{\text{SPZK}}$ and $\mathcal{S}_3$ is same as $\mathcal{S}$ in the ideal world. So $\text{EXEC}^{\mathcal{F}^{R_S}_{\text{SPZK}}}_{\pi_d, \mathcal{S}, \mathcal{Z}} = \text{EXEC}^{\mathcal{F}_3}_{\pi_d, \mathcal{S}_3, \mathcal{Z}}$.

Based on all discussions above, we obtain $\left| \text{EXEC}^{\mathcal{F}^{R_S}_{\text{SPZK}}}_{\pi_d, \mathcal{S}, \mathcal{Z}} - \text{EXEC}^{\mathcal{F}_{\text{CRS}}}_{\pi_{\Sigma(\text{SPZK})}, \mathcal{Z}} \right| \leq (q_1 + q_3) \cdot (\mathsf{Adv}^{\text{exc}}_{\text{hiding}} + \mathsf{Adv}^{\text{com}}_{\text{eq}} + \mathsf{Adv}^{\text{sigma}}_{\text{zk}}) + q_2 \cdot \nu \cdot (\mathsf{Adv}^{\text{com}}_{\text{binding}} + \mathsf{Adv}^{\text{sigma}}_{\text{sound}})$.

□

The ideal world simulator $\mathcal{S}$ proceeds as follows. Here we give a full description of ideal world simulator.

Setup:
The simulator $\mathcal{S}$ generates the CRS for each party, and keeps the trapdoor for himself: run $(pk_{\text{exc}}, xk_{\text{exc}}) \leftarrow \texttt{EXCgen}(1^\lambda)$ and $(pk_{\text{com}}, ek_{\text{com}}) \leftarrow \texttt{COMgen}(1^\lambda)$. Let $crs = \langle pk_{\text{exc}}, pk_{\text{com}} \rangle$ and $\tau = \langle xk_{\text{exc}}, ek_{\text{com}} \rangle$.

Simulation of the proof stage:
The simulation is only for static adversaries, and is much simpler than the one for adaptive adversaries.

(1) When the dummy prover P receives an input $(\texttt{ProveSPZK}, sid, V, x, w)$ from the environment $\mathcal{Z}$, it sends this message to $\mathcal{F}^{R_S}_{\text{SPZK}}$; if $(x, w) \in R_S$, then $\mathcal{F}^{R_S}_{\text{SPZK}}$ will "block" the witness w: record $\langle V, x, w \rangle$ in $history(P)$ and send $(\texttt{ProveSPZK}, sid, V, x)$ to $\mathcal{S}$. Once receiving this message from $\mathcal{F}^{R_S}_{\text{SPZK}}$, $\mathcal{S}$ simulates the party P as follows:

- randomly select $\widetilde{w}, \widetilde{\zeta} \xleftarrow{r} \texttt{RND}$, compute $E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, \widetilde{w}; \widetilde{\zeta})$.
- randomly select $\widetilde{e} \xleftarrow{r} \{0,1\}^{\ell_e}$, and run $(\widetilde{a}, \widetilde{z}, aux_{\text{sigma}}) \leftarrow \texttt{simulate}((x, E), \widetilde{e})$.
- randomly select $\widetilde{r} \xleftarrow{r} \texttt{RND}$, and compute $c \leftarrow \texttt{COMcom}(pk_{\text{com}}, \widetilde{a}; \widetilde{r})$, and set $aux_{\text{com}} \leftarrow \langle ek_{\text{com}}, \widetilde{a}, \widetilde{r} \rangle$.

Then $\mathcal{S}$ simulates the party P to send message $(\texttt{prover}_1, sid, x, E, c)$ to the simulated verifier V (through the simulated $\mathcal{Z}$).

(2) When $\mathcal{Z}$ delivers message $(\texttt{prover}_1, sid, x, E, c)$ from P to V, $\mathcal{S}$ simulates the party V to return $(\texttt{verifier}_1,\ sid, \widetilde{e})$ to the party P. Note that the simulated V is an honest verifier and $\widetilde{e}$ is the one selected in **(1)**.

(3) When $\mathcal{Z}$ delivers message $(\texttt{verifier}_1, sid, e)$ from V to P, $\mathcal{S}$ simulates the party P:

- if $e = \widetilde{e}$, which means V is honest, then let $a = \widetilde{a}$, $z = \widetilde{z}$, $r = \widetilde{r}$;
- else if $e \neq \widetilde{e}$, which means V is corrupted initially, then run $(a, z, aux_{\text{sigma}}) \leftarrow \texttt{simulate}((x, E), e)$; then run $r \leftarrow \texttt{COMequivocate}(pk_{\text{com}}, c, aux_{\text{com}}, a)$.

Then $\mathcal{S}$ simulates P to send message $(\texttt{prover}_2, sid, a, r, z)$ to V (through $\mathcal{Z}$).

(4) When $\mathcal{Z}$ delivers message $(\texttt{prover}_2, sid, a, r, z)$ from P to V, $\mathcal{S}$ simulates the party V to verify the equations $c = \texttt{COMcom}(pk_{\text{com}}, a; r)$ and $\texttt{verify}((x, E), a, e, z) = 1$. If both hold, then the simulator $\mathcal{S}$ returns $(\texttt{ProveSPZK}, sid, V, \textsf{VerifierComplete})$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, and $\mathcal{F}_{\text{SPZK}}^{R_S}$ returns $(\texttt{VerifiedSPZK}, sid, V, x)$ to dummy V, then to $\mathcal{Z}$. If one of the two equations does not hold, $\mathcal{S}$ returns $(\texttt{ProveSPZK}, sid, V, \textsf{VerifierError})$ to $\mathcal{F}_{\text{SPZK}}^{R_S}$, and $\mathcal{F}_{\text{SPZK}}^{R_S}$ returns $(\texttt{VerifiedSPZK}, sid, V, \perp)$ to dummy V, then to $\mathcal{Z}$.

In the case that P is corrupted initially, $\mathcal{S}$ uses the extractable trapdoor xk_{exc} to decrypt E into w and "patches" the pair $\langle x, w\rangle$ into $\mathcal{F}_{\text{SPZK}}^{R_S}$. If $\mathcal{F}_{\text{SPZK}}^{R_S}$ verifies $(x, w) \in R_S$, then it returns $(\texttt{VerifiedSPZK}, sid, V, x)$ to dummy V, then to $\mathcal{Z}$; If $(x, w) \notin R_S$, then return $(\texttt{VerifiedSPZK}, sid, V, \textsf{error})$ to dummy V, then to $\mathcal{Z}$. If such decryption fails, then $\mathcal{S}$ halts.

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

Proof stage: When party P is invoked with input $(\texttt{ProveSPZK}, sid, V, x, w)$ 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 $E \leftarrow \texttt{EXCcom}(pk_{\text{exc}}, w; \zeta)$, $a \leftarrow \texttt{prove}_1((x, E), (w, \zeta); r_a)$ 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 $(\texttt{prover}_1, sid, c)$ to party V (through $\mathcal{Z}$).

When party V is invoked with incoming $\texttt{prover}_1$ message, it randomly selects $e \xleftarrow{r} \{0, 1\}^{\ell_e}$, and sends message $(\texttt{verifier}_1, sid, e)$ to party P (through $\mathcal{Z}$).

When party P is invoked with incoming $\texttt{verifier}_1$ message, it computes $z \leftarrow \texttt{prove}_3((x, E), (w, \zeta), r_a, e)$, and sends message $(\texttt{prover}_2, sid, x, E, a, r, z)$ to party V (through $\mathcal{Z}$).

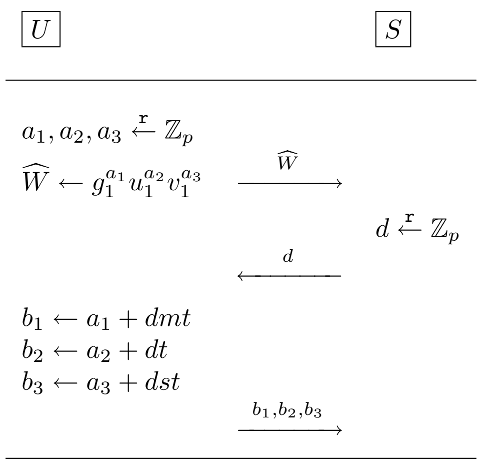
When party V is invoked with incoming $\texttt{prover}_2$ message, it verifies $c = \texttt{COMcom}(pk_{\text{com}}, a; r)$ and $\texttt{verify}((x, E), a, e, z) = 1$; if both hold, then it returns message $(\texttt{VerifiedSPZK}, sid, x)$ to $\mathcal{Z}$.

Corruption: When party P is invoked with incoming $(\texttt{Corrupt}, sid, P)$ by $\mathcal{Z}$, it sends outgoing $(\texttt{Corrupted}, sid, history(P))$ 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.

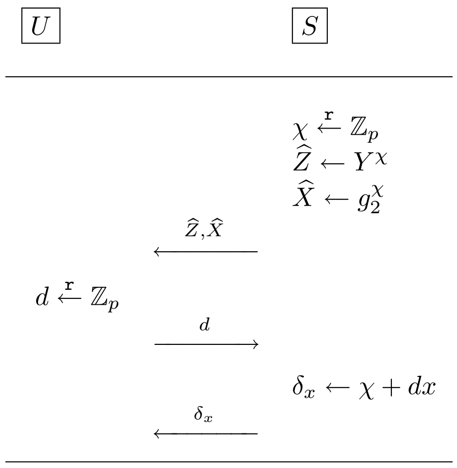
Remark 5.6. In Figure 21, we present an alternative way of transforming the two-party protocol of Figure 19 into a UC protocol. While the UC protocol implementation of Figure 20 is sufficient nevertheless there are advantages in using the alternative implementation: in particular, the UC protocol of Figure 21 is more conservative with respect to the information revealed by the prover party to a verifier party during the initial two communication moves; while the overall number of rounds is the same between the two UC protocols, the second protocol, i.e. the one in Figure 21, has the advantage that it may be initiated earlier by a prover that is acting also as a verifier in a more complex protocol interaction. In other words the advantage of the second realization is its potential in reducing the number of rounds when two parties are bilaterally proving to each other statements in zero-knowledge. This property (that we will take advantage of it later on in our efficient construction) is demonstrated also in Figure 26 and Figure 27.

Theorem 5.7. *Given an extractable commitment scheme* EXC*, an equivocal commitment scheme* COM*, and a Sigma protocol* $\langle \texttt{prove}_1, \texttt{prove}_3, \texttt{verify}, \texttt{simulate}\rangle$*, 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 adversary with advantage* $(q_1 + q_3) \cdot (\mathsf{Adv}_{\text{hiding}}^{\text{exc}} + \mathsf{Adv}_{\text{eq}}^{\text{com}} + \mathsf{Adv}_{\text{zk}}^{\text{sigma}}) + q_2 \cdot \nu \cdot (\mathsf{Adv}_{\text{binding}}^{\text{com}} + \mathsf{Adv}_{\text{sound}}^{\text{sigma}})$*, where* q_3 *is the number of the verifiers which are*



$\boxed{U}$
$\boxed{S}$
$a_1, a_2, a_3 \xleftarrow{r} \mathbb{Z}_p$
$\widehat{W} \leftarrow g_1^{a_1} u_1^{a_2} v_1^{a_3}$
$\xrightarrow{\widehat{W}}$
$d \xleftarrow{r} \mathbb{Z}_p$
$\xleftarrow{d}$
$b_1 \leftarrow a_1 + dmt$
$b_2 \leftarrow a_2 + dt$
$b_3 \leftarrow a_3 + dst$
$\xrightarrow{b_1, b_2, b_3}$

Figure 22: Σ^{R_U}-protocol, where $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$.



$\boxed{U}$
$\boxed{S}$
$\chi \xleftarrow{r} \mathbb{Z}_p$
$\widehat{Z} \leftarrow Y^{\chi}$
$\widehat{X} \leftarrow g_2^{\chi}$
$\xleftarrow{\widehat{Z}, \widehat{X}}$
$d \xleftarrow{r} \mathbb{Z}_p$
$\xrightarrow{d}$
$\delta_x \leftarrow \chi + dx$
$\xleftarrow{\delta_x}$

Figure 23: Σ^{R_S}-protocol, where $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (W v_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$.

not corrupted initially, q_1 is the number of the verifiers corrupted initially, q_2 is the number of successful protocols from the corrupted prover, $\mathsf{Adv}_{\mathrm{binding}}^{\mathrm{exc}}$ *is binding distance for the extractable commitment* `EXC`, $\mathsf{Adv}_{\mathrm{eq}}^{\mathrm{com}}$ *and* $\mathsf{Adv}_{\mathrm{binding}}^{\mathrm{com}}$ *are equivocality distance and binding distance for the equivocal commitment* `COM`, $\mathsf{Adv}_{\mathrm{zk}}^{\mathrm{sigma}}$ *and* $\mathsf{Adv}_{\mathrm{sound}}^{\mathrm{sigma}}$ *are honest-verifier zero-knowledge distance and soundness distance for the Sigma protocol.*

Proof. The proof is similar to the proof of Theorem 5.5. □

6 Efficient UC Blind Signatures against Adaptive Adversaries

6.1 Overview

In this section, we demonstrate how it is possible to design an efficient instantiation of Theorem 5.2. We need three ingredients: (1) an equivocal lite blind signature scheme, (2) a UC-realization of the ideal functionality $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$, (3) a UC-realization of the ideal functionality $\mathcal{F}_{\mathrm{SPZK}}^{R_S}$. Regarding (1) we will employ the lite blind signature scheme of Figure 8 that we proved it to be equivocal in Theorem 3.13. Regarding the two ZK functionalities we will follow the design strategy outlined in the previous section. Recall that $R_U = \{((crs, vk, \mathbf{u}), (m, \rho_1)) \mid \mathbf{u} = \mathsf{lbs}_1(crs, m; \rho_1)\}$ and $R_S = \{((crs, vk, \mathbf{u}, \mathbf{s}), (sk, \rho_2)) \mid \mathbf{s} = \mathsf{lbs}_2(crs, vk, \mathbf{u}, sk; \rho_2) \wedge (vk, sk) \in \textsc{KeyPair}\}$. Instantiating these relations for the protocol of Figure 8 we obtain that $R_U = \{((crs, X, W), (m, t, s)) \mid W = g_1^{mt} u_1^t v_1^{st}\}$ and $R_S = \{((crs, X, W, Y, l, r), x) \mid Y = (W v_1^l)^{\frac{1}{x+r}} \wedge X = g_2^x\}$. Two efficient Sigma protocols Σ^{R_U} and Σ^{R_S} for these relations are presented in Figure 22 for R_U and in Figure 23 for R_S, respectively.

Efficient instantiation of $\mathcal{F}_{\mathrm{SVZK}}^{R_U}$. Based on the single-verifier UC zero-knowledge protocol in Figure 18 in the previous section, and considering the underlying equivocal lite blind signature in Figure 8, we can realize $\mathcal{F}_{\mathrm{SVZK}}^{R_S}$ efficiently: as discussed in Remark 5.4, we can "borrow" W from the underlying lite blind signature protocol as the equivocal commitment which saves some computation/communication; we will use hashed Pedersen commitment [Ped91] to develop an efficient zero-knowledge proof of membership to bind the witness in the relation R_U and the one committed based on the mixed commitment (refer to Figure 26 and Figure 27).

Corollary 6.1. *Composing Sigma protocol in Figure 22 with Sigma protocol for relation* $\{((crs, E), (w, \zeta))| E = K^w \zeta^{\mathsf{n}} \bmod \mathsf{n}^2\}$*, we obtain SVZK protocol realizing* $\mathcal{F}_{\text{SVZK}}^{R_U}$.

Efficient instantiation of $\mathcal{F}_{\text{SPZK}}^{R_S}$**.** Based on the single-prover UC zero-knowledge protocol in Figure 21 presented in the previous section, we instantiate the extractable commitment by Paillier encryption; we still use the hashed Pedersen commitment to develop efficient zero-knowledge proof of membership to bind the witness in R_S to the witness committed within the extractable commitment.

Corollary 6.2. *Composing Sigma protocol in Figure 23 with Sigma protocol for relation* $\{((crs, E), (w, \zeta))| E = \mathtt{EXCcom}(pk_{\text{exc}}, w; \zeta)\}$*, we obtain SPZK protocol realizing* $\mathcal{F}_{\text{SPZK}}^{R_S}$ *against static adversary.*

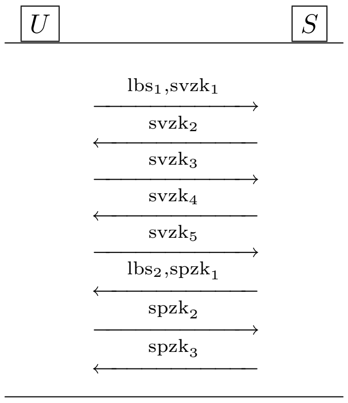


Figure 24: 8-move two side zero-knowledge proving for lite blind signature, SVZK, and SPZK.

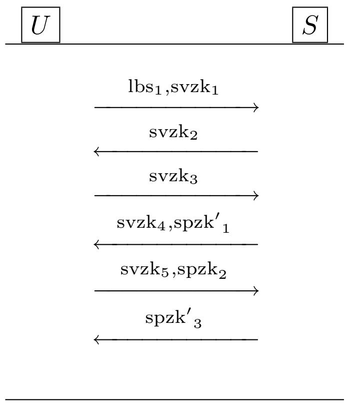


Figure 25: 6-move two side zero-knowledge proving for lite blind signature, SVZK, and SPZK. Here lbs_2 is committed inside spzk'_1, and will be opened until spzk'_3; except this difference, spzk'_1 and spzk'_3 are same as spzk_1, spzk_3 in Figure 24.

Communication rounds optimization. We can obtain an 8 moves blind signature protocol by putting the designs for the two sides together (refer to Figure 24). However, based on the discussion in Remark 5.6 in the previous section, we can achieve a 6 moves protocol by carefully interleaving the communication transcripts of the two sides (refer to Figure 25). The final 6-move protocol is presented in the next subsection. Please also refer to Figure 26 and Figure 27.

6.2 Detailed Description

In this subsection, we give a detailed description of our UC blind signature construction. 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.

Common reference string generation. 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 $\tau_{\mathsf{K}} \overset{r}{\leftarrow} \mathbb{Z}_{\mathsf{n}}$ 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,

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

$\boxed{U}$ $\qquad\qquad\qquad\qquad$ $\boxed{S}$

U		S
$vk = \langle X = g_2^x \rangle$		$vk = \langle X = g_2^x \rangle$
$msg = \langle m \rangle,\ m \in \mathbb{Z}_p$		$sk = \langle x \rangle$
$K_1 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}^2};\ \mu_1 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}};$		
$C_1 \leftarrow \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$		
$t, s \xleftarrow{\mathsf{r}} \mathbb{Z}_p;\ W \leftarrow g_1^{mt} u_1^t v_1^{st}$		
	$\xrightarrow{C_1, W}$	
		$K_2 \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}^2}$
	$\xleftarrow{K_2}$	
$K \leftarrow K_1 K_2 \bmod \mathsf{n}^2;$		
$\theta \leftarrow (mt \bmod p) + t2^\kappa + (st \bmod p)2^{2\kappa}$		
$A_\theta, B_\theta \xleftarrow{\mathsf{r}} \mathbb{Z}^*_{\mathsf{n}}$		
$E_\theta \leftarrow K^\theta (A_\theta)^{\mathsf{n}} \bmod \mathsf{n}^2$		
$a_i \xleftarrow{\mathsf{r}} \pm[0, 2^{\lambda_0 + \lambda_U + \ell_p}],\ i = 1, 2, 3$		
$\vartheta \leftarrow a_1 + a_2 2^\kappa + a_3 2^{2\kappa}$		
$\widehat{E}_\theta \leftarrow K^\vartheta (B_\theta)^{\mathsf{n}} \bmod \mathsf{n}^2$		
$\widehat{W} \leftarrow g_1^{a_1} u_1^{a_2} v_1^{a_3}$		
$\mu_2 \xleftarrow{\mathsf{r}} \mathbb{Z}_Q;\ \omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$		
$C_2 \leftarrow \mathbf{g}^{\omega_2} \mathbf{h}_2^{\mu_2}$		
	$\xrightarrow{\langle K_1, \mu_1 \rangle, \langle E_\theta, C_2 \rangle}$	
		$C_1 \overset{?}{=} \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$
		$d_U \xleftarrow{\mathsf{r}} \{0,1\}^{\lambda_U}$
		$r \xleftarrow{\mathsf{r}} \mathbb{Z}_p$ 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 27)

Figure 26: Blind signature generation protocol (part 1).

$\boxed{U}$ $\boxed{S}$

(continued from Figure 26)

$b_1 \leftarrow a_1 + d_U \cdot (mt \bmod p)$
$b_2 \leftarrow a_2 + d_U \cdot t$
$b_3 \leftarrow a_3 + d_U \cdot (st \bmod p)$
$F_\theta \leftarrow B_\theta(A_\theta)^{d_U} \bmod \mathsf{n}$
$d_S \xleftarrow{\mathtt{r}} \{0,1\}^{\lambda_S}$

$$\xrightarrow{\langle b_1,b_2,b_3,F_\theta\rangle,\langle \widehat{E}_\theta,\widehat{W},\mu_2\rangle,d_S}$$

$\omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$; $C_2 \stackrel{?}{=} \mathbf{g}^{\omega_2}\mathbf{h}_2^{\mu_2}$
$E_\theta \stackrel{?}{\in} \mathbb{Z}^*_{\mathsf{n}^2}$, $W \stackrel{?}{\in} \mathbb{G}_1$
$b_i \stackrel{?}{\in} \pm[0, 2^{\lambda_0+\lambda_U+\ell_p+1}]$, $i = 1, 2, 3$
$g_1^{b_1}u_1^{b_2}v_1^{b_3} \stackrel{?}{=} \widehat{W}W^{d_U}$
$K \leftarrow K_1K_2 \bmod \mathsf{n}^2$
$\eta \leftarrow b_1 + b_2 2^\kappa + b_3 2^{2\kappa}$
$K^\eta(F_\theta)^{\mathsf{n}} \stackrel{?}{=} \widehat{E}_\theta(E_\theta)^{d_U} \bmod \mathsf{n}^2$
$\delta_x \leftarrow \chi + d_S \cdot x,$
$F_x \leftarrow B_x(A_x)^{d_S} \bmod \mathsf{n}$

$$\xleftarrow{\langle \delta_x,F_x\rangle,\langle Y,l,r,\widehat{Z},\widehat{X},\widehat{E}_x,\mu_3\rangle}$$

$\omega_3 \leftarrow \mathcal{H}(Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x)$
$C_3 \stackrel{?}{=} \mathbf{g}^{\omega_3}\mathbf{h}_3^{\mu_3}$
$E_x \stackrel{?}{\in} \mathbb{Z}^*_{\mathsf{n}^2}$, $\delta_x \stackrel{?}{\in} \pm[0, 2^{\lambda_0+\lambda_S+\ell_p+1}]$
$g_2^{\delta_x} \stackrel{?}{=} \widehat{X}X^{d_S}$
$\mathsf{g}^{\delta_x}(F_x)^{\mathsf{n}} \stackrel{?}{=} \widehat{E}_x(E_x)^{d_S} \bmod \mathsf{n}^2$
$Y^{\delta_x} \stackrel{?}{=} \widehat{Z}(Wv_1^lY^{-r})^{d_S}$
$f, h \xleftarrow{\mathtt{r}} \mathbb{Z}_p$; $\varsigma \leftarrow Y^{\frac{1}{ft} \bmod p}$
$\alpha \leftarrow X^f g_2^{fr}$; $\beta \leftarrow s + \frac{l}{t} \bmod p$
$V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h$; $V_2 \leftarrow X^{fh+r}g_2^{frh}$
$\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2\rangle$
$\mathtt{verify}(crs, vk, m, \sigma) \stackrel{?}{=} 1$

output $(m; \sigma)$

Figure 27: Blind signature generation protocol (part 2).

let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups as defined in Section 2.2.5. Randomly select $g_2 \xleftarrow{r} \mathbb{G}_2$, $\tau_{u_2}, \tau_{v_2} \xleftarrow{r} \mathbb{Z}_p$, 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 $\tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3} \xleftarrow{r} \mathbb{Z}_Q$ 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 $\{\mathsf{p}, \mathsf{q}; \tau_{\mathsf{K}}; \tau_{u_2}, \tau_{v_2}; \tau_{\mathbf{h}_2}, \tau_{\mathbf{h}_3}\}$.

Notice that the collision hash function can be avoided at slight cost of our communication/computation efficiency.

Next we present the details of the protocol (please also refer to Figure 26 and Figure 27). Note that $d_U < p$, $d_S < p$, i.e. $\lambda_U < \ell_p$, $\lambda_S < \ell_p$, and $\kappa > \lambda_0 + \lambda_U + \ell_p + 3$, $\ell_\mathsf{n} \geq 3\kappa$. Here $\ell_p, \ell_\mathsf{n}, \ell_Q$ denote the lengths of p, n, Q.

Key generation:

(a) On input $(\mathtt{KeyGen}, sid)$, party S verifies that $sid = (S, sid')$ for some sid'. If not, it ignores the input. Else, party S sends outgoing message $(\mathtt{GetCRS}, sid)$ to $\mathcal{F}_{\text{CRS}}$ and receives crs. Then based on parameters $\langle p, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{\mathsf{e}}, \psi, u_2, v_2\rangle$, party S runs $(vk, sk) \leftarrow \mathtt{gen}(crs)$; $sk = \langle x\rangle$, $vk = \langle X\rangle$ where $x \xleftarrow{r} \mathbb{Z}_p$, $X = g_2^x$. Define $\mathsf{sig} \overset{\mathsf{def}}{=} \mathtt{sign}(crs, vk, sk, \cdot, \cdot)$ as Okamoto signing algorithm (refer to Section 2.2.5), and define $\mathsf{ver} \overset{\mathsf{def}}{=} \mathtt{verify}(crs, vk, \cdot, \cdot)$ as the corresponding verification algorithm. Then records $\langle \mathsf{sig}, \mathsf{ver}\rangle$ in $history(S)$, and outputs $(\mathtt{VerificationAlg}, sid, \mathsf{ver})$.

Signature generation:

(b) On input $(\mathtt{Sign}, sid, m, \mathsf{ver}')$ where $sid = (S, sid')$, party U sends outgoing message $(\mathtt{GetCRS}, sid)$ to $\mathcal{F}_{\text{CRS}}$ and receives crs; then records $(\mathtt{Sign}, sid, m, \mathsf{ver}', U)$ in $history(U)$; randomly selects $K_1 \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}^2}$ and $\mu_1 \xleftarrow{r} \mathbb{Z}^*_\mathsf{n}$, and computes $C_1 \leftarrow \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$; randomly selects $t, s \xleftarrow{r} \mathbb{Z}_p$, and computes $W \leftarrow g_1^{mt} u_1^t v_1^{st}$; records $\langle K_1, \mu_1, t, s\rangle$ in $history(U)$, then sends outgoing message $(\mathtt{user}_1, sid, C_1, W)$ to S.

(c) On the incoming $\mathtt{user}_1$ message, party S randomly selects $K_2 \xleftarrow{r} \mathbb{Z}^*_{\mathsf{n}^2}$, then sends outgoing message $(\mathtt{signer}_1, sid, K_2)$ to party U.

(d) On the incoming $\mathtt{signer}_1$ message, party U operates as follows:

$K \leftarrow K_1K_2 \bmod \mathsf{n}^2$;
$\theta \leftarrow (mt \bmod p) + t2^\kappa + (st \bmod p)2^{2\kappa}$; $A_\theta, B_\theta \xleftarrow{r} \mathbb{Z}^*_\mathsf{n}$; $E_\theta \leftarrow K^\theta(A_\theta)^\mathsf{n} \bmod \mathsf{n}^2$;
$a_i \xleftarrow{r} \pm[0, 2^{\lambda_0+\lambda_U+\ell_p}]$, $i = 1, 2, 3$; $\vartheta \leftarrow a_1 + a_2 2^\kappa + a_3 2^{2\kappa}$;
$\widehat{E}_\theta \leftarrow K^\vartheta(B_\theta)^\mathsf{n} \bmod \mathsf{n}^2$; $\widehat{W} \leftarrow g_1^{a_1} u_1^{a_2} v_1^{a_3}$;
$\mu_2 \xleftarrow{r} \mathbb{Z}_Q$; $\omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$; $C_2 \leftarrow \mathbf{g}^{\omega_2}\mathbf{h}_2^{\mu_2}$;

Then party U records $\langle A_\theta, B_\theta, a_1, a_2, a_3, \mu_2\rangle$, and sends outgoing message $(\mathtt{user}_2, sid, \langle K_1, \mu_1\rangle, \langle E_\theta, C_2\rangle)$ to party S.

(e) On the incoming $\mathtt{user}_2$ message, party S operates as follows:

$C_1 \overset{?}{=} \mathsf{K}^{K_1}(\mu_1)^{\mathsf{n}^2} \bmod \mathsf{n}^3$; $d_U \xleftarrow{r} \{0,1\}^{\lambda_U}$;
$r \xleftarrow{r} \mathbb{Z}_p$ s.t. $x + r \not\equiv 0 \bmod p$; $\chi \xleftarrow{r} \pm[0, 2^{\lambda_0+\lambda_S+\ell_p}]$; $A_x, B_x \xleftarrow{r} \mathbb{Z}^*_\mathsf{n}$; $l \xleftarrow{r} \mathbb{Z}_p$;
$Y \leftarrow (Wv_1^l)^{\frac{1}{x+r}}$; $\widehat{Z} \leftarrow Y^\chi$; $\widehat{X} \leftarrow g_2^\chi$; $E_x \leftarrow \mathsf{g}^x(A_x)^\mathsf{n} \bmod \mathsf{n}^2$; $\widehat{E}_x \leftarrow \mathsf{g}^\chi(B_x)^\mathsf{n} \bmod \mathsf{n}^2$;
$\mu_3 \xleftarrow{r} \mathbb{Z}_Q$; $\omega_3 \leftarrow \mathcal{H}(Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x)$; $C_3 \leftarrow \mathbf{g}^{\omega_3}\mathbf{h}_3^{\mu_3}$;

Then records $\langle r, \chi, A_x, B_x, l\rangle$ with PID U in $history(S)$, and sends outgoing message $(\mathtt{signer}_2, sid, \langle E_x, C_3\rangle, d_U)$ to party U.

(f) On the incoming $\mathtt{signer}_2$ message, party U operates as follows:

$b_1 \leftarrow a_1 + d_U \cdot (mt \bmod p)$; $\quad b_2 \leftarrow a_2 + d_U \cdot t$; $\quad b_3 \leftarrow a_3 + d_U \cdot (st \bmod p)$;
$F_\theta \leftarrow B_\theta(A_\theta)^{d_U} \bmod \mathsf{n}$; $\quad d_S \xleftarrow{\mathtt{r}} \{0,1\}^{\lambda_S}$;

Then sends outgoing message $(\mathtt{user}_3, sid, \langle b_1, b_2, b_3, F_\theta \rangle, \langle \widehat{E}_\theta, \widehat{W}, \mu_2 \rangle, d_S)$ to party S.

(g) On the incoming $\mathtt{user}_3$ message, party S operates as follows:
$\omega_2 \leftarrow \mathcal{H}(\widehat{E}_\theta, \widehat{W})$; $C_2 =^? \mathbf{g}^{\omega_2}\mathbf{h}_2^{\mu_2}$;
$E_\theta \in^? \mathbb{Z}^*_{\mathsf{n}^2}$, $W \in^? \mathbb{G}_1$; $b_i \in^? \pm[0, 2^{\lambda_0+\lambda_U+\ell_p+1}]$, $i = 1,2,3$; $\quad g_1^{b_1} u_1^{b_2} v_1^{b_3} =^? \widehat{W} W^{d_U}$;
$K \leftarrow K_1 K_2 \bmod \mathsf{n}^2$; $\eta \leftarrow b_1 + b_2 2^\kappa + b_3 2^{2\kappa}$; $K^\eta (F_\theta)^{\mathsf{n}} =^? \widehat{E}_\theta (E_\theta)^{d_U} \bmod \mathsf{n}^2$;
$\delta_x \leftarrow \chi + d_S \cdot x$; $\quad F_x \leftarrow B_x(A_x)^{d_S} \bmod \mathsf{n}$;

Then sends outgoing message $(\mathtt{signer}_3, sid, \langle \delta_x, F_x \rangle, \langle Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x, \mu_3 \rangle)$ to party U, and outputs $(\mathtt{Signature}, sid, U, \mathsf{completed})$ to $\mathcal{Z}$; If one of the above checks fails, outputs $(\mathtt{Signature}, sid, U, \perp)$ to $\mathcal{Z}$.

(h) On the incoming $\mathtt{signer}_3$ message, party U operates as follows:
$\omega_3 \leftarrow \mathcal{H}(Y, l, r, \widehat{Z}, \widehat{X}, \widehat{E}_x)$; $C_3 =^? \mathbf{g}^{\omega_3}\mathbf{h}_3^{\mu_3}$;
$E_x \in^? \mathbb{Z}^*_{\mathsf{n}^2}$; $\delta_x \in^? \pm[0, 2^{\lambda_0+\lambda_S+\ell_p+1}]$; $g_2^{\delta_x} =^? \widehat{X} X^{d_S}$; $\mathsf{g}^{\delta_x}(F_x)^{\mathsf{n}} =^? \widehat{E}_x (E_x)^{d_S} \bmod \mathsf{n}^2$; $Y^{\delta_x} =^? \widehat{Z}(W v_1^l Y^{-r})^{d_S}$;
$f, h \xleftarrow{\mathtt{r}} \mathbb{Z}_p$; $\varsigma \leftarrow Y^{\frac{1}{ft} \bmod p}$; $\alpha \leftarrow X^f g_2^{fr}$; $\beta \leftarrow s + \frac{l}{t} \bmod p$; $V_1 \leftarrow \psi(X)^{\frac{1}{f}} g_1^h$; $V_2 \leftarrow X^{fh+r} g_2^{frh}$;

Now lets $\sigma \leftarrow \langle \varsigma, \alpha, \beta, V_1, V_2 \rangle$, and if $\mathsf{ver}'(m, \sigma) = 1$, outputs $(\mathtt{Signature}, sid, m, \sigma)$, and records $\langle f, h \rangle$ in $history(U)$; else outputs $(\mathtt{Signature}, sid, \perp)$ to $\mathcal{Z}$.

Signature verification:
(i) On input $(\mathtt{Verify}, sid, m, \sigma, \mathsf{ver}')$, party V outputs $(\mathtt{Verified}, sid, m, \mathsf{ver}'(m, \sigma))$, and records the current history in $history(V)$.

Corruption:
(j) On input $(\mathtt{Corrupt}, sid, J)$, party J outputs $(\mathtt{Corrupted}, sid, J, history(J))$.

6.3 Efficiency

Choice of parameter lengths. Define the length of each parameter p, n, Q is ℓ_p, ℓ_{n}, ℓ_Q respectively and should be selected so that the following are satisfied: (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 discrete-logarithm (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. In this section, we count how many communication bits we need to generate a UC blind signature. We count the six flows 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.

6.4 Security

Based on Theorem 5.2 in the previous section and two corollaries, Corollary 6.1 and Corollary 6.2, in this section, we can obtain

Corollary 6.3. *Under the DCR assumption, the DLOG assumption, and the 2SDH assumption, and assuming existence of collision resistent hash function, the blind signature protocol $\pi_{\Sigma(\mathrm{BSIG})}$ in Figure 26 and Figure 27 securely realizes $\mathcal{F}_{\mathrm{BSIG}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model.*

Remark 6.4. The proof of Corollary 6.3 is very similar to the proof of Theorem 5.2. The only difference is that we instantiate the underlying equivocal lite blind signature and the ZK protocols with concrete protocols.

Here we calculate the distance introduced if we replace $\mathcal{F}_{\mathrm{BSIG}}$ with the blind signature protocol $\pi_{\Sigma(\mathrm{BSIG})}$ in Figure 26 and Figure 27 in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model.

Let q_1 be the number of the users which are corrupted initially; q_2 be the number of valid message-signature pairs obtained from a corrupted signer; q_3 be the number of the users which are not corrupted initially. Recall that the proof of Corollary 6.3 can follow the proof of Theorem 5.2 step by step.

- In the first step in the proof, we just restate the whole execution in the CRS hybrid world. So there is no distance introduced. Note that now CRS includes the CRS for the underlying equivocal lite blind signature and the CRS for SVZK and SPZK protocols.

- In the second step in the proof, consider that we instantiate $\mathcal{F}_{\mathrm{SVZK}}$ with a concrete SVZK protocol in the CRS model. When a user is corrupted initially, the simulator needs to extract m which is used by the corrupted user, and "patch" it into $\mathcal{F}_{\mathrm{BSIG}}$. Note that now we cannot patch such m perfectly. The probability that the patched m does not equal the m for the signature includes two parts:

  - due to the underlying three move proof based on Pedersen commitments, which is $\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\lambda_U}$; here $\mathsf{Adv}_{\mathrm{dlog}}$ is the DLOG advantage over the group which Pedersen commitment based, and $\mathsf{Adv}_{\mathrm{crhf}}$ is the advantage for collision-resistant hash function $\mathcal{H}$; and $2^{-\lambda_U}$ is the soundness error.
  - due to the fact that the UC commitment key, i.e. K is an E-key and m cannot be extracted, which is $2^{-\ell_{\mathsf{n}}}$;

  So for each user who is corrupted initially, the distance is $\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\ell_{\mathsf{n}}} + 2^{-\lambda_U}$.

  Together in this step, the distance introduced is $q_1 \cdot (\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\ell_{\mathsf{n}}} + 2^{-\lambda_U})$.

- In the third step we introduce the unforgeability distance:

  - the underlying equivocal lite blind signature will introduce distance $\mathsf{Adv}_{\mathrm{2sdh}}$ which is the 2SDH advantage over bilinear groups $(\mathbb{G}_1, \mathbb{G}_2)$; note that we instantiate the equivocal lite blind signature with the one in Figure 8 whose lite-unforgability is based on the 2SDH assumption.
  - still once the signing secret is revealed, the adversary of course can forge signatures; we instantiate $\mathcal{F}_{\mathrm{SPZK}}$ with a concrete SPZK protocol which introduce distance $\mathsf{Adv}_{\mathrm{dcr}} + 2^{-\lambda_0}$; here $\mathsf{Adv}_{\mathrm{dcr}}$ is the DCR advantage because we use Paillier encryption to encrypt the signing secret; and $2^{-\lambda_0}$ is the zero-knowledge distance.

So together in this step, the distance introduced is $\mathsf{Adv}_{\mathrm{2sdh}} + (q_1 + q_3) \cdot (\mathsf{Adv}_{\mathrm{dcr}} + 2^{-\lambda_0})$.

- Similar to the second step, in the fourth step, consider we instantiate $\mathcal{F}_{\mathrm{SPZK}}$ with a concrete SPZK protocol. When signer is corrupted at the some point, for every user request, the simulator needs to extract the signing secret x and patch it into $\mathcal{F}_{\mathrm{BSIG}}$. Note that now such patching is not perfect. The probability that the patched x does not equal the real one includes only one part: due to the three move proof based on Pedersen commitments, which is $\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\lambda_S}$. Note that this is slightly different from the second step where UC commitment is used; here only Paillier encryption which is an extractable commitment is used, and x can always be extracted. So this step introduces distance $q_2 \cdot (\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\lambda_S})$.

- In the last step, we need to equivocate users who are not corrupted initially. The equivocation includes equivocating the underlying lite blind signature and reconstruct the internals for the SVZK protocol. Consider that the underlying equivocal lite blind signature is unconditionally equivocal, which will not introduce any distance. Note that under the DCR assumption, with the equivocal trapdoor, we can construct the internal state for the SVZK protocol; this introduces distance $\mathsf{Adv}_{\mathrm{dcr}}$. Note also that once the SVZK protocol is not perfectly zero-knowledge, we will not equivocate the underlying lite blind signature and the SVZK protocol perfectly; this introduces distance $\mathsf{Adv}_{\mathrm{dcr}} + 3 \cdot 2^{-\lambda_0}$, where $\mathsf{Adv}_{\mathrm{dcr}}$ is the DCR advantage because we use the Damgård-Jurik encryption to encrypt m, and $3 \cdot 2^{-\lambda_0}$ is the zero-knowledge distance.

  So this step introduces distance $q_3 \cdot (2 \cdot \mathsf{Adv}_{\mathrm{dcr}} + 3 \cdot 2^{-\lambda_0})$.

So the whole distance is:

$$\begin{aligned}&\big(q_1 \cdot (\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\ell_{\mathsf{n}}} + 2^{-\lambda_U})\big)\\ &+ \big(\mathsf{Adv}_{\mathrm{2sdh}} + (q_1 + q_3) \cdot (\mathsf{Adv}_{\mathrm{dcr}} + 2^{-\lambda_0})\big)\\ &+ \big(q_2 \cdot (\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}} + 2^{-\lambda_S})\big) + \big(q_3 \cdot (2 \cdot \mathsf{Adv}_{\mathrm{dcr}} + 3 \cdot 2^{-\lambda_0})\big)\end{aligned}$$

which is $\mathsf{Adv}_{\mathrm{2sdh}} + (q_1 + 3q_3) \cdot \mathsf{Adv}_{\mathrm{dcr}} + (q_1 + q_2) \cdot (\mathsf{Adv}_{\mathrm{dlog}} + \mathsf{Adv}_{\mathrm{crhf}}) + q_1 \cdot 2^{-\ell_{\mathsf{n}}} + q_1 \cdot 2^{-\lambda_U} + q_2 \cdot 2^{-\lambda_S} + (q_1 + 4q_3) \cdot 2^{-\lambda_0}$.

Acknowledgments

We would like to thank Jesper Nielsen for helpful discussions.

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.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[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.

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

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

[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.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. 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`.

[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.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, pages 644–654, 1976.

[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.

[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`.

[HKKL07] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In *TCC 2007*, 2007. To appear. Available at `http://www.cs.biu.ac.il/~lindell/PAPERS/blind_sigs-TCC07.ps`.

[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/.

[KZ07] 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.

[LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *Selected Areas in Cryptography 1999*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.

[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.

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

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.