Triply Adaptive UC NIZK

Ran Canetti[*][1], Pratik Sarkar[**][1], and Xiao Wang[***][2]

[1] Boston University
[2] Northwestern University

Non-interactive zero knowledge (NIZK) enables proving the validity of NP statement without leaking anything else. We study multi-instance NIZKs in the common reference string (CRS) model, against an adversary that adaptively corrupts parties and chooses statements to be proven. We construct the first such *triply adaptive* NIZK that provides full adaptive soundness, as well as adaptive zero-knowledge, assuming either LWE or else LPN and DDH (previous constructions rely on non-falsifiable knowledge assumptions). In addition, our NIZKs are universally composable (UC). Along the way, we:

- Formulate an ideal functionality, $\mathcal{F}_{\mathsf{NICOM}}$, which essentially captures *non-interactive* commitments, and show that it is realizable by existing protocols using standard assumptions.
- Define and realize, under standard assumptions, Sigma protocols which satisfy triply adaptive security with access to $\mathcal{F}_{\mathsf{NICOM}}$.
- Use the Fiat-Shamir transform, instantiated with correlation intractable hash functions, to compile a Sigma protocol with triply adaptive security with access to $\mathcal{F}_{\mathsf{NICOM}}$ into a triply adaptive UC-NIZK argument in the CRS model with access to $\mathcal{F}_{\mathsf{NICOM}}$, assuming LWE (or else LPN and DDH).
- Use the UC theorem to obtain UC-NIZK in the CRS model.

Table of Contents

1 Introduction

Non-Interactive zero knowledge [BFM90, BSMP91] is a magical primitive: with the help of a trusted reference string, it allows parties to publicly assert knowledge of sensitive data and prove statements regarding the data while keeping the data itself secret. Proofs are written once and for all, to be inspected and verified by anyone at any time.

However, harnessing this magic in a concrete and realizable set of security requirements has turned out to be non trivial. A first thrust provides basic formulations of soundness and zero knowledge in the presence of a reference string, and constructions that satisfy them under standard assumptions [BSMP91, FLS99, GR13]. Indeed, even these basic requirements turn out to be non-trivial to formulate and obtain, especially in the case of multiple proofs that use the same reference string and where the inputs and witnesses are chosen adversarially in an adaptive way.

A second thrust addresses malleability attacks [SCO$^+$01, DDN91], and more generally universally composable security [CLOS02] in a multi-party setting. In particular, UC NIZK has been used as a mainstay for incorporating NIZK proofs in cryptographic protocols and systems - actively secure MPC [GMW87], CCA secure encryption [NY90, DDN91], signatures [BMW03, BKM06] and cryptocurrencies [BCG$^+$14].

A third thrust is to construct NIZK protocols that are secure in a multi-party setting where the adversary can corrupt parties adaptively [CLOS02, AMPS21, CGPS21], as the computation proceeds. Here the traditional definition (which requires that the attacker does not gain any advantage towards breaking the security of the overall system beyond the ideal case where the NIZK is replaced by a trusted party) is extended to the case where the attacker obtains the hidden internal state of some provers *after* the proof was sent. Indeed, this extended guarantee is essential whenever NIZK is used as a primitive within larger protocols that purport to obtain security against adaptive corruptions[3].

The first protocol that provides security against adaptive corruptions is that of Groth, Ostrovsky, and Sahai [GOS06, GOS12] (GOS). That protocol is also UC secure, even in a multi-proof, multi-party setting. However it only guarantees *culpable soundness*, namely that the sequence of instances proven to be in a language L during an execution of the protocol is indistinguishable (given the reference string) from a sequence of instances that are actually in L. The works of [KNYY19, KNYY20] have similar characteristics: they provide security against adaptive corruptions, but only culpable adaptive soundness.

Abe and Fehr [AF07] show how to prove full adaptive soundness of a variant of the GOS protocol, under a knowledge-of-exponent (KOE) assumption[4]. However, their analysis is incompatible with UC security [KZM$^+$15], KOE-style assumptions require existence of a knowledge extractor that has full access to whose code is larger than the code of the environment. In contrast, in the UC framework a single extractor/simulator would have to handle arbitrary poly-time environments. The recent work of [KKK21] investigated composable security for knowledge assumptions in the generic group model. They rule out general composition but demonstrate that it is possible under restricted settings. We refer to their paper for more details. Proving composable security of [AF07] in their model is still an open question.

---

[3] In cases where the prover is able to immediately erase all records of its sensitive state - specifically the witness and randomness used in generating the proof - adaptive security is easy to obtain. However such immediate and complete erasure of local state is not always practical.

[4] [AF07] provides adaptive soundness and adaptive zero knowledge and claims security against adaptive corruptions in Remark 11 of their paper.

We are thus left with the following natural question: Can we have triply adaptive NIZK protocols, namely full-fledged UC NIZK protocols in the multi-party, multi-proof setting, in the case of adaptive corruptions without erasures, and with full adaptive soundness? And if so, under what assumptions?

1.1 Our Contributions

We develop a general methodology for obtaining triply adaptive NIZKs, namely UC NIZKs with full adaptive soundness, withstanding adaptive corruptions with no erasures. Using this methodology, we obtain triply adaptive NIZK protocols from statically secure Sigma protocols. The NIZK protocols reuse a single crs for multiple NIZK instances between different pairs of parties. Moreover, one of the NIZK protocols also avoids expensive Karp reductions. Upon concrete instantiation based on either Learning With Errors, or Decisional Diffie Hellman plus Learning Parity with Noise assumption, we obtain the following result:

Theorem 1. *(Informal) Assuming either 1) LWE assumption holds or 2) both DDH and LPN assumptions hold, there exists a multi-theorem NIZK protocol that UC-securely implements the NIZK functionality(Fig. 3) against adaptive corruptions in the **crs** model for multiple instances. Furthermore, it is adaptively sound and adaptively zero knowledge.*

As an independent result we also obtain a compiler that (assuming either LWE or DDH) transforms a given NIZK protocol, where the length of the crs can depend on the NP relation to be asserted, to a NIZK protocol where the length of the crs depends only on the security parameter. Furthermore, we do so while preserving triple adaptive security. Previous such compilers [GGI+15, CsW19] were known only from LWE:

Theorem 2. *(Informal) Assuming either 1) LWE assumption holds or 2) both DDH and LPN assumptions hold, there exists a multi-theorem NIZK protocol that UC-securely implements the NIZK functionality(Fig. 3) against adaptive corruptions with short **crs** (i.e. $|crs| = poly(\kappa)$ and κ is the computational security parameter) for multiple instances. Furthermore, it is adaptively sound and adaptively zero knowledge.*

Furthermore, by plugging our NIZK protocol in the compiler of [CsW19] we can obtain a triply adaptive NIZK protocol from LWE, where the reference string size depends only on the security parameter and the proof size depends on the witness size and the security parameter. We compare the state-of-the-art results and our result in Tab. 1.

1.2 Our Techniques

We present our NIZK compiler in Fig. 1. Our approach follows the general paradigm of applying the Fiat-Shamir transform (instantiated via correlation in tractable hash functions) to Sigma protocols, as developed in [CGH98, HL18, CCH+19, PS19, BKM20, HLR21]. However, to preserve triple adaptivity the transform should be applied with some care.

Starting Point. Let us briefly recall the definition of a Sigma protocol: A Sigma protocol is a 3 round protocol for proving validity of an NP statement $x \in \mathcal{L}$ (where $\mathcal{L}$ is the language) using the knowledge of an accepting witness w. The prover sends the first message a, the verifier samples a random challenge c, and based on the challenge $c \in \mathcal{C}$ the prover computes

2

Protocol	Adaptive Soundness	Adaptive Zero-Knowledge	Security against Adaptive Corruptions	Assumptions	UC-Secure
[GOS12]	×	✓	✓	Pairings	✓
[KNYY19], [KNYY20]	×	✓	✓	Pairings	✓
CI-based NIZKs	✓	✓	×	LWE/DDH+LPN	✓
[AF07]	✓	✓	✓	Knowledge of Exponent	×
This work	✓	✓	✓	LWE/DDH+LPN	✓

Table 1. Comparison of Triply Adaptive NIZK Protocols.

the response z. The verifier accepts an honest proof when $x \in \mathcal{L}$. Soundness ensures that the verifier rejects cheating proofs with $\frac{1}{|\mathcal{C}|}$ probability. Honest verifier zero knowledge (HVZK) ensures that the simulator constructs an honest proof given a random challenge c and the simulated proof is indistinguishable from an honest proof. However, the usual Sigma protocols [FLS99, Blu86] are only secure against static corruption of prover, i.e. upon post-execution corruption of prover the HVZK simulator obtains witness w and is unable to provide randomness such that it is consistent with the proof (a, c, z) constructed by the HVZK simulator.

New UC-Commitment Functionality $\mathcal{F}_{\mathsf{NICOM}}$. To solve the above issue in a modular fashion, we first introduce a new non-interactive UC commitment functionality, $\mathcal{F}_{\mathsf{NICOM}}$, that enables modular analyzing of NIZK protocols that use commitments as an underlying primitive. Specifically, the new functionality returns a commitment string and a decommitment to the committer as an output of the commit phase, where the committer commits to a message. The open phase allows non-interactive verification of the commitment, decommitment and message tuple by a verifier. Moreover, the functionality is provided with an explicit simulation algorithm $\mathcal{S}_C$ which extracts committed messages from maliciously generated commitments and permits equivocation of simulated commitments. Looking ahead, the CI-hash function would be equipped with the $\mathcal{S}_C$ algorithm to run the bad challenge function and yet we would argue security of the NIZK protocol in the $\mathcal{F}_{\mathsf{NICOM}}$ model. Hence, $\mathcal{F}_{\mathsf{NICOM}}$ provides a cleaner abstraction of non-interactive UC commitments. The formal description of the $\mathcal{F}_{\mathsf{NICOM}}$ functionality can be found in Fig. 2.

Strengthening Sigma protocols in $\mathcal{F}_{\mathsf{NICOM}}$ model. Now, we define the notion of an *adaptively secure Sigma protocol* in the $\mathcal{F}_{\mathsf{NICOM}}$ model as a stepping stone towards security against adaptive corruptions. These are Sigma protocols which provide security against adaptive corruption of prover in the $\mathcal{F}_{\mathsf{NICOM}}$ model. To attain constructions of such Sigma protocols, we replace the underlying commitment scheme in the *commit-and-open* protocols of [Blu86, FLS99, HV16] with $\mathcal{F}_{\mathsf{NICOM}}$. Then we prove that these Sigma protocols are adaptively secure in the $\mathcal{F}_{\mathsf{NICOM}}$ model, while preserving special soundness. Furthermore, these protocols satisfies full adaptive soundness and provides adaptive ZK in the $\mathcal{F}_{\mathsf{NICOM}}$ model. If $\mathcal{F}_{\mathsf{NICOM}}$ is concretely instantiated using
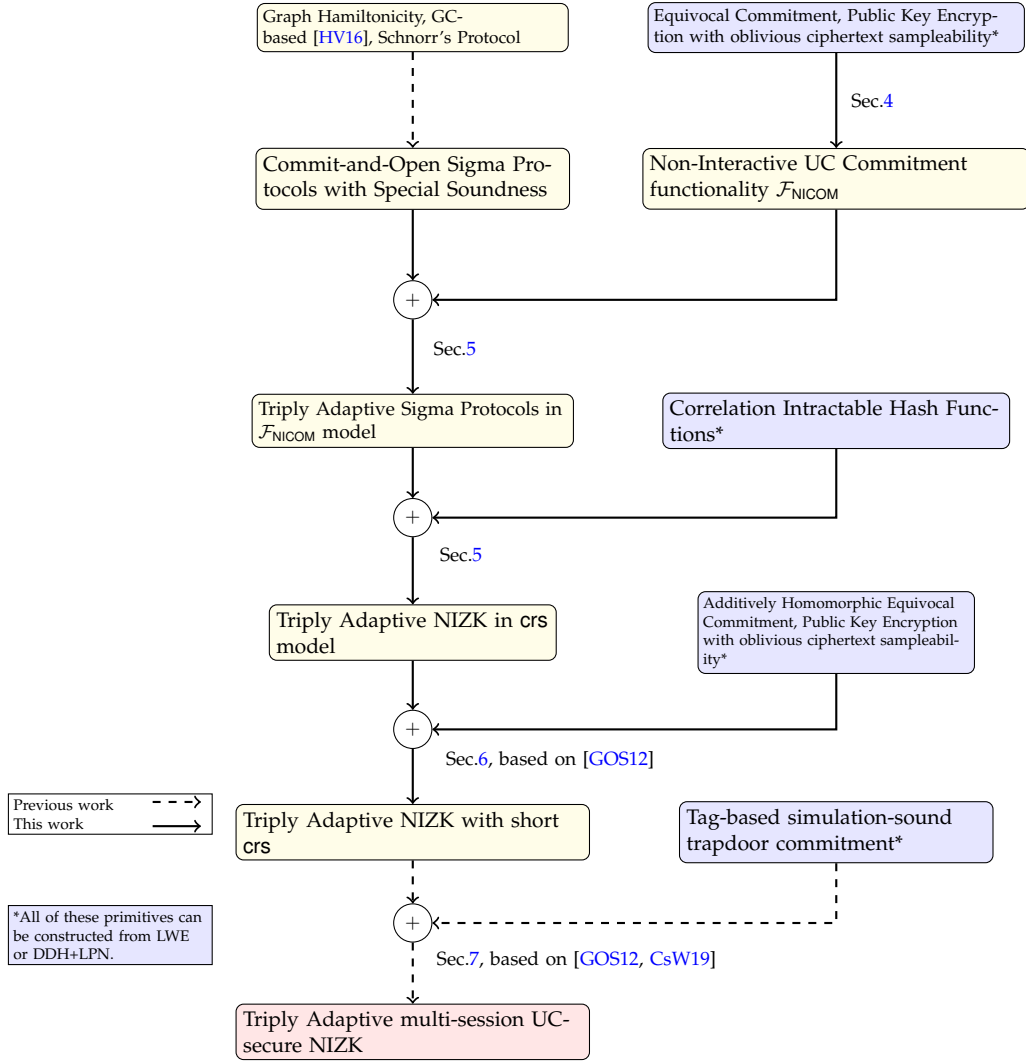
Fig. 1. An overview of our NIZK results

an adaptively secure non-interactive commitment in non-programmable crs model [5] then the protocol also preserves full adaptive soundness and adaptive ZK.

Removing Interaction. It is now tempting to apply the Fiat-Shamir (FS) transform [FS87] using correlation intractable hash functions (due to [CGH98, HL18, CCH+19, PS19, BKM20, HLR21]), and conclude that the resulting protocol is a NIZK. However, it is not clear how the transform would actually work: the bad challenge function for the CI hash function cannot be defined given blackbox access to a and the challenge space can be exponentially large, for example consider Schnorr's protocol [Sch90]. The current CI-based NIZKs [CGH98, HL18, CCH+19, PS19] consider specific Sigma protocols to construct NIZKs. We take a different route to solve this problem by relying on special soundness property. Special soundness property of a Sigma protocol ensures that given two accepting transcripts (a, c_0, z_0) and (a, c_1, z_1) for different challenges $c_0 \neq c_1$ there exists an extractor which extracts a valid witness from the transcripts. If

---

[5] The crs distribution in the real world is statistically close to the crs distribution in the ideal world

the statement $x \notin \mathcal{L}$ is not in the language then the prover cannot construct two such accepting transcripts for the same a.

We generalize the framework of [CD00] to construct our compiler. In our compiler, the prover computes a, samples two challenges c_0 and c_1, computes responses z_0 and z_1 and commits to (c_0, z_0) and (c_1, z_1) in $\mathcal{F}_{\mathsf{NICOM}}$ model. This step is repeated for $\tau = \mathcal{O}(\kappa)$ times, where κ is the security parameter. Let $\mathbf{Y}$ denote the commitments to (c_0, c_1, z_0, z_1) for the τ iterations. The CI-hash function is defined in the statistical mode equipped with the extraction algorithm $\mathcal{S}_C$ for $\mathcal{F}_{\mathsf{NICOM}}$. The hash function is CI for the bad challenge function - for each iteration (a, c_0, c_1, z_0, z_1) it outputs 0 if (a, c_0, z_0) is accepting. The prover invokes the CI hash function on $(a, \mathbf{Y})$ to obtain a challenge bit e for each iteration. For each iteration, the prover computes the response as the decommitment to (c_0, c_1, z_e). Special soundness of the Sigma protocol ensures that a malicious prover is unable to compute two such valid transcripts (a, c_0, z_0) and (a, c_1, z_1) for a false statement $x \notin \mathcal{L}$.

CI-based NIZK Transformations for Arguments. Now we would like to apply the analysis of [CCH$^+$19] to argue soundness of the NIZK protocol, which says that if the malicious prover is able to construct an accepting proof for $x \notin \mathcal{L}$ then it breaks correlation intractability. However, now we are faced with another barrier: The [CCH$^+$19] analysis for CI crucially needs the underlying Sigma protocol to be statistically sound. In contrast, our Sigma protocols are only computationally sound since it relies on the special soundness property (which can be computational) of the Sigma protocol and the computational binding property of the commitment scheme. Furthermore, this is inherent: Statistically sound ZK protocols cannot possibly be secure against adaptive corruptions. In particular, this means that we cannot "switch the crs in the hybrids to make the sigma protocol statistically sound": As soon as we do so, the protocol (in that hybrid) stops being secure against adaptive corruptions.

We get around this barrier[6] as follows: with each commitment made during the interaction we can associate an event B, determined at the time of commitment, such that: (a) event B can be shown to occur only with negligible probability, and (b) conditioned on event B not occurring, the commitment is statistically binding. Event B is the event where the adversary successfully evades the extraction algorithm $\mathcal{S}_C$ of $\mathcal{F}_{\mathsf{NICOM}}$ and yet the corresponding decommitment is accepted. Given that event B does not occur, we then associate an event D with each of the τ adaptively secure Sigma protocol executions, such that: (a) event D can be shown to occur only with negligible probability, and (b) conditioned on event D not occurring, the Sigma protocol is statistically sound. The event D is the event where the adversary breaks special soundness property of the Sigma protocol. The [CCH$^+$19] analysis can now be resurrected, conditioned on event B not occurring for any of the commitments made, and event D not occurring for the Sigma protocols. Initializing the hash function in the statistical mode ensures that soundness of the protocol is reduced to breaking statistical correlation intractability of the hash function, provided event B and event D does not occur.

Adaptive soundness of our protocol follows in a straightforward way from the fact that the entire proof is performed without changing the distribution of the crs in the $\mathcal{F}_{\mathsf{NICOM}}$ model. (Indeed, this important feature allows us to avoid the main obstacle that prevents the [GOS12] protocol from being adaptively sound.) Adaptive zero knowledge follows from the adaptive security of the Sigma protocol in the $\mathcal{F}_{\mathsf{NICOM}}$ model. If $\mathcal{F}_{\mathsf{NICOM}}$ is concretely instantiated using

---

[6] The recent work of [CJJ21] also applied the Fiat-Shamir paradigm on an interactive protocol which is not statistically sound using CI hash functions. However, their protocol is not adaptively sound. Meanwhile, the plain-model sigma protocol that [CCH$^+$19] start from is statistically sound.)

an adaptively secure non-interactive commitment in non-programmable crs model [7] then the protocol also preserves full adaptive soundness and adaptive ZK.

Instantiations of Adaptively Secure Sigma protocols in $\mathcal{F}_{\mathsf{NICOM}}$ model. We show that a wide variety of Sigma protocols satisfy (in $\mathcal{F}_{\mathsf{NICOM}}$ model) adaptive security with special soundness and adaptive soundness - Schnorr's protocol, Sigma protocol of [FLS99] (FLS), Blum's Hamiltonicity protocol and garbled circuit (GC) based protocol of [HV16]. Furthermore, the GC based protocol avoids expensive Karp reduction.

Instantiating the CI-hash and $\mathcal{F}_{\mathsf{NICOM}}$. The CI function can be instantiated from LWE [PS19], or it can be replaced by a CI-Approx [BKM20] function based on LPN+DDH. $\mathcal{F}_{\mathsf{NICOM}}$ is instantiated using the protocol of [CF01] from equivocal commitments and CCA-2 secure public key encryption with oblivious ciphertext sampling property in the non-programmable crs model.

Reducing crs size. By applying techniques from GOS, we obtain a compiler which *reduces the crs size* of a NIZK argument. Assuming reusable non-interactive equivocal commitments with additive homomorphism and PKE (with oblivious ciphertext sampleability) we compile any triply adaptive NIZK argument with a long multi-proof crs, i.e. $|\mathsf{crs}| = \mathrm{poly}(\kappa, |\mathcal{C}|)$ to obtain a triply adaptive NIZK argument with a short multi-proof common reference string scrs, where $|\mathsf{scrs}| = \mathrm{poly}(\kappa)$, $\mathcal{C}$ is the NP verification circuit and κ is the computational security parameter. The prover commits to each wire value (of the circuit) and proves that they are bit commitments using the NIZK. In addition, the prover applies some homomorphic operation on the input wire and output wire commitments for each gate. If the input and output wire values are consistent with the gate evaluation then the homomorphically evaluated commitment will be a bit commitment. The prover proves this using NIZK for every gate in the circuit. Each NIZK statement is short and depends only on the committer's algorithm ($= \mathrm{poly}(\kappa)$) and not on $|\mathcal{C}|$. As a result the crs size of the NIZK can be short. The commitment can be instantiated from DDH (Pedersen commitment or [CSW20]) or LWE/SIS [GVW15]. The encryption scheme can be instantiated from DDH assumption using Elgamal encryption or LWE [GSW13] assumption.

Obtaining multi-session UC security. We add non-malleability to our NIZK argument using standard techniques from GOS to obtain the multi-session UC-secure NIZK in the short crs model. It relies on a tag-based simulation-sound trapdoor commitment scheme and a strong one-time signature scheme . The tag-based commitment can be instantiated from UC-commitments - DDH [CSW20] and LWE [CsW19]. Strong one-time signatures can be constructed from one-way functions. This transformation also preserves triply adaptive security.

1.3 Related Works

The works of [GOS06, KNYY19, KNYY20] construct NIZKs which are secure against adaptive corruptions but they lack adaptive soundness. The works of [CCH+19, BKM20] construct statically secure NIZKs which attain adaptive soundness and adaptive ZK. A concurrent work by [CPV20] compiled delayed input Sigma protocol into a Sigma protocol which satisfies adaptive zero knowledge. Upon applying the result of [CPS+16] they obtain adaptive soundness. The Fiat-Shamir transform is applied using CI hash function to obtain NIZKs, but they lack

---

[7] The crs distribution in the real world is statistically close to the crs distribution in the ideal world

security against adaptive corruptions. The only work which achieves triple adaptive security is [AF07] based on knowledge assumptions; which is incompatible with the UC framework.

The literature consists of work [GGI+15, CsW19] that make the crs size independent of $|\mathcal{C}|$ but those approaches are instantiatable only from LWE. Whereas, our compiler can be instantiated from non-lattice based assumptions like DDH.

Paper Organization. In Section 2, we present the key intuitions behind our protocols. We introduce some notations and important concepts used in this work in Section 3. This is followed by our triply adaptively-secure NIZK compiler in Section 5. We present our compiler to reduce the crs length in Section 6. Finally, we conclude with our multi-session UC-NIZK protocol in the short crs model in Section 7.

Note. Throughout the paper we refer to *security against adaptive corruptions* as *adaptive security*.

2 Technical Overview

In this section we provide an overview of our protocols. As discussed in the Introduction, a key component in our approach is to break the Fiat-Shamir transformation into two steps: A first step that uses an ideal UC commitment fucntionality, and a second step of instantiating this functionality with an adaptively secure protocol. Validity of the approach would follow from the UC theorem and the special soundness of the sigma protocol.

We first overview the new formulation of ideal UC commitments, $\mathcal{F}_{\mathsf{NICOM}}$, that enables our two-step approach, and argue that known protocols, that UC realize the traditional ([CF01]) formulation of ideal commitment, realize $\mathcal{F}_{\mathsf{NICOM}}$ as well. Next, we overview our notion of fully adaptive Sigma protocols that use $\mathcal{F}_{\mathsf{NICOM}}$, followed by the first step of the Fiat-Shamir transform. We demonstrate that the resulting NIZKs satisfy triply adaptive security in the $\mathcal{F}_{\mathsf{NICOM}}$-hybrid model, and that triple adaptivity is preserved even after replacing $\mathcal{F}_{\mathsf{NICOM}}$ with a protocol that realizes it. Next, we show instantiations of adaptive Sigma protocol. Finally we show how to reduce the crs size of our NIZK protocols to $\mathrm{poly}(\kappa)$ by assuming homomorphic equivocal commitments. Till this point, all our protocols are triply adaptive and single-prover UC-secure. Finally, we make them UC-secure in the general, multi-prover sense by adding non-malleability.

2.1 UC non-interactive commitment functionality $\mathcal{F}_{\mathsf{NICOM}}$

Our new UC-commitment functionality $\mathcal{F}_{\mathsf{NICOM}}$ can be found in Fig. 2. The functionality receives an algorithm $\mathcal{S}_C$ algorithm from the adversary $\mathcal{S}$. When an honest committer P wants to commit to a message m for subsession ssid, the functionality invokes $\mathcal{S}_C$ for a commitment string π and an internal state st. π is independent of the message m. The functionality then invokes $\mathcal{S}_C$ with the message m and the state st to obtain a decommitment d and an updated state st. The functionality stores $(\mathsf{ssid}, \mathsf{P}, m, \pi, d, \mathsf{st})$ and returns the commitment string π and the decommitment d to the committer. The committer sends π as the commitment to message m. An honest committer decommits to a commitment string π' by sending (m', d') to the verifier V. The verifier locally verifies the decommitment by invoking $\mathcal{F}_{\mathsf{NICOM}}$ on the tuple (m', π', d'). The functionality returns verified if the tuple is stored in memory corresponding to the subsession and the same committer P. If the same commitment string π' is stored but with

7

- At first activation, obtain algorithm $\mathcal{S}_C$ from $\mathcal{S}$. Initialize a list $\mathcal{L} = \emptyset$ that would store illegitimate queries which failed to verify.
- **Commit:** On input $(\mathsf{Com}, \mathsf{ssid}, m)$ from committer P:
    - obtain commitment π and internal state st as $(\pi, \mathsf{st}) \leftarrow \mathcal{S}_C(\mathsf{Com}, \mathsf{ssid})$
    - obtain decommitment d and state st as $(d, \mathsf{st}) \leftarrow \mathcal{S}_C(\mathsf{Equiv}, \mathsf{ssid}, \pi, \mathsf{st}, m)$
    - store $(\mathsf{ssid}, m, \pi, d, \mathsf{st})$ and output $(\mathsf{Receipt}, \mathsf{ssid}, \pi, d)$ to P.
- **Open:** On input $(\mathsf{Open}, \mathsf{ssid}, m', \pi', d')$ from verifier V:
    - If $(\mathsf{ssid}, m', \pi', d', \mathsf{st})$ is stored for some st, then return $(\mathsf{verified}, \mathsf{ssid})$ to V.
    - If $(\mathsf{ssid}, m', \pi', d', \mathsf{st}) \in \mathcal{L}$, then return $(\mathsf{verification\text{-}failed}, \mathsf{ssid})$ to V.
    - If $(\mathsf{ssid}, m'', \pi', d'', \mathsf{st})$ is stored, and $m'' \neq m'$ or $d'' \neq d'$ then return $(\mathsf{verification\text{-}failed}, \mathsf{ssid})$ to V, and add $(\mathsf{ssid}, m', \pi', d')$ in list $\mathcal{L} = \mathcal{L} \cup (\mathsf{ssid}, m', \pi', d')$.
    - Else (i.e., no record $(\mathsf{ssid}, ...)$ is stored, or there is a stored record of the form $(\mathsf{ssid}, m'', \pi'', d'', \mathsf{st})$ where $\pi'' \neq \pi'$):
        - Obtain $(m'', \mathsf{st}) \leftarrow \mathcal{S}_C(\mathsf{Ext}, \mathsf{ssid}, \pi')$.
        - If $m'' \neq m'$, set $v = \mathsf{verification\text{-}failed}$.
        - If $m'' = m'$, set $v \leftarrow \mathcal{S}_C(\mathsf{Verify}, \mathsf{ssid}, \pi', d', \mathsf{st})$.
        - If $v == \mathsf{verified}$, then store the tuple $(\mathsf{ssid}, m', \pi', d', \mathsf{st})$ and return (v, ssid) to V. Else return $(\mathsf{verification\text{-}failed}, \mathsf{ssid})$ to V, and add $(\mathsf{ssid}, m', \pi', d')$ in list $\mathcal{L} = \mathcal{L} \cup (\mathsf{ssid}, m', \pi', d')$.
- **Corruption:** When receiving $(\mathsf{Corrupt}, \mathsf{ssid})$ from $\mathcal{S}$, mark ssid as corrupted. Send all the stored tuples of the form $(\mathsf{ssid}, ...)$ to $\mathcal{S}$. If there does not exist any tuple then send $(\mathsf{ssid}, \perp)$ to $\mathcal{S}$.
  On input $(\mathsf{corrupt\text{-}check}, \mathsf{sid}, \mathsf{ssid})$, return whether $(\mathsf{sid}, \mathsf{ssid})$ is marked as corrupted.

different messages/decommitments/committers/ssid then the functionality rejects the opening by sending $\mathsf{verification\text{-}failed}$. Finally, if the commitment string has never been stored in the memory of $\mathcal{F}_{\mathsf{NICOM}}$ then $\mathcal{F}_{\mathsf{NICOM}}$ invokes $\mathcal{S}_C$ to extract a valid message m'' from the commitment string π'. If $m'' == m'$ then the functionality invokes $\mathcal{S}_C$ with the opening (m', π', d') to verify the decommitment. If the decommitment correctly verifies then the functionality stores the tuple in the memory and returns $\mathsf{verified}$ to V. Else, it rejects the decommitment. Once an opening request - $(\mathsf{ssid}, m', \pi', d')$ gets rejected then that request is added to a list $\mathcal{L}$. Next time when the same tuple is requested to be verified $\mathcal{F}_{\mathsf{NICOM}}$ rejects it.

Our model allows a prover to send a commitment that was not computed by invoking the $\mathcal{F}_{\mathsf{NICOM}}$ functionality. Furthermore, access to the $\mathcal{S}_C$ algorithm enables extraction from a maliciously generated commitment and equivocating a simulated commitment. The command $\mathcal{S}_C(\mathsf{Equiv}, \mathsf{ssid}, \mathsf{P}, \pi, \mathsf{st}, m)$ is used to equivocate a commitment string π such that it opens to m. The $\mathcal{S}_C(\mathsf{Ext}, \mathsf{ssid}, \mathsf{P}, \pi)$ command is used to extract a message from the commitment π. These algorithms come in handy for simulation purposes when $\mathcal{F}_{\mathsf{NICOM}}$ is used in bigger protocols.

Implementing $\mathcal{F}_{\mathsf{NICOM}}$. We implement $\mathcal{F}_{\mathsf{NICOM}}$ in Section. 4 using the non-interactive commitment scheme of [CF01] based on equivocal commitments and CCA-2 secure public key encryption with oblivious ciphertext sampleability. The committer P commits to a bit message m as $c = \mathsf{Com}(m; r)$. The commitment randomness is encrypted via a pair of encryptions. The committer encrypts the corresponding randomness r, subsession id ssid and committer id P using a CCA-2 secure PKE as $E_m = \mathsf{Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); s_m)$ with randomness s_m. The other encryption E_{1-m} is obliviously sampled using randomness s_{1-m}. The commitment consists of (c, E_0, E_1) and the opening information is (m, r, s_0, s_1). The verifier performs the canonical verification by reconstructing the commitment. The equivocal commitment can be instantiated from Pedersen Commitment and the obliviously sampleable encryption scheme can be instantiated from Cramer Shoup encryption [CS98], yielding a protocol from DDH. Similarly, we can instantiate the equivocal commitment from LWE [CsW19] and the obliviously sampleable encryption scheme from LWE [MP12].

2.2 Adaptively Secure Sigma Protocols in the $\mathcal{F}_{\mathsf{NICOM}}$ model

We recall the definition of a Sigma protocol and then we introduce the notion of adaptively Sigma protocols in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

Sigma Protocol. A Sigma protocol consists of a prover possessing an NP statement $x \in \mathcal{L}$ (for language $\mathcal{L}$) and witness w which validates the statement. The verifier possesses the statement x. The prover constructs a first message a and the honest verifier challenges the prover with a random challenge $c \leftarrow_R \mathcal{C}$ from the challenge space $\mathcal{C}$. Based on the challenge, the prover computes a response z and sends it to the verifier. Completeness ensures that an honest verifier always accepts the proof (a, c, z). Soundness ensures that the verifier accepts a proof corresponding to an invalid statement $x' \notin \mathcal{L}$ with probability $\frac{1}{|\mathcal{C}|}$. The protocol is repeated κ times to obtain negligible (in κ) soundness error. We also require special soundness which guarantees a witness extractor given two accepting transcripts (a, c, z) and (a, c', z') corresponding to the same first message but different challenges $c \neq c' \in \mathcal{C}$. Finally, we need honest verifier zero knowledge which allows a simulator to simulate an accepting proof given an honestly sampled challenge c. The simulated proof should be indistinguishable from an honestly generated proof.

Limitations of a Sigma protocol. A Sigma protocol does not necessarily guarantee security against adaptive corruptions. The adversary can choose to corrupt the prover after obtaining the simulated proof. In such a case, the simulator obtains the witness and needs to provide prover's randomness such that the simulated proof is consistent with the witness. This problem crops up especially when the first message of the Sigma protocol [FLS99] is statistically binding and doesn't allow equivocation later on. To tackle this issue, we introduce the notion of adaptively secure Sigma protocols in the ideal UC commitment functionality (for multiple subsessions) $\mathcal{F}_{\mathsf{NICOM}}$ model. The traditional UC commitment functionality of [CF01] (Fig. 19) is not compatible with non-interactive commitments since the functionality is required to interact with the parties during Commit and open phases. So we use our new commitment functionality $\mathcal{F}_{\mathsf{NICOM}}$ which allows non-interactive Commit and Open phases.

Adaptively Secure Sigma Protocols. As seen above, the traditional Sigma protocols does not necessarily guarantee security against adaptive corruptions. In the light of this, we consider Sigma protocols in the $\mathcal{F}_{\mathsf{NICOM}}$ model. The prover sends the first message a to the verifier, the verifier sends a random challenge c to the prover and the prover computes the response z based on c. The prover and verifier have access to the $\mathcal{F}_{\mathsf{NICOM}}$ functionality during the protocol execution. In addition to HVZK and special soundness properties, we also require that the simulator is able to produce consistent randomness for a simulated proof and a valid witness when the prover gets corrupted post-execution. Looking ahead, the first message a will consist of commitments that are obtained by invoking $\mathcal{F}_{\mathsf{NICOM}}$ functionality. This enables the simulator to construct an HVZK proof during protocol execution - where it opens a few of the commitments in a which are required for verification. The other commitments in a remain unopened during the protocol. When the prover gets corrupted post-execution, the simulator obtains the witness w, and it equivocates the unopened commitments in a to produce a simulated prover's randomness such that it is indistinguishable from honestly sampled prover randomness (in the real world execution).

We also require special soundness property from our adaptively secure Sigma protocol to construct a NIZK protocol. We say that the protocol satisfies special soundness if there exists

an extractor which extracts the witness given two transcripts (a, c_0, z_0) and (a, c_1, z_1) corresponding to the same a.

2.3 Compiling to an Adaptively-Secure NIZK

Next, we implement the $\mathcal{F}_{\mathsf{NIZK}}$ functionality for a single session by using the Fiat-Shamir transform on $\tau = \mathcal{O}(\kappa)$ iterations of the adaptively secure Sigma protocol. We instantiate the hash function in the Fiat-Shamir Transform using a correlation intractable hash function H [PS19, CCH+19, BKM20].

Correlation Intractability. A correlation intractable hash function H has the following property: For every efficient function f, given a hash function $H \leftarrow \mathcal{H}$ from the hash family $\mathcal{H}$, it is computationally hard to find an x s.t. $f(x) = H(x)$. Based on the first message a of a trapdoor-Sigma Protocol, the Fiat-Shamir challenge $\mathbf{e}$ can be generated using the hash function as $\mathbf{e} = H(\mathbf{a})$. The prover computes the third message $\mathbf{z}$ using $\mathbf{e}$. Trapdoor-Sigma protocol ensures that for every statement not in the language there can be only one bad challenge $\mathbf{e} = g(\mathbf{a})$ s.t. $(\mathbf{a}, \mathbf{e}, \mathbf{z})$ is an accepting transcript. By setting the function $f = g$ as the bad challenge function in H it is ensured that a malicious prover who constructs a bad challenge $\mathbf{e} = H(\mathbf{a})$ can be used to break correlation intractability since $\mathbf{e} = g(\mathbf{a}) = f(\mathbf{a})$. This guarantees soundness of the NIZK protocol.

Protocol. We compile our adaptively secure Sigma protocol into an adaptively secure NIZK in the $\mathcal{F}_{\mathsf{NICOM}}$ model (the $\mathcal{F}_{\mathsf{NICOM}}$ functionality is later instantiated using an adaptively secure non-interactive commitment scheme [CF01]). The prover computes the first message a^j of the adaptively secure Sigma protocol for the jth iteration where $j \in [\tau]$. It samples two challenges c_0^j and c_1^j from the challenge space such that $c_0^j \neq c_1^j$. The prover computes the responses z_0^j and z_1^j corresponding to both challenges c_0^j and c_1^j respectively. The prover commits to the challenges c_0^j and c_1^j, and the responses z_0^j and z_1^j. Let us denote the set of commitments as Y^j. The prover repeats the above protocol for τ iterations. Let $\mathbf{Y} = \{Y^j\}_{j \in [\tau]}$ denote the complete set of commitments and let $\mathbf{a} = \{a^j\}_{j \in [\tau]}$ denote the complete set of first messages. The prover computes the challenge bit-vector $\mathbf{e} = H(\mathsf{k}, (\mathbf{a}, \mathbf{Y}))$ (where k is the hash key) by invoking the hash function on the commitments $\mathbf{Y}$. The hash function is initialized in the statistical mode and the hash key contains the algorithm $\mathcal{S}_C$ obtained from $\mathcal{F}_{\mathsf{NICOM}}$. The hash function internally runs $\mathcal{S}_C$ to extract from the commitments. We capture this using the bad challenge function $\mathcal{C}_{\mathsf{sk}}$ where the hash key k is embedded with the circuit $\mathcal{C}_{\mathsf{sk}}$ as follows:

$$\mathsf{k} = \mathcal{H}.\mathsf{StatGen}(\mathcal{C}_{\mathsf{sk}}), \text{ where}$$

$\mathcal{C}_{\mathsf{sk}}$ is a poly-size circuit that takes $\mathbf{Y}$ as input and $\mathsf{sk} = \mathcal{S}_C$ is the secret algorithm of $\mathcal{F}_{\mathsf{NICOM}}$. $\mathcal{C}_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y})$ is the circuit computing the function $f_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y}) = \mathbf{e}$ s.t. for $j \in [\tau]$, $e^j = 0$ iff (a^j, c_0^j, z_0^j) is an accepting proof where $\mathcal{C}_{\mathsf{sk}}$ extracts the challenges (c_0^j, c_1^j), and the responses (z_0^j, z_1^j) by running $\mathcal{S}_C$. Setting the hash function in the statistical mode ensures that the hash function H is correlation intractable for all relations of the form:

$$\mathcal{R}_{\mathsf{sk}} = \{(\mathbf{a}, \mathbf{Y}, \mathbf{e}) : \mathbf{e} = f_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y})\}$$

In the jth iteration, upon obtaining e as the challenge bit the prover decommits to (c_0^j, c_1^j, z_e^j). The NIZK proof for the jth iteration is $(a^j, c_0^j, c_1^j, z_e^j)$ and the decommitments corresponding

to (c_0^j, c_1^j, z_e^j). The verifier checks that - 1) the decommitments are correct, 2) the challenges are different, i.e. $c_0^j \neq c_1^j$, 3) the proof - (a^j, c_e^j, z_e^j) verifies. The verifier runs the verification protocols for every iteration $j \in [\tau]$. The verifier outputs accept if all the τ proofs verify correctly. Correctness of the protocol follows from the correctness of the commitment scheme and correctness of the sigma protocol.

Soundness and Proof of Knowledge. The soundness and proof of knowledge argument follows through a sequence of hybrids. The correlation intractability does not hold in the real world since we start off with a NIZK argument and not a NIZK proof. The hybrid argument starts off with the real-world protocol in the first hybrid. In the second hybrid, we rely on the binding property and extractability property of the commitment scheme to ensure that the committed messages can be either correctly extracted or the commitment fails to open correctly. In the next hybrid, we rely on the special soundness property of the Sigma protocol to ensure that if for any jth iteration (for $j \in [\tau]$) if the prover constructs an accepting proof for both $e^j = 0$ and $e^j = 1$ then the witness extractor of the sigma protocol correctly extracts the underlying witness. In the final hybrid, if the prover has evaded the witness extractor and yet succeeded in creating an accepting proof then it has predicted the challenge vector $\mathbf{e}$ correctly by breaking the correlation intractability of the hash function. However, we know that there does not exist $\mathbf{e}$ such that the following holds due to statistical correlation intractability and the underlying Sigma protocol in this hybrid is a proof. This ensures that either the witness extractor extracts an accepting witness from atleast one of the iterations or the proof does not verify. This completes our soundness argument.

Adaptive Soundness. This follows along the same lines provided the underlying sigma protocol satisfies adaptive soundness. The distribution of the crs is identical in the real and ideal world and hence we can argue that the proof fails to verify for a statement $x \notin \mathcal{L}$ since there does not exist any valid witness.

Security against Adaptive Corruptions and Adaptive ZK. The ZK property crucially relies on the adaptive security of the Sigma protocol and security against adaptive corruptions of the commitment scheme. The ZK simulator of the NIZK protocol invokes the HVZK simulator the sigma protocol to obtain a simulated proof (a^j, c^j, z^j) corresponding to a random ZK challenge c^j for the jth iteration. The simulator constructs the commitments $\mathbf{Y}$ in the equivocal mode and invokes the hash function to obtain the challenge string $\mathbf{e}$. Upon obtaining the challenge bits e^j (for $j \in [\tau]$) the simulator opens the commitments corresponding to e^j to the simulated proof (a^j, z^j, c^j). It also equivocates the commitment for the ZK challenge corresponding to bit $1 - e^j$ to open to a different challenge $c^{j'}$ as part of the protocol. The proof verifies correctly due to the HVZK property of the Sigma protocol and equivocal property of the commitment scheme. Upon post-execution corruption of the prover, the NIZK simulator obtains the correct witness w and it invokes the simulator of the adaptively secure Sigma protocol with w to obtain the internal prover state. Using these information the NIZK simulator constructs the response corresponding to challenge $c^{j'}$ for choice bit $1 - e^j$. The simulator equivocates the commitments in $\mathbf{Y}$ (mainly the commitment to the jth response for challenge bit $1 - e^j$) such that the proofs corresponding to challenge bits $1 - e^j$ verify for every jth proof. Indistinguishability follows from the adaptive security of the Sigma protocol and the adaptive security of the commitment scheme. Adaptive zero-knowledge also follows along the same lines provided the sigma protocol satisfies adaptive zero-knowledge.

11

2.4 Constructing Adaptively Secure Sigma Protocols with Special Soundness

Next, we show various instantiations of our adaptively sigma protocol which also satisfies special soundness. Plugging these protocols in a blackbox manner into our above compiler would yield a triply adaptive NIZK protocol.

Schnorr's [Sch90] Protocol. The classic Schnorr's identification protocol provides HVZK and satisfies special soundness. It also provides security against adaptive corruption. Let us recall the protocol and demonstrate that the Sigma protocol trivially satisfies adaptive security.

In the Schnorr's protocol the prover has a witness $w \in \mathbb{Z}_q$ and statement $x \in \mathbb{G}$ such that $x = g^w$, where $g \in \mathbb{G}$ is a generator of the cyclic group $\mathbb{G}$ where Discrete Log problem holds. The prover samples a random $r \in \mathbb{Z}_q$ and sets $a = g^r$. Upon obtaining a random challenge $c \in \mathbb{Z}_q$ from the verifier the prover sends $z = r + wc$ as the response. The verifier checks that $g^z \stackrel{?}{=} a \cdot x^c$. Given two accepting trancripts (a, c, z) and (a, c', z') the witness w can be extracted as $w = \frac{(z-z')}{c-c'}$. On the other hand, for HVZK the simulator samples a random $c \in \mathbb{Z}_q$ and a random $z \in Zq$ and computes $a = \frac{g^z}{x^c}$. Upon post-execution corruption of prover, the simulator obtains w and sets $r = z - wc$ as the internal state. It is straightforward to see that adaptive security follows.

Adaptive Soundness and Adaptive ZK. Adaptive soundness cannot be defined for Schnorr's protocol since every statement $x' \in \mathbb{G}$ lies in the language corresponding to the witness $w' \in \mathbb{Z}_q$ where $x' = g^{w'}$. Adaptive ZK follows from the HVZK property of the protocol.

Sigma protocol of [FLS99]. We briefly recall the Sigma protocol of [FLS99] for the sake of completeness. Let $\mathcal{R}_{\mathsf{Ham}}$ be the set of Hamiltonian graphs. The prover P proves that an n-node graph G is Hamiltonian, i.e. $G \in \mathcal{R}_{\mathsf{Ham}}$, given a Hamiltonian cycle σ as a witness. P samples a random n-node cycle H and commits to the adjacency matrix of the cycle as the first message a. The matrix contains n^2 entries, and P commits to the edges as $\mathsf{Com}(1)$, and non-edges as $\mathsf{Com}(0)$. The prover sends these commitments to the verifier V. V samples a random challenge bit e and sends it to the prover. If $c = 0$, then P decommits to the cycle H. Else, it computes a random permutation π s.t. $H = \pi(\sigma)$ and decommits to the non-edges in $\pi(G)$ and sends π. P sends the decommitments as its response z. Upon obtaining z, the verifier performs the following check:

- $c = 0$: Verify that z contains decommitments to 1, and they form a valid cycle, i.e. the prover must have committed to a valid n-node cycle.
- $c = 1$: Verify that z contains decommitments to 0, and the decommitted edges correspond to non-edges in $\pi(G)$.

Special Soundness. There are only two possible challenges in the boolean challenge space. Given the transcripts $(a, 0, z_0)$ and $(a, 1, z_1)$ where a_c and a'_c are computed as described above, the witness extractor obtains H from z_0 and π from z_1. The extractor computes the witness cycle as $\sigma = \pi^{-1}(H)$. This proves special soundness property of the Sigma protocol.

Honest Verifier Zero Knowledge. The FLS protocol achieves honest verifier zero knowledge. The ZK simulator samples a random challenge $e \in \{0, 1\}$ and based on that he computes (a, z) as follows.

- $c = 0$: The simulator samples a random n-node cycle H and commits to the adjacency matrix of the cycle as a. It sets z as the decommitment to the cycle.
- $c = 1$: The simulator sets all the commitments to 0 in a, i.e. commits to a null graph. It computes a random permutation π and decommits to the non-edges in $\pi(G)$. It sets z as π and the decommitments to the non-edges in $\pi(G)$.

Let us denote a proof as $\gamma = (a, e, z)$. It can be observed that an honest γ is identically distributed to a simulated γ when $e = 0$. When $e = 1$, an honestly γ contains a committed cycle whereas γ contains commitments to 0. The two proofs are indistinguishable due to the hiding of the commitment scheme.

Adaptive Security in $\mathcal{F}_{\textsf{NICOM}}$ model. We observe that the FLS protocol satisfies adaptive security if the commitments in a are computed in the $\mathcal{F}_{\textsf{NICOM}}$ model. We consider the simulated ZK proof and adaptive corruption of prover as follows:

- $c = 0$: The HVZK simulator samples a random n-node cycle H and commits to the adjacency matrix of the cycle as a by invoking $\mathcal{F}_{\textsf{NICOM}}$. It sets z as the decommitment to the cycle.
  Upon post execution corruption of prover, the simulator obtains the witness cycle σ and it computes the permutation π such that $H = \pi(\sigma)$. The internal state of the prover is set as a, permutation π and the internal state of the committer returned by $\mathcal{F}_{\textsf{NICOM}}$ (for computing the commitments in a).
- $c = 1$: The HVZK simulator sets a as the commitments to 0 in the $\mathcal{F}_{\textsf{NICOM}}$ model, i.e. the simulator commits to a null graph. It computes a random permutation π, and sets z as the random permutation π and the decommitments to the non-edges in $\pi(G)$.
  Upon post execution corruption of prover, the simulator obtains the witness cycle σ and it computes the permutation π such that $H = \pi(\sigma)$. The simulator equivocates the unopened commitments in a by invoking the $\mathcal{F}_{\textsf{NICOM}}$ simulator, such that the unopened commitments decommit to H. The internal state of the prover is set to the permutation π and the commitment randomness returned by $\mathcal{F}_{\textsf{NICOM}}$ for all the commitments.

For the case of $c == 0$, it can be observed that the simulated internal state is identical to the honest prover internal state. When $c == 1$, the simulated proof consists of commitments to 0 and the simulated prover internal state consists of equivocation randomness which was returned by $\mathcal{F}_{\textsf{NICOM}}$. Hence, the real and ideal world views are identically distributed in the $\mathcal{F}_{\textsf{NICOM}}$ model. This shows that the FLS protocol can be plugged into our NIZK compiler to obtain a NIZK protocol which is secure against adaptive corruptions.

Adaptive Soundness and Adaptive ZK. In FLS, the first message a of the prover is computed based on the parameter n without the knowledge of the graph or the witness. After obtaining c from V, the prover requires the input graph G and the witness cycle σ to construct the response. Thus, only the last message in this protocol depends on the input. This property is called delayed-input property. And hence the FLS protocol trivially satisfies adaptive soundness and adaptive ZK in the $\mathcal{F}_{\textsf{NICOM}}$ model where the input statement can be adversarially chosen after observing the setup string distribution. This allows our NIZK protocol to be adaptively sound and satisfy adaptive ZK when the FLS Sigma protocol is plugged into the triply adaptive NIZK compiler.

Blum's protocol for Hamiltonicity. Next, we build upon the result of The work of [LZ09] constructed the first adaptively secure proofs for NP based on Blum's protocol [Blu86] for Hamiltonicity. They use instance-dependent commitment schemes for this purpose. However, such commitment schemes are incompatible (shown on the next page for [HV16]) with CI-hash functions for Fiat-Shamir transform since they lack an extraction trapdoor. We replace the instance-dependent commitment scheme with $\mathcal{F}_{\mathsf{NICOM}}$ and show that Blum's protocol satisfies adaptive security and special soundness in the $\mathcal{F}_{\mathsf{NICOM}}$ model. The protocol itself does not satisfy the delayed input property since the first message of the prover depends on the statement. However, the protocol does achieve adaptive soundness since a malicious prover would be unsuccessful in generating an accepting proof for a statement $x \notin \mathcal{L}$ in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

Garbled circuit based protocol of [HV16]. Next, we modify the GC-based protocol of [HV16] to obtain an adaptively secure sigma protocol with special soundness in the $\mathcal{F}_{\mathsf{NICOM}}$ model. We recall their protocol and then discuss the bottlenecks involved.

Protocol of [HV16]. The protocol of [HV16] constructs an adaptively secure ZK proof from one-way functions in the plain model. Their protocol relies on a special commitment scheme called adaptive-instance dependent commitment (AIDCS) schemes. It depends on the statement being proven. AIDCS is statistically binding when the statement (being proven) is not in the language. AIDCS is equivocal when the statement is in the language. The committer can open a commitment to any message given an accepting witness for the statement. In [HV16], the prover constructs a garbled circuit computing the NP relation on the statement x. The prover commits to the garbled circuit $\mathbf{GC}$ (Section. 3.4 contains garbling notations), encoding information $\mathbf{u}$ and the decoding information $\mathbf{v}$ using the AIDCS. These commitments are jointly denoted as the first message a. The verifier sends the challenge bit c. If the bit is $c = 0$ then the prover decommits to $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$. The verifier checks that the garbled circuit was correctly constructed. If the bit is $c = 1$ then the prover computes the input wire labels W corresponding to the witness w and decommits to W, the decoding information $\mathbf{v}$ and the path of the computation as $\mathsf{path} = \Pi_{\mathsf{Ev}}(\mathbf{GC}, \mathsf{W})$ in the $\mathbf{GC}$ which corresponds to evaluation of $\mathbf{GC}$ on W. The verifier accepts if the computation of the garbled circuit on W along the path outputs 1. When x is not in the language the AIDCS is statistically binding and hence the prover has to guess the verifier's bit. For ZK, the ZK simulator will guess the random challenge bit of verifier and it will rewind if the guess is wrong. When the prover gets corrupted post-execution, the simulator can equivocate the commitments given the witness w using the equivocal property of AIDCS.

Bottlenecks in obtaining NIZK. The protocol of [HV16] obtains soundness but it does not achieve special soundness since the ZK proof is not binding when $x \in \mathcal{L}$. If a corrupt prover knows the witness then the AIDCS is equivocal given the witness and as a result a malicious prover evades the special soundness property by equivocating the commitments. We describe the following adversarial strategy for completeness: Assume $x \in \mathcal{L}$, then the adversary constructs the AIDCS in the equivocal mode as the first message a and it constructs the responses as follows:

- $c_0 == 0$: It samples a garbled circuit as $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$ and sets z_0 as $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$ and the decommitment of a to $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$.
- $c_1 == 1$: It invokes the privacy simulator of the garbled circuit on output 1 to obtain a simulated GC and input wire labels for evaluation. The adversary sets the response z_1 as

these wire labels and the path of GC evaluation. The response z_1 also contains the decommitments of a to the wire labels and the evaluation path.

The adversary is able to equivocate the AIDCS to open to different values and this hampers witness extraction from the two accepting transcripts (a, c_0, z_0) and (a, c_1, z_1). This hampers the special soundness property.

Our Solution. We solve this issue by replacing the AIDCS with the $\mathcal{F}_{\mathsf{NICOM}}$ model and demonstrate that the new Sigma protocol in the $\mathcal{F}_{\mathsf{NICOM}}$ model satisfies adaptive security and special soundness property. The prover constructs a garbled circuit computing the NP relation on the statement x. The prover sets a as the commitment to garbled circuit $\mathbf{GC}$, encoding information $\mathbf{u}$ and the decoding information $\mathbf{v}$ in the $\mathcal{F}_{\mathsf{NICOM}}$ model. The prover sends a to the verifier. The verifier sends the challenge bit c. The prover performs the following based on challenge c:

- $c = 0 :$ The prover decommits to the garbled circuit $\mathbf{GC}$, encoding information $\mathbf{u}$ and decoding information $\mathbf{v}$ as the response z_0.
- $c = 1 :$ The prover decommits to the input wire labels and the evaluation path in the garbled circuit as the response z_1.

The verifier performs verification using the original verifier algorithm of [HV16]. Completeness is straightforward. Next, we show that the above Sigma protocol satisfies special soundness property and adaptive security in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

Special Soundness. There are only two possible challenges in the boolean challenge space. Given two accepting transcripts $(a, 0, z_0)$ and $(a, 1, z_1)$, the witness extractor obtains the encoding information $\mathbf{u}$ and the input wire labels W. Assuming the garbling scheme is projective (for every input wire in the circuit the encoding information consists of two posible wire labels corresponding to bit values 0 and 1), it maps the wire labels with the encoding information to extract the witness w. This proves special soundness property of the Sigma protocol.

Adaptive Security in $\mathcal{F}_{\mathsf{NICOM}}$ model. We describe the HVZK simulator and then extend it to satisfy adaptive security in the $\mathcal{F}_{\mathsf{NICOM}}$ model. We crucially rely on the reconstructability property of the garbling scheme to argue adaptive security. Reconstructability property says that given a path of computation, the input wire labels and the input to a garbled circuit for circuit C it is possible to reconstruct the rest of the garbled circuit as being honestly generated by the garbling algorithm. We define the HVZK simulator as follows based on the challenge $c :$

- $c = 0 :$ The HVZK simulator computes a fresh garbled circuit as $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$ and commits to it using $\mathcal{F}_{\mathsf{NICOM}}$ as the first message a. It sets a as the commitment to $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$. The simulator sends z_0 as $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$ and the decommitments to a.
  When the prover gets adaptively corrupted, the simulator obtains the witness w and it sets the randomness used to garble $\mathbf{GC}$ and the commitment randomness as the internal randomness.
- $c = 1 :$ The HVZK simulator invokes the GC privacy simulator on output 1 and circuit C to obtain a simulated garbled circuit, input wire label, decoding information and internal state - $(\mathbf{GC}', \mathsf{W}', \mathbf{v}', \mathsf{st}')$. The HVZK simulator sets a as the commitment to $(\mathbf{GC}', 0^{|\mathbf{u}|}, \mathbf{v}')$ in the $\mathcal{F}_{\mathsf{NICOM}}$ model. The simulator computes the path of computation as $\mathsf{path} = \mathit{\Pi}_{\mathsf{Ev}}(\mathbf{GC}', \mathsf{W}')$ on wire labels W'. The simulator sends z_1 as $(\mathsf{path}, \mathsf{W}')$ and decommitment to $(\mathsf{path}, \mathsf{W}')$ from the set of commitments in a.

15

When the prover gets adaptively corrupted, the simulator obtains the witness w. Using input w, simulated input wire labels W' and the computation path path, it uses the reconstructability property of the garbling scheme to reconstruct a fresh garbled $\mathbf{GC}$, encoding information $\mathbf{u}$ and decoding information $\mathbf{v}$ and the corresponding garbling randomness. It sets the garbling randomness as the internal state and invokes the $\mathcal{F}_{\mathsf{NICOM}}$ simulator to equivocate the commitments in a such that they open to $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$.

For the case of $c == 0$, it can be observed that the simulated internal state is identical to the honest prover internal state. When $c == 1$, the proof contains the evaluation path, the input wire labels and their decommitments. Upon post execution corruption the simulator relies on the reconstructability property of the garbling scheme to construct the garbled circuit. The distribution of the simulated a in the ideal world is indistinguishable from the honestly constructed a in the real world in the $\mathcal{F}_{\mathsf{NICOM}}$ model due to the reconstructability property. The garbling scheme of [LP09] based on one-way functions satisfies all the required properties for the Sigma protocol. This was shown in the work of [HV16].

Adaptive Soundness and Adaptive ZK. The protocol achieves adaptive soundness and adaptive ZK even when the functionality $\mathcal{F}_{\mathsf{NICOM}}$ is implemented by an adaptively secure commitment protocol [CF01] in the crs model. The distribution of crs is identical in the real and ideal world. A malicious prover fails to prove a false statement $x \notin \mathcal{L}$ without breaking the binding of the commitment scheme (implementing $\mathcal{F}_{\mathsf{NICOM}}$ functionality). Adaptive ZK follows from the adaptive security of the protocol.

2.5 Reducing the Length of the crs

Let $\mathsf{crs}_{\mathsf{NIZK}}$ denote the setup string for our NIZK protocol Π_{NIZK}. Currently the $\mathsf{crs}_{\mathsf{NIZK}}$ contains the public hash key k which depends on the circuit length. We reduce this to $\mathrm{poly}(\kappa)$ by applying a compiler which compiles any single-prover multi-proof NIZK protocol Π_{NIZK} in the $\mathsf{crs}_{\mathsf{NIZK}}$ model to a NIZK protocol Π_{sNIZK} in the short $\mathsf{crs}_{\mathsf{sNIZK}}$ model, where $|\mathsf{crs}_{\mathsf{sNIZK}}| = \mathrm{poly}(\kappa)$, assuming additively homomorphic equivocal commitment Com and a PKE with oblivious ciphertext sampling algorithm. Our compiler is inspired from the work of GOS and it can be instantiated from DDH or LWE.

Given the witness y and the statement x for a language $\mathcal{L}$, the prover computes a circuit $\mathcal{C}$ s.t. $\mathcal{C}(y) = \mathcal{R}(x, y)$ where $\mathcal{R}$ is the NP verification relation. Let $y = \{y_i\}_{i \in [|y|]}$. The prover commits to each bit y_i as $c_i = \mathsf{Com}(y_i; r_i)$ and encrypts the corresponding randomness as $e_{i,y_i} = \mathsf{Enc}(\mathsf{pk}, r_i; s_i)$ while $e_{i,\overline{y_i}}$ is sampled obliviously. The output wire is committed as $\mathsf{Com}(1; 1)$. Using Π_{NIZK} the prover proves that each c_i is a commitment to 0 or 1 and the underlying commitment randomness is also encrypted correctly in $e_{i,0}$ or $e_{i,1}$. For each jth NAND gate with input wires α and β and output wires Γ, it computes $C_j = c_\alpha + c_\beta + 2c_\Gamma - 2\mathsf{Com}(1; 0)$ and proves using Π_{NIZK} that C_j is a commitment to 0 or 1. The GOS protocol showed that if the order of the message domain of Com is at least 4 then C_j will always be a commitment to 0 or 1. The verifier verifies the proofs and checks that the commitment corresponding to the output wire is $\mathsf{Com}(1; 1)$.

Adaptive Soundness and Proof of Knowledge. The distribution of the crs is identical in the real and ideal world. A corrupt prover P^* can adaptively chose the statement based on the crs distribution. If P^* constructs a proof for a statement $x \notin \mathcal{L}$, then he must have broken the binding property of the commitment scheme or the soundness of Π_{NIZK}. Else if P^* constructs a proof for

a statement $x \in \mathcal{L}$ then the simulator can extract the witness bits y_i from the individual proofs by invoking the simulator of Π_{NIZK}. Note that we encrypt the randomness r_i for each commitment c_i for reduction in the security proof. If a corrupt prover computes two valid openings of c_i then those openings can be decrypted and used to break the binding property of Com.

Adaptive Zero Knowledge. Zero-knowledge is ensured since a ZK simulator can construct the c_i commitments in the equivocal mode, i.e. $c_i = \mathsf{Com}(0; r_i) = \mathsf{Com}(1; r_i')$, and set the encryptions as $(e_{i,0}, e_{i,1}) = (\mathsf{Enc}(\mathsf{pk}, r_i; s_i), \mathsf{Enc}(\mathsf{pk}, r_i'; s_i'))$. He sets the output wire commitment as $\mathsf{Com}(1; 1)$. He invokes the ZK simulator of Π_{NIZK} to produce the proofs for each wire and each NAND gate. In the real world one of the encryptions corresponding to a commitment is honestly generated while the other is obliviously sampled. In the ideal world both encryptions are honestly generated. The two cases are indistinguishable due to oblivious ciphertext sampling property of PKE. Thus, ZK follows from the ZK of Π_{NIZK}, hiding of Com and oblivious ciphertext sampling property of PKE. A statically corrupt verifier can also choose the statement adaptively based on the crs distribution. Adaptive ZK of this protocol is ensured since Π_{NIZK} supports adaptive ZK, the Com is hiding and PKE provides oblivious ciphertext sampling property.

Security against Adaptive Corruptions. When the prover gets corrupted and it obtains the witness w it can compute y. Suppose $y_i = 0$, then he opens c_i and $e_{i,0}$ as $(0, r_i, s_i)$ and claims that $e_{i,1}$ was obliviously sampled. It invokes the Π_{NIZK} simulator of wire i with input witness (y_i, r_i) to obtain randomness for the NIZK proof corresponding to wire i. Similar steps are repeated to obtain randomness for each NAND gate j and the corresponding NIZK proof. The encryption of r_i' (in ideal world) is indistinguishable from an obliviously sampled ciphertext (in real world) due to the oblivious sampling property. Thus, security against adaptive corruption is ensured due to adaptive security of Π_{NIZK}, equivocal property of Com and oblivious sampleability of PKE.

Multi-proof. The protocol Π_{sNIZK} also allows the prover to prove multiple statements using the same $\mathsf{crs}_{\mathsf{sNIZK}}$. If a corrupt party breaks the security of the protocol in one of the proof then that party can be used to either break the multi-proof security of Π_{NIZK} or the security of Com.

2.6 Obtaining UC Security for Multiple Subsessions

We add non-malleability to our Π_{sNIZK} protocol to attain UC-security for multiple statements in different subsessions. This is performed in the same way as GOS using tag based simulation-sound trapdoor commitment $\mathsf{Com}_{\mathsf{SST}}$ (defined in Section 3.2) and strong one-time signature SIG. The prover generates a pair of signature keys $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG}.\mathsf{KeyGen}$. It commits to the witness bits w using $\mathsf{Com}_{\mathsf{SST}}$ with the tag being $(\mathsf{vk}, \mathsf{sid}, \mathsf{ssid}, x)$ (where $\mathsf{sid}, \mathsf{ssid}$ is the session ID of the multi-instance NIZK functionality and ssid is the sub-session ID for the particular proof) and encrypts the randomness for the commitments. It proves using Π_{sNIZK} that $\mathcal{R}(x, w) = 1$ and the witness bits are correctly committed to compute a proof π. It signs the proof π using sk and sends the proof π and the signature as the final proof. The signature enables that an adversary cannot forge a signature on a different proof with the same vk. Whereas, $\mathsf{Com}_{\mathsf{SST}}$ and Π_{sNIZK} ensures that an adversary cannot reuse the same proof π in a different session ssid since it is bound to the vk and ssid.

Security against Statically Corrupt Prover. Soundness follows from the binding property of $\mathsf{Com_{SST}}$, unforgeability of SIG and adaptive soundness of Π_{sNIZK}. The witness can be extracted from the commitments by decrypting the randomness of the commitments from the encryptions using sk. Next, we briefly discuss the different cases for triple adaptive security.

Security against Adaptive Corruption of Prover. The ZK simulator commits to all 0s as witness and invokes the ZK simulator of Π_{sNIZK} to construct the simulated proof. Upon obtaining the witness it can equivocate the commitments using the trapdoor and equivocate the proof by invoking the adaptive simulator of Π_{sNIZK}.

Adaptive Soundness and Adaptive Zero Knowledge. The crs distribution is identical in the real and ideal world for our multi-session UC protocol. Adaptive soundness follows from the unforgeability of signature, binding property of the tag-based commitment scheme and adaptive soundness of underlying single instance NIZK protocol. For adaptive ZK our ZK simulator (mentioned in previous paragraph) suffices.

3 Preliminaries

Notations. We denote by $a \leftarrow D$ a uniform sampling of an element a from a distribution D. The set of elements $\{1, 2, \ldots, n\}$ is represented by $[n]$. A function $\mathsf{neg}(\cdot)$ is said to be negligible, if for every polynomial $p(\cdot)$, there exists a constant c, such that for all $n > c$, it holds that $\mathsf{neg}(n) < \frac{1}{p(n)}$. We denote a probabilistic polynomial time algorithm as PPT. We denote the computational and statistical security parameters by κ by μ respectively. We denote computational and statistical indistinguishability by $\overset{c}{\approx}$ and $\overset{s}{\approx}$ respectively. When a party $\mathcal{P}$ gets corrupted we denote it by $\mathcal{P}^*$. Let $\mathcal{R}_{\mathsf{Ham}}$ denote the set of n-node Hamiltonian graphs for $n > 1$. We prove security of our protocol in the Universal Composability (UC) model. We refer to the original paper [Can01] for details. Our protocols are in the common reference string model where the parties of a session (sid) have access to a public reference string crs sampled from a distribution. In the one-time crs model, each crs is local to each (sid). In the reusable crs model, the same crs can be reused across different sessions by different parties. The simulator knows the trapdoors of the crs in both cases. We refer to [CLOS02] for more details.

Definition 1. *([DN00]* **PKE** *with oblivious ciphertext sampling) A public key encryption scheme* $\mathrm{PKE} = (\textbf{\textit{KeyGen}}, \textbf{\textit{Enc}}, \textbf{\textit{Dec}})$ *over message space* $\mathcal{M}$*, ciphertext space* $\mathcal{C}$ *and randomness space* $\mathcal{R}$ *satisfies oblivious ciphertext sampling property if there exists PPT algorithms* $(\textbf{\textit{oEnc}}, \textbf{\textit{Inv}})$ *s.t. for any message* $m \in \mathcal{M}$*, the following two distributions are computationally indistinguishable to a PPT adversary* $\mathcal{A}$*:*

$$\big| \Pr[\mathcal{A}(m, c, r) = 1 | m \leftarrow \mathcal{A}(\textbf{\textit{pk}}), c \leftarrow \textbf{\textit{Enc}}(\textbf{\textit{pk}}, m; r'), r \leftarrow \textbf{\textit{Inv}}(\textbf{\textit{pk}}, c)]$$

$$- \Pr[\mathcal{A}(m, \tilde{c}, r) = 1 | m \leftarrow \mathcal{A}(\textbf{\textit{pk}}), \tilde{c} \leftarrow \textbf{\textit{oEnc}}(\textbf{\textit{pk}}; r)] \big| \leq \textbf{\textit{neg}}(\kappa),$$

where $(\textbf{\textit{pk}}, \textbf{\textit{sk}}) \leftarrow \textbf{\textit{KeyGen}}(1^\kappa)$.

We instantiate CCA-2 secure PKE with oblivious ciphertext sampling from DDH [CS98] and LWE [MP12].

3.1 Non-Interactive Zero Knowledge

We provide the ideal UC-NIZK functionality in Fig. 3 for a single prover and a single proof. It also considers the case for adaptive corruption of parties where the prover gets corrupted after outputting the proof π. In such a case, the adversary receives the internal state of the prover.

Fig. 3. Single-Proof Non-Interactive Zero-Knowledge Functionality $\mathcal{F}_{\mathsf{NIZK}}$

$\mathcal{F}_{\mathsf{NIZK}}$ is parametrized by an NP relation $\mathcal{R}$. (The code treats $\mathcal{R}$ as a binary function.)

- **Proof:** On input $(\mathsf{prove}, \mathsf{sid}, x, w)$ from party P: If $\mathcal{R}(x, w) = 1$ then send $(\mathsf{prove}, P, \mathsf{sid}, x)$ to $\mathcal{S}$. Upon receiving $(\mathsf{proof}, \mathsf{sid}, \pi)$ from $\mathcal{S}$, store $(\mathsf{sid}, x, w, \pi)$ and send $(\mathsf{proof}, \mathsf{sid}, \pi)$ to P.
- **Verification:** On input $(\mathsf{verify}, \mathsf{sid}, x, \pi)$ from a party V: If $(\mathsf{sid}, x, w, \pi)$ is stored, then return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \mathcal{R}(x, w))$ to V. Else, send $(\mathsf{verify}, V, \mathsf{sid}, x, \pi)$ to $\mathcal{S}$. Upon receiving $(\mathsf{witness}, \mathsf{sid}, w)$ from $\mathcal{S}$, store $(\mathsf{sid}, x, w, \pi)$, and return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \mathcal{R}(x, w))$ to V.
- **Corruption:** When receiving $(\mathsf{corrupt}, \mathsf{sid})$ from $\mathcal{S}$, mark sid as corrupted. If there is a stored tuple $(\mathsf{sid}, x, w, \pi)$, then send it to $\mathcal{S}$.

We also consider $\mathcal{F}_{\mathsf{NIZK}}^{\mathsf{m}}$ (Fig. 4) functionality where a single prover can parallelly prove multiple statements in a single session. The verifier verifies each of them separately. It is a weaker notion than multi-session UC NIZK since $\mathcal{F}_{\mathsf{NIZK}}^{\mathsf{m}}$ considers only a single session between a pair of parties with roles preserved. Different provers have to use different instances of $\mathcal{F}_{\mathsf{NIZK}}^{\mathsf{m}}$ to prove statements.

Fig. 4. Non-Interactive Zero-Knowledge Functionality $\mathcal{F}_{\mathsf{NIZK}}^{\mathsf{m}}$ for single prover multi-proof setting

---

$\mathcal{F}_{\mathsf{NIZK}}$ is parametrized by an NP relation $\mathcal{R}$. (The code treats $\mathcal{R}$ as a binary function.)

- **Proof:** On input $(\mathsf{prove}, \mathsf{sid}, x, w, P)$ from party P: If there exists $(\mathsf{sid}, P') \in \mathcal{Q}$ and $P \neq P'$ or $\mathcal{R}(x, w) \neq 1$ then ignore the input. Else record $\mathcal{Q} = (\mathsf{sid}, P)$. Send $(\mathsf{prove}, P, \mathsf{sid}, x)$ to $\mathcal{S}$. Upon receiving $(\mathsf{proof}, \mathsf{sid}, \pi)$ from $\mathcal{S}$, store $(\mathsf{sid}, x, w, \pi)$ and send $(\mathsf{proof}, \mathsf{sid}, \pi)$ to P.
- **Verification:** On input $(\mathsf{verify}, \mathsf{sid}, x, \pi)$ from a party V: If $(\mathsf{sid}, x, w, \pi)$ is stored, then return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \mathcal{R}(x, w))$ to V. Else, send $(\mathsf{verify}, V, \mathsf{sid}, x, \pi)$ to $\mathcal{S}$. Upon receiving $(\mathsf{witness}, \mathsf{sid}, w)$ from $\mathcal{S}$, store $(\mathsf{sid}, x, w, \pi)$, and return $(\mathsf{verification}, \mathsf{sid}, x, \pi, \mathcal{R}(x, w))$ to V.
- **Corruption:** When receiving $(\mathsf{corrupt}, \mathsf{sid})$ from $\mathcal{S}$, mark (sid) as corrupted. If there are stored tuples of the form $(\mathsf{sid}, x, w, \pi)$, then send it to $\mathcal{S}$.
  On input $(\mathsf{corrupt\text{-}check}, \mathsf{sid})$, return whether (sid) is marked as corrupted.

Next, we define the notion of triple adaptive security for NIZK protocols and provide the property-based definitions of NIZK for completeness. UC-secure NIZKs in the crs model imply adaptive ZK since an environment can statically corrupt the verifier, obtain the crs of the protocol and then choose the statement x to be proven by the honest prover. The simulator against a corrupt verifier ensures that it constructs an accepting simulated proof which is indistinguishable from an honestly generated proof. Hence, UC-NIZK implies adaptive ZK if the environment is allowed to choose the statement being proven after corrupting the verifier.

Definition 2. *A non-interactive zero-knowledge argument system (NIZK) for an NP-language $\mathcal{L}$ consists of three PPT machines $\Pi_{\textit{NIZK}} = (\textit{Gen}, \textit{P}, \textit{V})$, that have the following properties:*

- *Completeness: For all $\kappa \in \mathbb{N}$, and all $(x, w) \in \mathcal{R}$, it holds that:*

$$\Pr[\textit{V}(\textit{crs}, x, \textit{P}(\textit{crs}, x, w)) = 1 | (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^\kappa, 1^{|x|})] = 1.$$

– **Soundness**: *For all PPT provers P^* and $x \notin \mathcal{L}$ the following holds for all $\kappa \in \mathbb{N}$:*

$$\Pr[V(crs, x, \pi) = 1 | (crs, td) \leftarrow Gen(1^\kappa, 1^{|x|}), \pi \leftarrow P^*(crs)] \leq neg(\kappa).$$

– **Zero knowledge**: *There exists a PPT simulator $\mathcal{S}$ such that for every $(x, w) \in \mathcal{R}$, the following distribution ensembles are computationally indistinguishable:*

$$\{(crs, \pi) | (crs, td) \leftarrow Gen(1^\kappa, 1^{|x|}), \pi \leftarrow P(crs, x, w)\}_{\kappa \in \mathbb{N}}$$

$$\approx \{(crs, \{\mathcal{S}(1^\kappa, x, td)\}) | (crs, td) \leftarrow Gen(1^\kappa, 1^{|x|}\}_{\kappa \in \mathbb{N}}$$

Definition 3. *(**Full Adaptive Soundness**) Π_{NIZK} is adaptively sound if for every PPT cheating prover P^* the following holds:*

$$\Pr[x \notin \mathcal{L} \wedge V(crs, x, \pi) = 1 | (crs, td) \leftarrow Gen(1^\kappa, 1^{|x|}), (x, \pi) \leftarrow P^*(crs)] < neg(\kappa).$$

Definition 4. *(**Adaptive Zero-Knowledge**) Π_{NIZK} is adaptively zero-knowledge if for all PPT verifiers V^* there exists a PPT simulator $\mathcal{S}$ such that the following distribution ensembles are computationally indistinguishable:*

$$\{(crs, P(crs, x, w), aux)\} \overset{c}{\approx} \{\mathcal{S}(crs, td, 1^\kappa, x)\}_{\kappa \in \mathbb{N}}$$

where $(crs, td) \leftarrow Gen(1^\kappa, 1^{|x|})$ and $(x, w, aux) \leftarrow V^*(crs)$.

The Gen algorithm takes the $|x|$ (length of the statement) as input to generate the crs. This shows that the crs size depends on $|x|$. When the crs is independent of $|x|$, the Gen algorithm only takes 1^κ as input.

Definition 5. *(**Triple Adaptive Security for a single instance**)*
Let $\Pi_{NIZK} = (Gen, P, V)$ be a NIZK protocol in the crs model. Then Π_{NIZK} satisfies triple adaptive security for a single instance if it securely implements $\mathcal{F}_{NIZK}$ functionality for a single instance and provides adaptive soundness and adaptive zero knowledge.

Definition 6. *(**Triple Adaptive Security for multiple instances**)*
Let $\Pi_{NIZK} = (Gen, P, V)$ be a NIZK protocol in the crs model. Then Π_{NIZK} satisfies triple adaptive security for multiple instances if it UC-securely implements $\mathcal{F}_{NIZK}$ functionality for multiple instances and provides adaptive soundness and adaptive zero knowledge.

3.2 Commitment Schemes

A commitment scheme $Com = (Gen, Com, Ver, Equiv)$ allows a committing party C to compute a commitment c to a message m, using randomness r, towards a party V in the Com phase. Later in the open phase, C can open c to m by sending the decommitment to V who verifies it using Ver. It should be binding, hiding and equivocal using Equiv algorithm given trapdoor td of the crs. Moreover, we require our commitment scheme to be additively homomorphic for message domain of size at least four, i.e. $Com(m_1; r_1) + Com(m_2; r_2) = Com(m_1 + m_2; r_1 + r_2)$. We also need a tag-based simulation sound commitment consists of $Com_{SST} = (KeyGen, Com, Ver, TCom, TOpen)$ for our protocols.

We define an equivocal commitment scheme $Com = (Gen, Com, Ver, Equiv)$ as follows:

Definition 7. *(Correctness) Com is a correct commitment scheme if the following holds true*

$$\Pr\left[\textit{Ver}(m, c, \textit{crs}, r) = 1 | (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^{\kappa}), c \leftarrow \textit{Com}(m, \textit{crs}; r)\right] = 1$$

Definition 8. *(Binding) Com is computationally binding scheme if the following holds true for all PPT adversary $\mathcal{A}$*

$$\Pr\left[(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\textit{crs}) | (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^{\kappa}),\right.$$
$$\left.\textit{Com}(m_0; r_0) = \textit{Com}(m_1; r_1)\right] \leq \textit{neg}(\kappa)$$

Definition 9. *(Hiding) Com is computationally hiding scheme if the following holds true for all PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.*

$$\Pr\left[b == b' | (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^{\kappa}), (m_0, m_1, \textit{st}) \leftarrow \mathcal{A}_1(\textit{crs}), b \leftarrow_r \{0, 1\},\right.$$
$$\left.(c, d) \leftarrow \textit{Com}(m_b), b' \leftarrow \mathcal{A}_2(c; \textit{st})\right] \leq \frac{1}{2} + \textit{neg}(\kappa)$$

Definition 10. *(Equivocal) Com is equivocal if it has a PPT algorithm Equiv s.t. the following holds true for all PPT adversary $\mathcal{A}$ and all message pairs (m, m').*

$$\left| \Pr\left[\mathcal{A}(c, r) = 1 | (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^{\kappa}), (m, m') \leftarrow \mathcal{A}(\textit{crs}), c = \textit{Com}(\textit{crs}, m; r)\right]\right.$$
$$- \Pr\left[\mathcal{A}(c, r) = 1 | (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^{\kappa}), (m, m') \leftarrow \mathcal{A}(\textit{crs}), c = \textit{Com}(\textit{crs}, m'; r'),\right.$$
$$\left.\left. r = \textit{Equiv}(m, r', c, \textit{td})\right]\right| \leq \textit{neg}(\kappa), \text{ for } m \neq m'$$

Definition 11. *(Extractable) Com is extractable if it has a PPT algorithm Ext if the following holds true for any $m \in \{0, 1\}$ and $r \in \{0, 1\}^*$.*

$$\Pr\left[\textit{Ver}(m, c, \textit{crs}, r) = 1 \wedge \textit{Ext}(\textit{td}, c) \neq m : (\textit{crs}, \textit{td}) \leftarrow \textit{Gen}(1^{\kappa}),\right.$$
$$\left. c = \textit{Com}(m; r)\right] \leq \textit{neg}(\kappa)$$

We denote $\mathsf{Com}(m, \mathsf{crs}; r)$ as $\mathsf{Com}(m; r)$ to avoid notation overloading.

Tag-based simulation soundness. A tag-based simulation sound commitment scheme is denoted as $\mathsf{Com}_{\mathsf{SST}} = (\mathsf{KeyGen}, \mathsf{Com}, \mathsf{Dec}, \mathsf{Ver}, \mathsf{TCom}, \mathsf{TOpen})$. We define it following [GOS12]. The key generation algorithm KeyGen produces a crs as well as a trapdoor key td. There is a Com algorithm that takes as input crs, a message m and any tag t and outputs a commitment $c = \mathsf{Com}(\mathsf{crs}, m, t; r)$. To open a commitment c with tag t we reveal m and the randomness r. Verification is performed by verifying the commitment with r on (m, t). $\mathsf{Com}_{\mathsf{SST}}$ is a correct, binding and hiding commitment scheme. $\mathsf{Com}_{\mathsf{SST}}$ has trapdoor opening if the following holds for all PPT adversary $\mathcal{A}$.

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, m, t, r) = 1 | (c, \alpha) \leftarrow \mathsf{TCom}(\mathsf{td}, t), m \leftarrow \mathcal{A}(c, \mathsf{crs}),\right.$$
$$\left. r \leftarrow \mathsf{TOpen}(\mathsf{crs}, \alpha, c, m, t)\right] = 1$$

The tag-based simulation-soundness property means that a commitment using tag t remains binding even if we have made equivocations for commitments using different tags. For all non-uniform PPT adversaries $\mathcal{A}$ we have

$$\Pr\left[(\mathsf{crs}, td) \leftarrow \mathsf{KeyGen}(1^\kappa), (c, t, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}^{O(\cdot)}(\mathsf{crs}) | t \notin Q, \right.$$

$$\left. c = \mathsf{Com}(m_0, t; r_0) = \mathsf{Com}(m_1, t; r_1), m_0 \neq m_1\right] \leq \mathsf{neg}(\kappa).$$

where $O(\mathsf{Com}, t)$ computes $(c, \alpha) \leftarrow \mathsf{TCom}(td, t)$ returns c and stores (c, t, α), and $O(open, c, m, t)$ returns $r \leftarrow \mathsf{TOpen}(\mathsf{crs}, \alpha, c, m, t)$ if (c, t, α) has been stored, and where Q is the list of tags for which equivocal commitments have been made by oracle $O(\cdot)$.

$\mathcal{F}_{\mathsf{NICOM}}$**-model.** We also provide a new non-interactive UC-commitment functionality in Fig. 2. The $\mathcal{F}_{\mathsf{NICOM}}$ functionality (Fig. 2) is implemented against adaptive adversaries using adaptively secure non-interactive UC commitments [CF01] in the crs model. We perform this using equivocal commitments and CCA-2 secure PKE with oblivious ciphertext sampleability in the non-programmable crs model. It can be found in Fig. 5. We prove in Appendix. A that this new functionality implies the old UC commitment functionality (Fig. 19) but our new functionality is more compatible with non-interactive protocols.

3.3 Correlation Intractability.

As in [CCH+19, PS19, BKM20] we define efficiently searchable relations and recall the definitions of correlation intractability, in their computational and statistical versions.

Definition 12. *We say that a relation $R \subseteq X \times Y$ is searchable in size S if there exists a function $f : X \to Y$ that is implementable as a boolean circuit of size S, such that if $(x, y) \in R$ then $y = f(x)$. (In other words, $f(x)$ is the unique witness for x, if such a witness exists.)*

Definition 13. *Let $R = \{\mathcal{R}_\kappa\}$ be a relation class, i.e., a set of relations for each κ. A hash function family $\mathcal{H} = (\mathsf{Gen}, H)$ is correlation intractable for R if for every non-uniform PPT adversary $\mathcal{A} = \{\mathcal{A}_\kappa\}$ and every $R \in \mathcal{R}_\kappa$ the following holds:*

$$\Pr[(x, H(k, x)) \in R : k \leftarrow \mathsf{Gen}(1^\kappa), x = \mathcal{A}_\kappa(k)] \leq \mathsf{neg}(\kappa)$$

Definition 14. *Let $R = \{\mathcal{R}_\kappa\}$ be a relation class. A hash function family $\mathcal{H} = (\mathsf{Gen}, H)$ with a fake-key generation algorithm $\mathsf{StatGen}$ is somewhere statistically correlation intractable for R if for every $R \in R_\kappa$ and circuits $\exists z_R \in Z_\kappa$ s.t:*

$$\Pr[\exists x \; s.t. \; (x, H(k, x)) \in R : k \leftarrow \mathsf{StatGen}(1^\kappa, z_R)] \leq \mathsf{neg}(\kappa).$$

and for every $z_\kappa \in Z_\kappa$ if the following distributions the indistinguishable:

$$\{\mathsf{StatGen}(1^\kappa, z_\kappa)\}_\kappa \stackrel{c}{\approx} \{\mathsf{Gen}(1^\kappa)\}_\kappa.$$

Definition 15. *A hash family $\mathcal{H} = (\mathsf{Gen}, H)$, with input and output length $n := n(\kappa)$ and, resp., $m := m(\kappa)$, is said to be programmable if the following two conditions hold:*

- *1-Universality: For every $\kappa \in \mathsf{N}$, $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^m$, the following holds: $\Pr[H(k, x) = y : k \leftarrow \mathsf{Gen}(1^\kappa)] = 2^{-m}$.*
- *Programmability: There exists a PPT algorithm $\mathsf{Gen}'(1^\kappa, x, y)$ that samples from the conditional distribution $Sample(1^\kappa)|H(k, x) = y$.*

3.4 Garbling Schemes

A garbling scheme [Yao86, LP09, BHR12] consists of the following algorithms: Gb takes a circuit C as input and outputs a garbled circuit $\mathbf{GC}$, encoding information $\mathbf{u}$, and decoding information $\mathbf{v}$. En takes an input x and encoding information $\mathbf{u}$ and outputs a garbled input X. Ev takes a garbled circuit and garbled input X and outputs a garbled output Y. Finally, De takes a garbled output Y and decoding information and outputs a plain circuit-output (or an error, $\perp$). There is an additional verification algorithm Ve in the garbling scheme which when accepts a given $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$ signifies that the $\mathbf{GC}$ is correct, and that the garbled output corresponding to any clear output can be extracted. Formally, a *garbling scheme* is defined by a tuple of functions Garble $= (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, \mathsf{Ve})$, described as follows:

- Gb $(1^\kappa, C) = (\mathbf{GC}, \mathbf{u}, \mathbf{v})$: A randomized algorithm which takes as input the security parameter and a circuit $C : \{0,1\}^n \to \{0,1\}^m$ and outputs a tuple of strings $(\mathbf{GC}, \mathbf{u}, \mathbf{v})$, where $\mathbf{GC}$ is the garbled circuit, $\mathbf{u}$ denotes the input-wire labels, and $\mathbf{v}$ denotes the decoding information.
- En $(x, \mathbf{u}) = \mathsf{X}$: a deterministic algorithm that outputs the garbled input X corresponding to input x.
- Ev $(\mathbf{GC}, \mathsf{X}) = \mathsf{Y}$: A deterministic algorithm which evaluates garbled circuit $\mathbf{GC}$ on garbled input X, and outputs a garbled output Y.
- De $(\mathsf{Y}, \mathbf{v}) = y$: A deterministic algorithm that outputs the plaintext output corresponding to Y or $\perp$ signifying an error if the garbled output Y is invalid.
- Ve $(C, \mathbf{GC}, \mathbf{u}, \mathbf{v}) = 1 or \perp$: A deterministic algorithm which takes as input a circuit $C : \{0,1\}^n \mapsto \{0,1\}^m$, a garbled circuit (possibly malicious) $\mathbf{GC}$, encoding information e and decoding information $\mathbf{v}$. It outputs 1 when $\mathbf{GC}$ is a valid garbling of C, and $\perp$ otherwise.

The garbling scheme used in our protocols need to satisfy several properties such as *correctness, privacy, verifiability and reconstructability*. We borrow the notations from the work of [HV16].

Definition 16. Perfect Correctness: *A garbling scheme* Garble *is perfectly **correct** if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$ and inputs $x \in \{0,1\}^n$, the following holds:*

$$\Pr\left[\mathsf{De}(\mathsf{Ev}(\mathbf{GC}, \mathsf{En}(\boldsymbol{u}, x)), \boldsymbol{v}) \neq C(x) : (\mathbf{GC}, \boldsymbol{u}, \boldsymbol{v}) \leftarrow \mathsf{Gb}(1^\kappa, C)\right] = 1.$$

Definition 17. Privacy: *A garbling scheme* Garble *is **private** if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, there exists a **PPT** simulator $\mathcal{S}$ such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following two distributions are computationally indistinguishable:*

- REAL(C, x) : *run* $(\mathbf{GC}, \boldsymbol{u}, \boldsymbol{v}) \leftarrow \mathsf{Gb}(1^\kappa, C)$, *and output* $(\mathbf{GC}, \mathsf{En}(x, \boldsymbol{u}), \boldsymbol{v})$.
- IDEAL$_\mathcal{S}(C, C(x))$: *Compute* $(\mathbf{GC}', \mathsf{X}, \boldsymbol{v}', \mathsf{st}) \leftarrow \mathcal{S}_{GC}(1^\kappa, C, C(x))$ *and output* $(\mathbf{GC}', \mathsf{X}, \boldsymbol{v}')$.

Definition 18. Verifiability: *A garbling scheme* Garble *is **verifiable** if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and PPT adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\mathsf{De}(\mathsf{Ev}(\mathbf{GC}, \mathsf{En}(x, \boldsymbol{u})), \boldsymbol{v}) \neq C(x) : \begin{array}{l} (\mathbf{GC}, \boldsymbol{u}, \boldsymbol{v}) \leftarrow \mathcal{A}(1^\kappa, C) \\ \mathsf{Ve}(C, \mathbf{GC}, \boldsymbol{u}, \boldsymbol{v}) = 1 \neq \perp \end{array}\right).$$

We are interested in a class of garbling schemes referred to as *projective* in [BHR12]. When garbling a circuit $C : \{0,1\}^n \mapsto \{0,1\}^m$, a projective garbling scheme produces encoding information of the form $\mathbf{u} = \left(\mathbb{K}_i^0, \mathbb{K}_i^1\right)_{i \in [n]}$, and the encoded input X corresponding to $x = (x_i)_{i \in [n]}$ can be interpreted as $\mathsf{X} = \mathsf{En}(x, \mathbf{u}) = (\mathbb{K}_i^{x_i})_{i \in [n]}$. In addition to the above properties, we also require it to be reconstructable as defined in [HV16]. Given a path of computation in a garbled circuit for circuit C it is possible to reconstruct the rest of the garbled circuit as being honestly generated by the garble algorithm. The path of computation in the garbled circuit is computed by computing a function $\mathsf{path} = \Pi_{\mathsf{Ev}}(\mathbf{GC}, \mathsf{X})$.

Definition 19. Reconstructability: *A garbling scheme* Garble *is* **verifiable** *is reconstructable if there exists a PPT algorithm* $\mathcal{S}_{GC} = (\mathcal{S}_{GC}^2, \mathcal{S}_{GC}^2)$ *and a projection function* Π_{Ev} *(as defined in [HV16]) such that for* $(\mathbf{GC}, \mathbf{u}, \mathbf{v}) \leftarrow \mathsf{Gb}(1^\kappa, C)$ *and* $\mathsf{X} \leftarrow \mathsf{En}(\mathbf{u}, x)$, *we have* $\mathsf{De}(\mathsf{Ev}(\Pi_{\mathsf{Ev}}(\mathbf{GC}, \mathsf{X}), \mathsf{X}), \mathbf{v}) = C(x) = y$ *and the following holds:*

$$\{(\mathbf{GC}', \mathsf{X}', \mathbf{v}, \mathsf{st}) \leftarrow \mathcal{S}_{GC}^1(1^\kappa, C, y) : \mathcal{S}_{GC}^2(\mathsf{st}, x)\} \approx$$

$$\{(\mathbf{GC}, \mathbf{u}, \mathbf{v}) \leftarrow \mathsf{Gb}(1^\kappa, C; r_{\mathsf{Gb}}), \mathsf{X} \leftarrow \mathsf{En}(\mathbf{u}, x) : (r_{\mathsf{Gb}}, \mathsf{X})\}$$

The scheme of [LP09] based on one-way functions satisfies all the above properties and it was shown in the work of [HV16].

4 New UC-Commitment Functionality $\mathcal{F}_{\mathsf{NICOM}}$

We provide a new non-interactive UC-commitment functionality in Fig. 2. The $\mathcal{F}_{\mathsf{NICOM}}$ functionality (Fig. 2) is implemented against adaptive adversaries using adaptively secure non-interactive UC commitments [CF01] in the crs model. We perform this using equivocal commitments and CCA-2 secure PKE with oblivious ciphertext sampleability in the non-programmable crs model. It can be found in Fig. 5. We prove in Appendix. A that this new functionality implies the old UC commitment functionality (Fig. 19) but our new functionality is more compatible with non-interactive protocols.

We implement $\mathcal{F}_{\mathsf{NICOM}}$ (Fig. 2) in this section based on the [CF01] protocol in Fig. 5. The committer P commits to a bit message m as $c = \mathsf{Com}(m; r)$. The commitment randomness is encrypted via a pair of encryptions. The committer encrypts the corresponding randomness r, subsession id ssid and committer id P using a CCA-2 secure PKE as $E_m = \mathsf{Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); s_m)$ with randomness s_m. The other encryption E_{1-m} is obliviously sampled using randomness s_{1-m}. The commitment consists of (c, E_0, E_1) and the opening information is (m, r, s_0, s_1). The verifier performs the canonical verification by reconstructing the commitment. The equivocal commitment can be instantiated from Pedersen Commitment and the obliviously sampleable encryption scheme can be instantiated from Cramer Shoup encryption [CS98], yielding a protocol from DDH. Similarly, we can instantiate the equivocal commitment from LWE [CsW19] and the obliviously sampleable encryption scheme from LWE [MP12]. We show that the [CF01] protocol in Fig. 5 implements the $\mathcal{F}_{\mathsf{NICOM}}$ functionality by proving Thm. 3 as follows.

Theorem 3. *Assuming* **Com** *is a non-interactive equivocal commitment scheme and* PKE *is a CCA2 secure public key encryption with oblivious ciphertext sampling and negligible correctness error, the protocol in Fig. 5 implements* $\mathcal{F}_{\mathsf{NICOM}}$ *in the crs model.*

– **Primitives:** Non-interactive equivocal commitment scheme Com=(Gen, Com, Ver, Equiv) and a CCA2-secure public key encryption scheme PKE = (KeyGen, Enc, Dec) with oblivious ciphertext sampling algorithm oEnc.
– **Public Inputs:** Setup string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{Com}}, \mathsf{pk})$ where $(\mathsf{crs}_{\mathsf{Com}}, \mathsf{td}) \leftarrow \mathsf{Com.Gen}(1^\kappa)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow$ PKE.KeyGen(1^κ). The trapdoors of crs are set as $\mathsf{td} = (\mathsf{td}_{\mathsf{Com}}, \mathsf{sk})$.
– **Private Inputs:** Committer P has a bit message $m \in \{0, 1\}$.

Commit Phase: Upon getting invoked with command $(\mathsf{Com}, \mathsf{ssid}, \mathsf{P}, m)$, the committer P commits to $m \in \{0, 1\}$ as follows:
– The committer samples randomness $r \leftarrow \{0, 1\}^*$ and computes $c = \mathsf{Com}(\mathsf{crs}_{\mathsf{Com}}, m; r)$.
– If $m == 0$, the committer encrypts commitment randomness r as $E_0 = \mathsf{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); s_0)$ with randomness $s_0 \leftarrow \{0, 1\}^*$ and obliviously samples $E_1 = \mathsf{PKE.oEnc}(\mathsf{pk}; s_1)$ with randomness $s_1 \leftarrow \{0, 1\}^*$.
– If $m == 1$, the committer encrypts commitment randomness r as $E_1 = \mathsf{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); s_1)$ with randomness $s_1 \leftarrow \{0, 1\}^*$ and obliviously samples $E_0 = \mathsf{PKE.oEnc}(\mathsf{pk}; s_0)$ with randomness $s_0 \leftarrow \{0, 1\}^*$.
The committer generates $\pi = \{c, E_0, E_1\}$ as the commitment to m and stores $\mathsf{st} = d = \{m, r, s_0, s_1\}$ as the opening information d and internal prover state st.

Decommitment Phase :
The committer sets decommitments as $d = \{m, r, s_0, s_1\}$.

Verification Phase: Upon getting invoked with the command $(\mathsf{Open}, \mathsf{ssid}, \mathsf{P}, m', \pi', d')$, the V verifies the decommitments $d = \{m, r, s_0, s_1\}$ corresponding to commitment $\{c, E_0, E_1\}$ as follows:
– Abort if $c \neq \mathsf{Com}(\mathsf{crs}_{\mathsf{Com}}, m; r)$.
– Abort if $E_m \neq \mathsf{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); s_m)$.
– Abort if $E_{\overline{m}} \neq \mathsf{PKE.oEnc}(\mathsf{pk}; s_{\overline{m}})$.
The verifier outputs (verified, ssid, P) if the above checks verify. Else, output (verification-failed, ssid, P).

Proof. We provide an overview of our security proof before proceeding to the formal proof details. The simulation algorithm is provided in Fig. 6. The simulator $\mathcal{S}$ constructs the $\mathcal{S}_C$ algorithm. Upon being invoked with Com command, $\mathcal{S}_C$ algorithm constructs a commitment in equivocal mode. This is performed by constructing c in equivocal mode with randomness r_0 and r_1 corresponding to bit messages 0 and 1. E_0 and E_1 are correct encryptions of r_0 and r_1 with randomness s_0 and s_1. $\mathcal{S}_C$ equivocates the commitment to open m when invoked with Equiv command and m as argument. This is performed by returning r_m and s_m as the randomness for c and E_m. It claims that $E_{\overline{m}}$ was obliviously sampled by running the randomness inversion algorithm Inv on $E_{\overline{m}}$. Indistinguishability follows due to the equivocal property of the commitment scheme, IND-CCA-2 property and the oblivious ciphertext sampleability of the encryption scheme. CCA-2 security ensures that a simulated commitment/opening generated by the simulator, when the committer is honest, in a subsession cannot be reused by a corrupt committer in a different subsession. The Com and Equiv are useful for simulation when the committer is honest and the commitment is returned by the ideal functionality. The commitment is constructed in equivocal mode and is later equivocated by $\mathcal{S}_C$ to open to the functionality provided message m.

Next, $\mathcal{S}_C$ can also be used to extract a committed message from a valid commitment upon invocation with command Ext. $\mathcal{S}_C$ decrypts E_0 and E_1 to obtain candidate randomness r_0 and r_1. Then it reconstructs c using $(0, r_1)$ and $(1, r_1)$. If both reconstructions fail then commitment c is invalid and $\mathcal{S}_C$ outputs $m = \bot$. If both reconstructions succeed then the corrupt committer has broken the binding of the commitment scheme and hence $\mathcal{S}_C$ outputs $m = \bot$. Else if $c == \mathsf{Com}(b; r_b)$ then $\mathcal{S}_C$ extracts $m = b$. This is returned by $\mathcal{S}_C$ as the extracted message.

Ideal Functionality:
The ideal functionality of $\mathcal{F}_{\mathsf{NICOM}}$ (Fig. 2) is invoked in the ideal world.

The simulator $\mathcal{S}$ forwards messages between the environment $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$. $\mathcal{S}$ constructs $\mathcal{S}_C$ algorithm as follows.

$\mathcal{S}_C$ **Algorithm:**
$\mathcal{S}$ hardcodes the trapdoors $\mathsf{td} = (\mathsf{td}_{\mathsf{Com}}, \mathsf{sk})$ inside the $\mathcal{S}_C$ algorithm. $\mathcal{S}$ constructs the $\mathcal{S}_C$ algorithm corresponding to different commands as follows:

- $(\mathsf{Com}, \mathsf{ssid}, \mathsf{P})$: $\mathcal{S}_C$ samples randomness $r_0 \leftarrow \{0,1\}^*$ and computes $c = \mathsf{Com}(\mathsf{crs}_{\mathsf{Com}}, 0; r_0)$. $\mathcal{S}_C$ also obtains $r_1 = \mathsf{Equiv}(1, r_0, c, \mathsf{td}_{\mathsf{Com}})$. $\mathcal{S}_C$ samples $s_0, s_1 \leftarrow \{0,1\}^*$. $\mathcal{S}_C$ computes $E_0 = \mathrm{PKE}.\mathsf{Enc}(\mathsf{pk}, (r_0, \mathsf{ssid}, \mathsf{P}); s_0)$ and $E_1 = \mathrm{PKE}.\mathsf{Enc}(\mathsf{pk}, (r_1, \mathsf{ssid}, \mathsf{P}); s_1)$. $\mathcal{S}_C$ sets $\pi = \{c, E_0, E_1\}$ and $\mathsf{st} = (r_0, r_1, s_0, s_1)$. $\mathcal{S}_C$ returns (π, st).

- $(\mathsf{Equiv}, \mathsf{ssid}, \mathsf{P}, \pi, \mathsf{st}, m)$: $\mathcal{S}_C$ parses $\pi = (\pi, E_0, E_1)$ and $\mathsf{st} = (r_0, r_1, s_0, s_1)$, and computes $s'_m = s_m$ and $s'_{\overline{m}} = \mathrm{PKE}.\mathsf{Inv}(\mathsf{pk}, E_{\overline{m}})$. $\mathcal{S}_C$ sets $d = (m, r_m, s'_0, s'_1)$. $\mathcal{S}_C$ returns (st, d).

- $(\mathsf{Ext}, \mathsf{ssid}, \mathsf{P}, \pi')$: $\mathcal{S}_C$ parses $\pi' = (c, E_0, E_1)$. $\mathcal{S}_C$ computes $(r'_0, \mathsf{id}') = \mathrm{PKE}.\mathsf{Dec}(\mathsf{sk}, E_0)$ and $(r'_1, \mathsf{id}'') = \mathrm{PKE}.\mathsf{Dec}(\mathsf{sk}, E_1)$. $\mathcal{S}_C$ sets $(m = \perp, \mathsf{st} = \perp)$ if 1) $\mathsf{id}' \neq (\mathsf{ssid}, \mathsf{P})$ and $\mathsf{id}'' \neq (\mathsf{ssid}, \mathsf{P})$, 2) $c == \mathsf{Com}(0; r_0) == \mathsf{Com}(1; r_1)$, or 3) $c \neq \mathsf{Com}(0; r_0)$ and $c \neq \mathsf{Com}(1; r_1)$. Else, $\mathcal{S}_C$ sets $(m = b, \mathsf{st} = r_b)$ where $c == \mathsf{Com}(b; r_b)$. $\mathcal{S}_C$ returns (m, st).

- $(\mathsf{Verify}, \mathsf{ssid}, \mathsf{P}, \pi', d', \mathsf{st})$: $\mathcal{S}_c$ parses $\pi' = (c', E'_0, E'_1)$, $d' = (m', r', s'_0, s'_1)$ and $\mathsf{st} = r$. $\mathcal{S}_C$ sets $v = \mathsf{verified}$ if 1) $r == r'$, 2) $c' == \mathsf{Com}(m'; r')$, 3) $E_m == \mathrm{PKE}.\mathsf{Enc}(\mathsf{pk}, (r', \mathsf{ssid}, \mathsf{P}); s'_m)$, and 4) $E_{\overline{m}} == \mathrm{PKE}.\mathsf{oEnc}(\mathsf{pk}, s'_{\overline{m}})$. Else, $\mathcal{S}_C$ sets $v = \mathsf{verification\text{-}failed}$. $\mathcal{S}$ returns v.

$\mathcal{S}$ activates $\mathcal{F}_{\mathsf{NICOM}}$ by sending the $\mathcal{S}_C$ algorithm.

When $\mathcal{Z}$ instructs (via the dummy adversary $\mathcal{A}$) the ideal-world adversary to corrupt an honest committer in subsession ssid, the simulator $\mathcal{S}$ invokes the $\mathcal{F}_{\mathsf{NICOM}}$ functionality with command $(\mathsf{Corrupt}, \mathsf{ssid})$. $\mathcal{S}$ receives a tuple of the form $(\mathsf{ssid}, m, \pi, d, \mathsf{st})$ or $(\mathsf{ssid}, \perp)$ from $\mathcal{F}_{\mathsf{NICOM}}$. $\mathcal{S}$ forwards the tuple to the dummy adversary $\mathcal{A}$. At the end of the protocol, $\mathcal{A}$ forwards it view to $\mathcal{S}$ which is forwarded to $\mathcal{Z}$ as the ideal world adversary view.

The Verify command of $\mathcal{S}_C$ allows to verify that a provided decommitment is valid. When the environment $\mathcal{Z}$ issues corruption request for the honest committer via the dummy adversary $\mathcal{A}$, the simulator algorithm issues corruption request to the ideal functionality and receive the internal state of the committer. This is returned to the environment via dummy adversary $\mathcal{A}$.

We argue that the real world adversary view is indistinguishable from the ideal world adversary view via a sequence of hybrids. We modify the ideal functionality and the $\mathcal{S}_C$ algorithm in the hybrids to argue indistinguishability between the consecutive pair of hybrids. We provide the indistinguishability arguments as follows.

- $\mathtt{Hyb}_4$: This is the ideal world execution of the protocol as demonstrated in Fig. 6.
- $\mathtt{Hyb}_3$: The hybrid (ideal functionality, $\mathcal{S}_C$ algorithm and the simulator) can be found in Fig. 7.

  Indistinguishability: This is identically distributed to $\mathtt{Hyb}_4$ (Fig. 6) as it only involves assembling the $\mathcal{S}_C(\mathsf{Com}, \mathsf{ssid})$ and $\mathcal{S}_C(\mathsf{Equiv}, \mathsf{ssid}, \pi, \mathsf{st}, m)$ steps from $\mathtt{Hyb}_4$ into the protocol step $\mathcal{S}_C(\mathsf{Com}, \mathsf{ssid}, m)$.
- $\mathtt{Hyb}_2$: This is same as $\mathtt{Hyb}_3$, except in commitment phase, E_m and $E_{\overline{m}}$ encrypt the same commitment randomness r corresponding to $c = \mathsf{Com}(m; r)$. The hybrid can be found in Fig. 8.

  Indistinguishability: A distinguisher $\mathcal{D}_{23}$ for the two hybrids can be used to break CCA2 security of the encryption scheme. The adversary for the CCA2 game uses $\mathcal{D}_{23}$. In the com-

Fig. 7. Hybrid Hyb_3

Ideal Functionality:

- **Commit:** On input $(\mathsf{Com}, \mathsf{ssid}, \mathsf{P}, m)$ from committer P: Obtain commitment π, decommitment d and internal state st as $(\pi, d, \mathsf{st}) \leftarrow \mathcal{S}_C(\mathsf{Com}, \mathsf{ssid}, \mathsf{P}, m)$. Store $(\mathsf{ssid}, \mathsf{P}, m, \pi, d, \mathsf{st})$ and output $(\mathsf{Receipt}, \mathsf{ssid}, \mathsf{P}, \pi, d)$ to P. Ignore future execution requests - $(\mathsf{Com}, \mathsf{ssid}, \cdot)$, with the same ssid.
- **Open:** Same as the ideal functionality $\mathcal{F}_{\mathsf{NICOM}}$ in Hyb_4 (Fig. 6).
- **Corruption:** Same as the ideal functionality $\mathcal{F}_{\mathsf{NICOM}}$ in Hyb_4 (Fig. 6).

The simulator $\mathcal{S}$ forwards messages between the environment $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$. $\mathcal{S}$ constructs $\mathcal{S}_C$ algorithm as follows.

$\mathcal{S}_C$ **Algorithm:**
$\mathcal{S}$ hardcodes the trapdoors $\mathsf{td} = (\mathsf{td}_{\mathsf{Com}}, \mathsf{sk})$ inside the $\mathcal{S}_C$ algorithm. $\mathcal{S}$ constructs the $\mathcal{S}_C$ algorithm corresponding to different commands as follows:

- $(\mathsf{Com}, \mathsf{ssid}, \mathsf{P}, m)$: $\mathcal{S}_C$ samples randomness $\widetilde{r} \leftarrow \{0,1\}^*$ and computes $c = \mathsf{Com}(\mathsf{crs}_{\mathsf{Com}}, 1 - m; \widetilde{r})$. $\mathcal{S}_C$ computes $r = \widetilde{\mathsf{Equiv}}(m, \widetilde{r}, c, \mathsf{td}_{\mathsf{Com}})$. $\mathcal{S}_C$ samples $\widetilde{s_0}, \widetilde{s_1} \leftarrow \{0,1\}^*$. $\mathcal{S}_C$ computes $E_m = \mathrm{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); \widetilde{s_m})$ and $E_{1-m} = \mathrm{PKE.Enc}(\mathsf{pk}, (\widetilde{r}, \mathsf{ssid}); \widetilde{s_{1-m}})$. Sets $s_m = \widetilde{s_m}$ and $s_{1-m} = \mathrm{PKE.Inv}(\mathsf{pk}, E_{1-m})$. $\mathcal{S}_C$ sets $\pi = \{c, E_0, E_1\}$, $d = (m, r, s_0, s_1)$ and $\mathsf{st} = (r, s_0, s_1)$. $\mathcal{S}_C$ returns (π, d, st).
- $(\mathsf{Ext}, \mathsf{ssid}, \mathsf{P}, \pi')$: Same as Hyb_4.
- $(\mathsf{Verify}, \mathsf{ssid}, \mathsf{P}, \pi', d', \mathsf{st})$: Same as Hyb_4.

$\mathcal{S}$ activates the above ideal functionality by sending the $\mathcal{S}_C$ algorithm.

When $\mathcal{Z}$ instructs (via the dummy adversary $\mathcal{A}$) the ideal-world adversary to corrupt an honest committer in subsession ssid the simulator $\mathcal{S}$ corrupts the honest party similar to Hyb_4 by invoking the above ideal functionality with $(\mathsf{Corrupt}, \mathsf{ssid})$. Upon obtaining the state of the honest party, $\mathcal{S}$ forwards the tuple to $\mathcal{A}$. At the end of the protocol, $\mathcal{A}$ forwards its view to $\mathcal{S}$, who forwards it to $\mathcal{Z}$.

Fig. 8. Hybrid Hyb_2

Ideal Functionality:
Same as the ideal functionality in Hyb_3 (Fig. 7).

The simulator $\mathcal{S}$ forwards messages between the environment $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$. $\mathcal{S}$ constructs $\mathcal{S}_C$ algorithm as follows.

$\mathcal{S}_C$ **Algorithm:**
$\mathcal{S}$ hardcodes the trapdoors $\mathsf{td} = (\mathsf{td}_{\mathsf{Com}}, \mathsf{sk})$ inside the $\mathcal{S}_C$ algorithm. $\mathcal{S}$ constructs the $\mathcal{S}_C$ algorithm corresponding to different commands as follows:

- $(\mathsf{Com}, \mathsf{ssid}, \mathsf{P}, m)$: $\mathcal{S}_C$ samples randomness $\widetilde{r} \leftarrow \{0,1\}^*$ and computes $c = \mathsf{Com}(\mathsf{crs}_{\mathsf{Com}}, 1 - m; \widetilde{r})$. $\mathcal{S}_C$ computes $r = \widetilde{\mathsf{Equiv}}(m, \widetilde{r}, c, \mathsf{td}_{\mathsf{Com}})$. $\mathcal{S}_C$ samples $\widetilde{s_0}, \widetilde{s_1} \leftarrow \{0,1\}^*$. $\mathcal{S}_C$ computes $E_0 = \mathrm{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); \widetilde{s_0})$ and $E_1 = \mathrm{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); \widetilde{s_1})$. Sets $s_m = \widetilde{s_m}$ and $s_{\overline{m}} = \mathrm{PKE.Inv}(\mathsf{pk}, E_{\overline{m}})$. $\mathcal{S}_C$ sets $\pi = \{c, E_0, E_1\}$, $d = (m, r, s_0, s_1)$ and $\mathsf{st} = (r, s_0, s_1)$. $\mathcal{S}_C$ returns (π, d, st).

- $(\mathsf{Ext}, \mathsf{ssid}, \mathsf{P}, \pi')$: Same as Hyb_3.
- $(\mathsf{Verify}, \mathsf{ssid}, \mathsf{P}, \pi', d', \mathsf{st})$: Same as Hyb_3.

$\mathcal{S}$ activates the above ideal functionality by sending the $\mathcal{S}_C$ algorithm.

When $\mathcal{Z}$ instructs (via the dummy adversary $\mathcal{A}$) the ideal-world adversary to corrupt an honest committer in subsession ssid the simulator $\mathcal{S}$ corrupts the honest party similar to Hyb_3 by invoking the above ideal functionality with $(\mathsf{Corrupt}, \mathsf{ssid})$. Upon obtaining the state of the honest party, $\mathcal{S}$ forwards the tuple to $\mathcal{A}$. At the end of the protocol, $\mathcal{A}$ forwards its view to $\mathcal{S}$, who forwards it to $\mathcal{Z}$.

mitment phase, when the adversary receives a commitment request to bit m, it constructs

---

Ideal Functionality:
Same as the ideal functionality in Hyb$_2$ (Fig. 8).

The simulator $\mathcal{S}$ forwards messages between the environment $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$. $\mathcal{S}$ constructs $\mathcal{S}_C$ algorithm as follows.

$\mathcal{S}_C$ **Algorithm:**
$\mathcal{S}$ hardcodes the trapdoors $\mathsf{td} = (\mathsf{td}_{\mathsf{Com}}, \mathsf{sk})$ inside the $\mathcal{S}_C$ algorithm. $\mathcal{S}$ constructs the $\mathcal{S}_C$ algorithm corresponding to different commands as follows:

- $(\mathsf{Com}, \mathsf{ssid}, \mathsf{P}, m) : \mathcal{S}_C$ samples randomness $r \leftarrow \{0,1\}^*$ and computes $c = \mathsf{Com}(\mathsf{crs}_{\mathsf{Com}}, m; r)$. $\mathcal{S}_C$ samples $\widetilde{s_0}, \widetilde{s_1} \leftarrow \{0,1\}^*$. $\mathcal{S}_C$ computes $E_0 = \mathrm{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); \widetilde{s_0})$ and $E_1 = \mathrm{PKE.Enc}(\mathsf{pk}, (r, \mathsf{ssid}, \mathsf{P}); \widetilde{s_1})$. Sets $s_m = \widetilde{s_m}$ and $s_{\overline{m}} = \mathrm{PKE.Inv}(\mathsf{pk}, E_{\overline{m}})$. $\mathcal{S}_C$ sets $\pi = \{c, E_0, E_1\}$, $d = (m, r, s_0, s_1)$ and $\mathsf{st} = (r, s_0, s_1)$. $\mathcal{S}_C$ returns (π, d, st).
- $(\mathsf{Ext}, \mathsf{ssid}, \mathsf{P}, \pi') :$ Same as Hyb$_2$.
- $(\mathsf{Verify}, \mathsf{ssid}, \mathsf{P}, \pi', d', \mathsf{st}) :$ Same as Hyb$_2$.

$\mathcal{S}$ activates the above ideal functionality by sending the $\mathcal{S}_C$ algorithm.

When $\mathcal{Z}$ instructs (via the dummy adversary $\mathcal{A}$) the ideal-world adversary to corrupt an honest committer in subsession ssid the simulator $\mathcal{S}$ corrupts the honest party similar to Hyb$_2$ by invoking the above ideal functionality with $(\mathsf{Corrupt}, \mathsf{ssid})$. Upon obtaining the state of the honest party, $\mathcal{S}$ forwards the tuple to $\mathcal{A}$. At the end of the protocol, $\mathcal{A}$ forwards its view to $\mathcal{S}$, who forwards it to $\mathcal{Z}$.

$c = \mathsf{Com}(m; r) = \mathsf{Com}(1 - m; \widetilde{r})$. E_m encrypts r with randomness s_m. To construct E_{1-m}, query the CCA2 challenger with messages $(r, \widetilde{r})$ to receive a ciphertext E^*. Set $E_{1-m} = E^*$. Return the commitment to m as $\pi = (c, E_0, E_1)$ and record the tuple. In the opening phase if $\mathcal{D}_{23}$ sends an opening (m', π', d') then parse $\pi' = (c, E'_0, E'_1)$ and proceed as follows:

- If E_0 and E_1 were not returned as challenge ciphertexts in the CCA-2 game, then invoke the decryption oracle to decrypt them to obtain candidate randomness values and run the simulation algorithm similar to $\mathcal{S}_C(\mathsf{Ext}, \mathsf{ssid}, \pi')$ to extract a message m''.
- If E_0 or E_1 were returned as challenge ciphertexts in the CCA-2 game, then it was returned as a commitment by an honest committer previously and got corrupted later on. If this has been a different party than P then ignore the adversary's message. Else, (the encryption(s) appeared in a commitment made by P before P got corrupted) recall the corresponding stored bit m'' from the previous commitment.

If $m' \neq m''$ and the verification using $\mathcal{S}_C(\mathsf{Verify}, \mathsf{ssid}, \pi', d', \mathsf{st})$ succeeds then the CCA2 adversary outputs 1. Else, $\mathcal{D}_{23}$ outputs bit b which is forwarded by the CCA-2 adversary to the challenger of the CCA-2 game.
When the challenge bit of the CCA-2 game is 0 (resp. 1), E^* is the encryption of r ($\widetilde{r}$) and $\mathcal{D}_{23}$ is in Hyb$_2$ (resp. Hyb$_3$). Thus, when the challenge bit is 0 (resp. 1) the adversary outputs $b = 1$ with the same probability when $\mathcal{D}_{23}$ is in Hyb$_2$ (resp. Hyb$_3$). Hence, the distinguishing advantage of $\mathcal{D}_{23}$ gets translated into the advantage of the CCA-2 adversary.
- Hyb$_1$: This is same as Hyb$_2$, except the commitment is honestly generated corresponding to message m and randomness r. The hybrid can be found in Fig. 9.

Indistinguishability: A distinguisher $\mathcal{D}_{12}$ for the two hybrids can be used to break the equivocal property of the commitment scheme. The adversary for the equivocal property invokes the challenger of the game with input $(m, \overline{m})$ to obtain the commitment c and randomness $\widetilde{r}$. $\widetilde{r}$ is encrypted in E_0 and E_1. $\pi = (c, E_0, E_1)$ is provided as the commitment. To open the commitment π the adversary provides (m, r, s_0, s_1) as the decommitment where s_{1-m} is computed by the inverting the encryption randomness of E_{1-m}. $\mathcal{D}_{12}$ outputs bit b which

is forwarded by the adversary to the challenger of the equivocal property game. If c was honestly generated (resp. generated via equivocation) then the $\mathcal{D}_{12}$ receives the view of the $\mathtt{Hyb}_1$ (resp. $\mathtt{Hyb}_2$). Indistinguishability follows from the equivocal property of the commitment scheme.

- $\mathtt{Hyb}_0$: This is the real world execution of the protocol corresponding to the [CF01] protocol in Fig. 5. After rearranging the protocol steps and incorporating a modified ideal functionality in $\mathtt{Hyb}_1$, the major changes between $\mathtt{Hyb}_0$ and $\mathtt{Hyb}_1$ are highlighted as follows:

  1. In the commitment phase, both E_0 and E_1 are generated as encryptions of commitment randomness r. E_{1-m} is inverted to compute the encryption randomness s_{1-m}.
  2. In the opening phase, a corrupt prover provides a decommitment d' to a commitment $\pi' = (c', E_0', E_1')$. Here, the $\mathcal{S}_C$ algorithm decrypts both E_0' and E_1' to obtain r_0' and r_1' as candidate commitment randomness respectively. $\mathcal{S}_C$ rejects the opening if 1)$c' ==$ $\mathsf{Com}(0; r_0') == \mathsf{Com}(1; r_1')$, or 2) $c' \neq \mathsf{Com}(0; r_0')$ and $c' \neq \mathsf{Com}(1; r_1')$.

Indistinguishability: We show that a distinguisher $\mathcal{D}_{01}$ between $\mathtt{Hyb}_0$ and $\mathtt{Hyb}_1$ distinguishes with negligible probability based on the above two changes as follows:

1. If $\mathcal{D}_{01}$ can distinguish between the two hybrids based on the distribution of E_{1-m} then it can be used to break the oblivious ciphertext sampling property of the PKE. An adversary for the oblivious ciphertext sampling game invokes the challenger of the game with input r. It receives a challenge ciphertext $\widetilde{c}$ and sampling randomness $\widetilde{r}$. The adversary sets $E_{1-m} = \widetilde{c}$ and $s_{1-m} = \widetilde{r}$. This is returned to the distinguisher $\mathcal{D}_{01}$ for the hybrids. $\mathcal{D}_{01}$ outputs bit b which is forwarded by the adversary to the challenger of the oblivious ciphertext sampling game. If $\widetilde{c}$ was honestly (resp. obliviously) generated then the $\mathcal{D}_{01}$ receives the view of the $\mathtt{Hyb}_1$ (resp. $\mathtt{Hyb}_0$). Indistinguishability follows from the oblivious ciphertext sampling property if $\mathcal{D}_{01}$ does not break the binding property of the commitment scheme and the correctness of the encryption scheme.

2. $\mathcal{D}_{01}$ can distinguish between the two hybrids if it constructs c' in the equivocal mode, i.e. $c' == \mathsf{Com}(0; r_0') == \mathsf{Com}(1; r_1')$ or the encrypted randomness (in $E_{m'}$) is different from the randomness decrypted by $\mathcal{S}_C$. In the first case, the committer breaks the binding property of the commitment scheme where $(0, r_0')$ and $(1, r_1')$ can be returned as the response to the challenger of the binding property for the commitment. In the second case, the decrypted randomness from $E_{m'}$ is invalid and hence the verification fails in $\mathtt{Hyb}_1$, whereas the verification succeeds in $\mathtt{Hyb}_0$. This occurs since the committer can show that it has encrypted a different randomness which is consistent with the commitment construction in $\mathtt{Hyb}_0$. This breaks the correctness of PKE where $E_{m'}$ is the ciphertext which fails to provide correct decryption.

$\square$

We prove in Section. A that this new functionality implies the old UC commitment functionality (Fig. 19) of [CF01] but our new functionality is more compatible with non-interactive protocols.

5   Triply Adaptive NIZK Argument in the crs model

In this section, we present our NIZK protocol. First, we recall the definition of Sigma protocol in the crs model and then build upon it to define adaptively Sigma protocol in the $\mathcal{F}_{\mathsf{NICOM}}$ model. Finally, we compile adaptively Sigma protocols into NIZKs using the Fiat-Shamir transform.

5.1 Sigma Protocol

We consider Sigma protocol [CPV20] $\Sigma = (\mathsf{Setup}, \mathsf{P}_1, \mathsf{V}_1, \mathsf{P}_2, \mathsf{V}_2)$ for relation $\mathcal{R}$ between a prover P and a verifier V that receive a common input statement x. P receives an additional private input a witness w for x. The protocol has the following form:

- $\mathsf{Setup}(1^\kappa)$: The Setup algorithm runs on security parameter κ and generates a common reference string crs and a trapdoor td. The crs is published as the public setup string.
- $\mathsf{P}_1(\mathsf{crs}, x, w, 1^\kappa; \mathsf{st})$: The prover runs algorithm P_1 on common input x, crs, private input w, security parameter κ and randomness st obtaining the first message $a = \mathsf{P}_1(x, w, 1^\kappa; \mathsf{st})$ and sends a to verifier.
- $\mathsf{V}_1(\mathsf{crs}, a)$: Verifier samples random challenge $c \leftarrow_R \mathcal{C}$ and sends c to prover.
- $\mathsf{P}_2(\mathsf{crs}, x, w, \mathsf{st}, c)$: The prover runs algorithm P_2 on input $x, w, \mathsf{crs}, \mathsf{st}, c$ and obtain z. It sends z to verifier.
- $\mathsf{V}_2(\mathsf{crs}, x, a, c, z)$: The verifier outputs 1 if it accepts the proof else it outputs 0 to reject the proof.

The above protocol should satisfy completeness, honest verifier zero knowledge and special soundness. We call the property definitions of Sigma protocol as follows.

We consider Sigma protocol [CPV20] $\Sigma = (\mathsf{Setup}, \mathsf{P}_1, \mathsf{V}_1, \mathsf{P}_2, \mathsf{V}_2)$ for relation $\mathcal{R}$ between a prover P and a verifier V that receive a common input statement x. P receives an additional private input a witness w for x. The protocol has the following form:

- $\mathsf{Setup}(1^\kappa)$: The Setup algorithm runs on security parameter κ and generates a common reference string crs and a trapdoor td. The crs is published as the public setup string.
- $\mathsf{P}_1(\mathsf{crs}, x, w, 1^\kappa; \mathsf{st})$: The prover runs algorithm P_1 on common input x, crs, private input w, security parameter κ and randomness st obtaining the first message $a = \mathsf{P}_1(x, w, 1^\kappa; \mathsf{st})$ and sends a to verifier.
- $\mathsf{V}_1(\mathsf{crs}, a)$: The verifier samples a random challenge $c \leftarrow_R \mathcal{C}$ and sends c to prover.
- $\mathsf{P}_2(\mathsf{crs}, x, w, \mathsf{st}, c)$: The prover runs algorithm P_2 on input $x, w, \mathsf{crs}, \mathsf{st}, c$ and obtain z. It sends z to verifier.
- $\mathsf{V}_2(\mathsf{crs}, x, a, c, z)$: The verifier outputs 1 if it accepts the proof else it outputs 0 to reject the proof.

The above protocol should satisfy completeness, honest verifier zero knowledge and special soundness.

Definition 20. *A 3-move protocol $\Sigma = (a, c, z)$ over corresponding domains $(\mathcal{A}, \mathcal{C}, \mathcal{Z})$ is a Sigma protocol in the crs model for a relation $\mathcal{R}$ if it satisfies the following properties:*

1. **Completeness.** *If $(x, w) \in \mathcal{R}$, then all honest 3-move transcripts for (x, w) are accepting.*
2. **Special soundness.** *There exists an efficient algorithm Ext that, on input two accepting transcripts (a, c, z) and (a, c', z') for x with $c \neq c' \in \mathcal{C}$ outputs a witness w such that $(x, w) \in \mathcal{R}$.*
3. **Honest-verifier zero knowledge (HVZK).** *There exists a PPT simulator algorithm $\mathcal{S}$ that takes as input the setup string crs, statement $x \in \mathcal{L}$, a challenge random $c \leftarrow_R \mathcal{C}$ and trapdoor td of crs, and outputs an accepting transcript (a, z) for x and an internal state st. The simulated transcript is computationally indistinguishable from an honestly generated proof. More formally, the following holds for every PPT adversary $\mathcal{A}$:*

$$\left| \Pr\left[\mathcal{A}(a, c, z, x) = 1 | a \leftarrow \mathsf{P}_1(\mathsf{crs}, x, w, 1^\kappa; \mathsf{st}), c \leftarrow_R \mathcal{C}, z \leftarrow \mathsf{P}_2(\mathsf{crs}, x, w, \mathsf{st}, c) \right] - \right.$$

$$\Pr\left[\mathcal{A}(a,c,z,x) = 1 | c \leftarrow_R \mathcal{C}, (a,z,\textsf{st}) \leftarrow \mathcal{S}(\textsf{crs},x,c,\textsf{td})\right]\bigg| \leq neg(\kappa),$$

where $(\textsf{crs},\textsf{td}) \leftarrow \textsf{Setup}(1^\kappa)$.

5.2   Fully Adaptive Sigma Protocol in $\mathcal{F}_{\textsf{NICOM}}$ model

The traditional Sigma protocols are not secure against adaptive corruption of parties. Hence, we introduce the notion of fully adaptive Sigma protocols in the UC-commitment functionality $\mathcal{F}_{\textsf{NICOM}}$ model. Consider the above Sigma protocol transcript (a,c,z). In the fully adaptive Sigma protocol, the prover has access to the $\mathcal{F}_{\textsf{NICOM}}$ functionality while computing the first message a. The prover sends a to the verifier. Upon obtaining the challenge c, the prover computes the response z and sends it to the verifier.

Definition 21. *Let* $\Sigma = (\textsf{Setup}, P_1, V_1, P_2, V_2)$ *be a Sigma protocol for relation* $\mathcal{R}$ *over corresponding domains* $(\mathcal{A}, \mathcal{C}, \mathcal{Z})$, *where parties make use of an instance of* $\mathcal{F}_{\textsf{NICOM}}$ *where the prover is the commiter, and where the first message consists exclusively of a sequence of commitment strings that the prover obtains from* $\mathcal{F}_{\textsf{NICOM}}$. *Then* Σ *is* fully adaptive *in the* $\mathcal{F}_{\textsf{NICOM}}$ *model if the following requirements hold:*

1. **Completeness.** *If* $(x,w) \in \mathcal{R}$, *then honest transcripts of the form* (x,a,c,z) *obtained by the verifier for* (x,w) *are accepting.*
2. **Computational Special soundness.** *There exists a PPT algorithm* $\textsf{Ext}$ *such that for any polytime adversarial prover* P^* *and two transcripts* (a,c,z) *and* (a,c',z'), *such that* $P^*(\kappa) \to (S_C, x, a)$ *where* S_C *is the adversarial code used by* $\mathcal{F}_{\textsf{NICOM}}$, $P^*(\kappa,c) \to z$, $P^*(\kappa,c') \to z'$, $c' \neq c$, *and such that the verifier accepts both transcripts when given access to* $\mathcal{F}_{\textsf{NICOM}}^{S_C}$, *it holds that:*

$$\Pr[\textsf{Ext}(\textsf{crs}, S_C, x, a, c, z, c', z') = w \ \& \ (x,w) \notin \mathcal{R}] < neg(\kappa)$$

3. **Adaptive Honest-verifier zero knowledge.** *There exists PPT algorithm* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that, for any* $(x,w) \in \mathcal{R}$, *any PPT distinguisher* $\mathcal{A}$, *and any PPT adversarial code* S_C *for* $\mathcal{F}_{\textsf{NICOM}}$:

$$\bigg| \Pr\left[(a,c,z,st) \leftarrow \mathcal{S}_1(\textsf{crs}, S_C, x; \textsf{td}), r \leftarrow \mathcal{S}_2(st,w) : \mathcal{A}^{\mathcal{F}_{\textsf{NICOM}}^{S_C}}(a,c,z,r) = 1\right] -$$

$$\Pr\left[r \leftarrow \{0,1\}^\kappa, (a,st) \leftarrow P_1^{\mathcal{F}_{\textsf{NICOM}}^{S_C}}(x,w,r), c \leftarrow \mathcal{C}, z \leftarrow P_2^{\mathcal{F}_{\textsf{NICOM}}^{S_C}}(x,w,st,c) :$$

$$\mathcal{A}^{\mathcal{F}_{\textsf{NICOM}}^{S_C}}(a,c,z,r) = 1\right]\bigg| \leq neg(\kappa)$$

where $(\textsf{crs}, \textsf{td}) \leftarrow \textsf{Setup}(1^\kappa)$.

5.3   Our NIZK Compiler in the $\mathcal{F}_{\textsf{NICOM}}$ model

We apply the Fiat-Shamir transform on the Sigma protocol using correlation intractable hash functions H to remove interaction and obtain our NIZK protocol. The CI hash function is provided with the description of the $\mathcal{S}_C$ algorithm to extract the prover's view and compute the bad challenge function. Our compiler can be found in Figure. 10, 11.

We show that our NIZK protocol $\Pi_{\textsf{NIZK}}$ implements $\mathcal{F}_{\textsf{NIZK}}$ functionality against adaptive corruption of parties by proving Thm. 4. It can be further shown that the same protocol implements the single prover multi-proof NIZK functionality $\mathcal{F}_{\textsf{NIZK}}^m$.

Fig. 10. Adaptively Secure NIZK Protocol

Π_{NIZK}

- **Primitives:** Adaptively-secure Sigma Protocol $\Sigma = (\mathsf{Setup}, \mathsf{P}_1, \mathsf{V}_1, \mathsf{P}_2, \mathsf{V}_2)$, that uses functionality $\mathcal{F}_{\mathsf{NICOM}}$ (with algorithm $\mathcal{S}_C$). Correlation Intractable hash function family $\mathcal{H} = (\mathsf{Gen}, \mathsf{StatGen}, H)$.
- **Public Inputs:** Setup string $\mathsf{crs}_{\mathsf{NIZK}} = (\mathsf{k}, \mathsf{crs}_\Sigma)$ where $(\mathsf{crs}_\Sigma, \mathsf{td}_\Sigma) \leftarrow \Sigma.\mathsf{Setup}(1^\kappa)$ and $\mathsf{k} \leftarrow \mathcal{H}.\mathsf{StatGen}(1^\kappa, C_{\mathsf{sk}})$ where $\mathsf{sk} = (\mathsf{td}_\Sigma, \mathcal{S}_C).$ [a] The Sigma protocol is repeated for $\tau = \mathcal{O}(\kappa)$ times.
- **Private Inputs:** V has input statement x. P has the same input statement x and secret witness w such that $\mathcal{R}(x, w) = 1$.

$\mathsf{P}(\mathsf{crs}_{\mathsf{NIZK}}, x, w, 1^\kappa):$
Upon invoked with command $(\mathsf{prove}, \mathsf{sid}, x, w)$ the prover performs the following for $j \in [\tau]:$

- $(a^j, \mathsf{st}_\Sigma^j) \leftarrow \Sigma.\mathsf{P}_1(\mathsf{crs}_\Sigma, x, w, 1^\kappa).$
- Sample $c_0^j, c_1^j \leftarrow_R \mathcal{C}$ such that $c_0^j \neq c_1^j$. Commit to challenges as $(\mathsf{Receipt}, C^j, \mathsf{st}_C^j) \leftarrow \mathcal{F}_{\mathsf{NICOM}}(\mathsf{Com}, 3j + 2, (c_0^j, c_1^j)).$
- For $\delta \in \{0, 1\}$, the prover performs the following:
    - Compute $z_\delta^j \leftarrow \Sigma.\mathsf{P}_2(\mathsf{crs}_\Sigma, x, w, \mathsf{st}_\Sigma^j, c_\delta^j).$
    - Commit to the responses as follows: $(\mathsf{Receipt}, 3j + \delta, \mathsf{P}, Z_\delta^j, \mathsf{st}_{Z,\delta}^j) \leftarrow \mathcal{F}_{\mathsf{NICOM}}(\mathsf{Com}, 3j + \delta, z_\delta^j).$
- The commitments for the jth run are denoted as $Y^j = (C^j, Z_0^j, Z_1^j).$

Assemble the commitments as $\mathbf{Y} = \{Y^j\}_{j \in [\tau]}$ and and the first messages as $\mathbf{a} = \{a^j\}_{j \in [\tau]}$. Compute the challenge as $\mathbf{e} = \{e^j\}_{j \in [\tau]} = H(\mathsf{k}, (\mathbf{a}, \mathbf{Y}))$. The prover performs the following for $j \in [\tau]$:

- Set the challenge as $\delta = e^j \in \{0, 1\}.$
- Construct the response as $U^j = (c_0^j, c_1^j, z_\delta^j, \mathsf{st}_C^j, \mathsf{st}_{Z,\delta}^j)$ by decommitting to the challenges and the response $z_\delta^j.$

The prover sends the NIZK proof $\gamma = (\mathbf{a}, \mathbf{Y}, \mathbf{U})$ where $\mathbf{U} = \{U^j\}_{j \in [\tau]}$ to the verifier.

---

[a] C_{sk} is a poly-size circuit computing the function $f_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y}) = \mathbf{e}$, such that for every $j \in [\tau], e^j = 0$ iff $\Sigma.\mathsf{V}_2(\mathsf{crs}_\Sigma, x, a^j, c_0^j, z^j) = 1$ where $(c_0^j, c_1^j) \leftarrow \mathcal{F}_{\mathsf{NICOM}}(\mathcal{S}_C, \mathsf{Ext}, 3j + 2, C^j), z^j \leftarrow \mathcal{F}_{\mathsf{NICOM}}(\mathcal{S}_C, \mathsf{Ext}, 3j, Z_0^j).$

Fig. 11. Adaptively Secure NIZK Protocol (Cont.)

Π_{NIZK}

$\mathsf{V}(\mathsf{crs}_{\mathsf{NIZK}}, x, \gamma):$
Upon invoked with command $(\mathsf{verify}, \mathsf{sid}, x, \gamma)$ the verifier performs the following:

- Parse the proof $\gamma = (\mathbf{a}, \mathbf{Y}, \mathbf{U}) = \{a^j, Y^j, U^j\}_{j \in [\tau]}.$
- Compute the challenge string as $\mathbf{e} = \{e^j\}_{j \in [\tau]} = \mathcal{H}.H(\mathsf{k}, (\mathbf{a}, \mathbf{Y}))$ where $\mathbf{e} \in \{0, 1\}^\tau.$
- For $j \in [\tau]$, the verifier performs the following :
    - The verifier sets $\delta = e^j \in \{0, 1\}.$
    - Parse the proof as $Y^j = (C^j, Z_0^j, Z_1^j)$ and $U^j = (c_0^j, c_1^j, z^j, \mathsf{st}_Z^j, \mathsf{st}_C^j).$
    - Verifies the provided decommitments and proofs. Output $(\mathsf{verification}, \mathsf{sid}, x, \gamma, 0)$ if any of the following occurs:
        1. If $\mathcal{F}_{\mathsf{NICOM}}(\mathsf{Open}, 3j + 2, (c_0^j, c_1^j), C^j, \mathsf{st}_C^j)$ returns $\mathsf{verification\text{-}failed}.$
        2. If $\mathcal{F}_{\mathsf{NICOM}}(\mathsf{Open}, 3j + \delta, z^j, Z_\delta^j, \mathsf{st}_Z^j)$ returns $\mathsf{verification\text{-}failed}.$
        3. If $\Sigma.\mathsf{V}_2(\mathsf{crs}_\Sigma, x, a^j, c_\delta^j, z^j) = 0.$

The verifier outputs $(\mathsf{verification}, \mathsf{sid}, x, \gamma, 1)$ if all the above τ proofs verified correctly and the above decommitments were correct.

Theorem 4. *If $\mathcal{H}$ is a somewhere statistically correlation intractable hash function family with programmability, $\Sigma = (\textsf{Setup}, \textsf{P}_1, \textsf{V}_1, \textsf{P}_2, \textsf{V}_2)$ is an adaptively secure Sigma protocol (in the $\mathcal{F}_{\textsf{NICOM}}$ model) with computational special soundness then $\Pi_{\textsf{NIZK}}$ implements $\mathcal{F}_{\textsf{NIZK}}$ functionality in ($\textsf{crs}_{\textsf{NIZK}}$, $\mathcal{F}_{\textsf{NICOM}}$) model against adaptive corruption of parties.*

Proof. The hash key $\textsf{k}$ of H function is generated in the statistical mode by running the $\textsf{StatGen}$ algorithm on $\mathcal{C}_{\textsf{sk}}$, where $\textsf{sk}$ contains the trapdoors to extract the committed values in $\mathbf{Y}$. $\mathcal{C}$ is a poly-size circuit that takes $\mathbf{Y}$ (of Σ) as input, computing the function $f_{\textsf{sk}}(\mathbf{a}, \mathbf{Y}) = \mathbf{e}$ s.t. for every $j \in [\tau]$, $e^i = 0$ iff the proof computed corresponding to challenge c_0^j is valid. The hash function is somewhere statistical correlation intractable if the commitments are binding and the Sigma protocol satisfies special soundness. It ensures that if a corrupt prover constructs an accepting proof then a valid witness can be extracted by relying on the special soundness property of the Sigma protocol. For zero-knowledge, the simulator invokes the HVZK simulator $\Sigma.\mathcal{S}_1$ of the Sigma protocol to construct a simulated proof. It commits to it for both branches. For post-execution corruption of prover, it again invokes the $\Sigma.\mathcal{S}_2$ algorithm with a correct witness to compute a valid first message and then simulate the jth proof for the other branch. It relies on the equivocal property of the $\mathcal{F}_{\textsf{NICOM}}$ to obtain randomness that allows equivocation of the commitments. Security follows from the adaptive security of the Sigma protocol in the $\mathcal{F}_{\textsf{NICOM}}$ model. Next, we present our formal security proof. Our proof contains two corruption cases.

Security against statically corrupt prover $\textsf{P}^*$. We provide our simulation algorithm in Fig. 12. The hybrid argument follows:

Fig. 12. Simulation of $\Pi_{\textsf{NIZK}}$ against a statically corrupt prover $\textsf{P}^*$

– **Simulator Inputs:** Simulator $\mathcal{S}_{\textsf{V}}$ has input statement x, trapdoor $\textsf{td}_{\textsf{NIZK}} = \textsf{td}_{\Sigma}$ and $\mathcal{S}_C$ algorithm of $\mathcal{F}_{\textsf{NICOM}}$, where $\textsf{td}_{\Sigma}$ is the trapdoor of Σ.

$\textsf{P}^*(\textsf{crs}_{\textsf{NIZK}}, x, 1^\kappa)$

The corrupt prover sends the NIZK proof $\gamma = (\mathbf{a}, \mathbf{Y}, \mathbf{U})$ where $\mathbf{a} = \{a^j\}_{j \in [\tau]}$, $\mathbf{Y} = \{Y^j\}_{j \in [\tau]}$ and $\mathbf{U} = \{U^j\}_{j \in [\tau]}$ to the simulated verifier.

$\mathcal{S}_{\textsf{P}}(\textsf{crs}_{\textsf{NIZK}}, x, \mathcal{S}_C, 1^\kappa)$

– Parse the proof $\gamma = (\mathbf{a}, \mathbf{Y}, \mathbf{U}) = \{a^j, Y^j, U^j\}_{j \in [\tau]}$.
– For $j \in [\tau]$, the verifier performs the following :
   • Parse the proof $Y^j = (C^j, Z_0^j, Z_1^j)$.
   • Extracts $(c_0^j, c_1^j) = \mathcal{S}_C(\textsf{Ext}, 3j, C^j)$.
   • For $\delta \in \{0, 1\}$, the simulator extracts $z_\delta^j = \mathcal{S}_C(\textsf{Ext}, 3j + \delta, Z_\delta^j)$.
   • If $\Sigma.\textsf{V}_2(\textsf{crs}_\Sigma, x, a^j, c_0^j, z_0^j) == \Sigma.\textsf{V}_2(\textsf{crs}_\Sigma, x, a^j, c_1^j, z_1^j) == 1$ then the simulator extracts w as follows:

$$w = \Sigma.\textsf{Ext}(\mathcal{S}_C, x, a^j, c_0^j, c_1^j, z_0^j, z_1^j).$$

   The simulator aborts if $\Sigma.\textsf{Ext}$ fails to extract a valid witness.
– The simulator runs the honest verifier algorithm. The simulator outputs 0 if the verifier algorithm outputs 0.
– The simulator aborts if the decommitted values (checked by the honest verifier algorithm) in the proof for the commitments are different from the values extracted from the commitments in the previous step.
– If the simulator has extracted a valid witness w from any of the above τ proofs then it invokes $\mathcal{F}_{\textsf{NIZK}}$ functionality with witness w, else it aborts the protocol.

– **Hyb$_0$:** Real world execution of the protocol.

– **Hyb$_1$:** Same as Hyb$_0$, except the simulator aborts if the decommitted values (checked by the honest verifier algorithm) in the proof for the commitments are different from the values extracted from the commitments in the simulation algorithm.

An adversary distinguishing between the two hybrids produces correct decommitments corresponding to (C^j, Z_0^j, Z_1^j) and e^j but evades extraction. This contradicts the $\mathcal{F}_{\mathsf{NICOM}}$ model for a statically corrupt committer.

– **Hyb$_2$:** Same as Hyb$_1$, except $\exists j \in [\tau]$, such that (a^j, c_0^j, z_0^j) and (a^j, c_1^j, z_1^j) are valid proofs for the jth iteration but $\Sigma.\mathsf{Ext}$ fails to extract a correct witness from the two proofs.

An adversary distinguishing between the two hybrids breaks computational special soundness of the sigma protocol.

– **Hyb$_3$:** Same as Hyb$_2$, except the simulator aborts if it fails to extract a valid witness w from any of the τ proofs.

The simulator fails to extract the correct witness when the corrupt prover P^* finds an $\mathbf{Y}$ which satisfies $f_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y}) = \mathbf{e}$, such that for every $j \in [\tau], e^j = 0$ iff $\Sigma.\mathsf{V}_2(\mathsf{crs}_\Sigma, x, a^j, c_0^j, z^j) = 1$. In these two hybrids the commitments are extractable if their decommitments are valid. Hence, the only way a corrupt prover distinguishes between the two hybrids if the proof is accepting but the simulator fails to extract a correct witness. This is possible if the adversary breaks the correlation intractability of the hash function H. However, H is statistically correlation intractable for $\mathcal{R}_{\mathsf{sk}} = \{(\mathbf{a}, \mathbf{Y}, \mathbf{e}) : \mathbf{e} = f_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y})\}$ and hence such an $(\mathbf{a}, \mathbf{Y})$ does not exist.

This completes the simulation of a statically corrupt prover. Adaptive corruption of the verifier can be trivially simulated since the verifier does not possess any input. A random tape can be returned as the verifier's internal state upon post-execution corruption.

Security against statically corrupt verifier V^*. Next, we demonstrate zero knowledge by simulating against a statically corrupt verifier. For proving ZK, we need to modify the distribution of the hash key from the statistical mode (where the hash key generation depends on the secret key sk) to a mode where it is independent of the secret key sk. We provide our simulation algorithm in Fig. 13. The hybrid argument follows:

– **Hyb$_0$:** Real world.
– **Hyb$_1$:** Same as Hyb$_0$, except the hash key k is computed by programming the hash function as follows $k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\kappa)|H(\mathsf{k}, (\mathbf{a}, \mathbf{Y})) = \mathbf{e}$ where $(\mathbf{a}, \mathbf{Y})$ is generated by running the honest prover algorithm and $\mathbf{e}$ is random. The proof γ is computed by running the honest prover algorithm.

The adversary cannot distinguish between the two hash keys due to indistinguishability between the two modes of the CI hash function. An adversary for the hash function can simulate either of the two views by running the honest prover algorithm using the witness w. A distinguisher for the hybrids successfully distinguishes between the CI modes.

– **Hyb$_2$:** Same as Hyb$_1$, except the simulator runs the honest prover algorithm with the witness w to compute $(a^j, z_{e_j}^j)$ correctly but it computes (C^j, Z_δ^j) for $\delta = 1 - e_j$ and $j \in [\tau]$ following the simulation algorithm in Fig. 13. If the prover gets corrupted post-execution then the simulator equivocates (C^j, Z_δ^j).

Indistinguishability follows from the equivocal property of the commitment strings in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

– **Hyb$_3$:** Same as Hyb$_2$, except the simulator computes according to the simulation algorithm in Fig. 13. If the prover gets corrupted post-execution then the simulator produces the internal state of the prover following the simulation algorithm.

34

Fig. 13. Simulation of Π_{NIZK} against a statically corrupt verifier V^*

– **Simulator Inputs:** Simulator $\mathcal{S}_\mathsf{V}$ has input statement x and trapdoors $\mathsf{td}_{\mathsf{NIZK}} = (\mathsf{td}_{\mathsf{Com}}, \mathsf{td}_\Sigma)$, where $\mathsf{td}_{\mathsf{Com}}$ is the trapdoor of Com and td_Σ is the trapdoor of Σ.

$\mathcal{S}_\mathsf{V}^1(\mathsf{crs}_{\mathsf{NIZK}}, x, \mathcal{S}_C, 1^\kappa)$

$(a, c, z, st) \leftarrow \mathcal{S}_1(S_C, x), r \leftarrow \mathcal{S}_2(st, w)$ The prover performs following for $j \in [\tau]$:

– Sample a random challenge $c^j \leftarrow \mathcal{C}$.
– Obtain the simulated Sigma protocol proof $(a^j, c^j, z^j, \mathsf{st}_\Sigma^j) \leftarrow \Sigma.\mathcal{S}_1(\mathsf{crs}_\Sigma, S_C, x; \mathsf{td}_\Sigma)$ by invoking the HVZK simulator.
– Commit to garbage challenge strings as $(C^j, \mathsf{st}_C^j) = \mathcal{S}_C(\mathsf{Com}, 3j)$.
– The prover commits to the HVZK proof z^j as follows: For $\delta \in \{0,1\}$, commit to z^j as $(Z_\delta^j, \mathsf{st}_{Z,\delta}^j) = \mathcal{S}_C(\mathsf{Com}, 3j + \delta)$.
– The commitments for the jth run are denoted as $Y^j = (C^j, Z_0^j, Z_1^j)$.

Assemble the commitments as $\mathbf{Y} = \{Y^j\}_{j \in [\tau]}$, first messages as $\mathbf{a} = \{a_j\}_{j \in [\tau]}$ and compute the verifier challenge as $\mathbf{e} = \{e^j\}_{j \in [\tau]} = H(\mathsf{k}, (\mathbf{a}, \mathbf{Y}))$. The simulator performs the following for $j \in [\tau]$:

– Set the verifier challenge as $\delta = e^j \in \{0,1\}$.
– Sample $c_0^j, c_1^j \leftarrow_R \mathcal{C}$ such that $c_\delta^j = c^j$ and $c_0^j \neq c_1^j$. Equivocate C^j such that it opens to (c_0^j, c_1^j) by computing $(\widehat{\mathsf{st}_C^j}, \widehat{\mathsf{st}_C^j}) = \mathcal{S}_C(\mathsf{Equiv}, 3j, C^j, \widehat{\mathsf{st}_C^j}, (c_0^j, c_1^j))$.
– Construct the response as $U^j = (a^j, c_0^j, c_1^j, z^j, \mathsf{st}_{Z,\delta}^j, \widehat{\mathsf{st}_C^j})$ by decommitting to the challenges and Sigma protocol response z^j.

The simulator sends the simulated NIZK proof $\gamma = (\mathbf{a}, \mathbf{Y}, \mathbf{U})$, where $\mathbf{U} = \{U^j\}_{j \in [\tau]}$ to the verifier. Sets the internal state as $\mathsf{st}_\mathcal{S} = \{a^j, e^j, c_0^j, c_1^j, z^j, z^j, \mathsf{st}_\Sigma^j, \widehat{\mathsf{st}_C^j}, \mathsf{st}_{Z,0}^j, \mathsf{st}_{Z,1}^j\}_{j \in [\tau]}$.

$\mathcal{S}_\mathsf{V}^2(\mathsf{crs}_{\mathsf{NIZK}}, x, w, \gamma, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{st}_\mathcal{S}, \mathcal{S}_C, 1^\kappa)$

The prover gets corrupted post-execution. The simulator obtains witness w and performs the following for $j \in [\tau]$:

– Set $\delta = 1 - e^j$.
– Compute $r^j \leftarrow \Sigma.\mathcal{S}_2(\mathsf{crs}_\Sigma, \mathsf{st}_\Sigma^j, w)$ and construct $\widehat{z_\delta^j} = \Sigma.\mathsf{P}_2(\mathsf{crs}_\Sigma, x, w, r^j, c_\delta^j)$.
– Equivocate Z_δ^j such that it opens to $\widehat{z_\delta^j}$ under randomness $(\widehat{\mathsf{st}_{Z,\delta}^j}, \widehat{\mathsf{st}_{Z,\delta}^j}) = \mathcal{S}_C(\mathsf{Equiv}, 3j + \delta, Z_\delta^j, \widehat{\mathsf{st}_{Z,\delta}^j}, \widehat{z_\delta^j})$. Set $\mathsf{st}_{Z,\delta}^j = \widehat{\mathsf{st}_{Z,\delta}^j}$ and $z_\delta^j = \widehat{z_\delta^j}$.

Return prover's internal state as $\mathsf{st}_\mathcal{S} = \{a^j, e^j, c_0^j, c_1^j, z_0^j, z_1^j, r^j, \widehat{\mathsf{st}_C^j}, \mathsf{st}_{Z,0}^j, \mathsf{st}_{Z,1}^j\}_{j \in [\tau]}$.

The simulated NIZK proof γ is indistinguishable from an honestly generated one due to HVZK property of the underlying Sigma protocol. When the prover gets corrupted post-execution, the simulator equivocates the prover's internal state using the knowledge of the witness by invoking the $\Sigma.\mathcal{S}_2$ algorithm. This equivocated prover state is indistinguishable from an original prover state due to adaptive security of the Sigma protocol. The decommitments corresponding to the equivocated commitments are generated using the $\mathcal{S}_C$ algorithm. The randomness for the commitments are indistinguishable due to the equivocal property of the commitment scheme in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

– **Hyb$_4$:** Same as Hyb$_3$, except the hash key k is generated in the statistical mode k $\leftarrow$ $\mathcal{H}.\mathsf{StatGen}(1^\kappa, \mathcal{C}_{\mathsf{sk}})$.

The adversary cannot distinguish between the two hash keys due to indistinguishability between the two modes of the CI hash function. An adversary for the hash function can simulate either of the two simulated views by running the NIZK simulator using the trapdoor $\mathsf{td}_{\mathsf{NIZK}}$. A distinguisher for the hybrids successfully distinguishes between the CI modes.

$\square$

The NIZK protocol can be made triply adaptive secure by adding adaptive soundness and adaptive zero-knowledge. The NIZK protocol satisfies adaptive soundness if the underlying Sigma protocol satisfies adaptive soundness and Com is a non-interactive UC-commitment in the non-programmable crs model. Moreover, the NIZK protocol satisfies adaptive zero knowledge if the underlying Sigma protocol satisfies adaptive zero knowledge and $\mathcal{F}_{\mathsf{NICOM}}$ is implemented using a non-interactive adaptively secure commitment in the non-programmable crs model. We demonstrate this by proving Thm. 5.

Theorem 5. *If $\mathcal{H}$ is a somewhere statistically correlation intractable hash function family, Σ=(Setup, P_1, V_1, P_2, V_2) is a Sigma protocol satisfying adaptive special soundness and adaptive zero knowledge, and $\mathcal{F}_{NICOM}$ is implemented using an adaptively secure UC commitment in the non-programmable crs_{Com} model then Π_{NIZK} satisfies adaptive soundness and adaptive zero knowledge in the (crs_{NIZK}, crs_{Com}) model.*

Proof. We implement $\mathcal{F}_{\mathsf{NICOM}}$ using a non-interactive adaptively secure commitment [CF01] scheme Com whose the crs distribution of the commitment string is identical in real and ideal world. Our $\mathsf{crs}_{\mathsf{NIZK}}$ distribution is identical in real and ideal world execution. It contains the setup string of the commitment scheme and Sigma protocol, and the hash key, i.e. $\mathsf{crs}_{\mathsf{NIZK}}(= (\mathsf{k}, \mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_\Sigma))$. The hash function is defined in the statistical mode. An adversary breaking adaptive soundness creates an accepting proof by adaptively choosing the statement $x \notin \mathcal{L}$ based on the $\mathsf{crs}_{\mathsf{NIZK}}$ distribution. If an adversary $\mathcal{A}$ breaks adaptive soundness of Π_{NIZK} then we can use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{\mathsf{COM}}$ to break the binding property of Com or an adversary $\mathcal{A}_\Sigma$ to break the computational special soundness of Σ. We show that such an adversary should be able to distinguish between the real world proof and the simulation algorithm provided in Fig. 12. In the simulation algorithm the simulator fails to extract a valid witness (since $x \notin \mathcal{L}$) and hence it always aborts, meanwhile the real world proof is accepting. Our security argument proceeds in form of hybrids:

– **Hyb$_0$:** Real world execution of the protocol.
– **Hyb$_1$:** Same as Hyb$_0$, except the simulator aborts if the decommitted values (checked by the honest verifier algorithm) in the proof for the commitments are different from the values extracted from the commitments in the simulation algorithm.

The commitment adversary $\mathcal{A}_{\mathsf{COM}}$ obtains $\mathsf{crs}_{\mathsf{Com}}$ as the setup string for the UC commitment. It samples k and crs_Σ according to protocol, and constructs $\mathsf{crs}_{\mathsf{NIZK}}$. It provides $\mathsf{crs}_{\mathsf{NIZK}}$ to $\mathcal{A}$. The adversary $\mathcal{A}$ distinguishing between the two hybrids produces correct decommitments corresponding to (C^j, Z_0^j, Z_1^j) and e^j but evades extraction. The commitment adversary $\mathcal{A}_{\mathsf{COM}}$ returns those commitments to break the UC security of the commitment scheme.

- **Hyb$_2$:** Same as $\mathtt{Hyb}_1$, except $\exists j \in [\tau]$, such that (a^j, c_0^j, z_0^j) and (a^j, c_1^j, z_1^j) are valid proofs for the jth iteration but $\Sigma.\mathsf{Ext}$ fails to extract a correct witness from the two proofs.

  The Sigma protocol adversary $\mathcal{A}_\Sigma$ obtains crs_Σ as the setup string for the Sigma protocol. It samples k and $\mathsf{crs}_{\mathsf{Com}}$ according to protocol, and constructs $\mathsf{crs}_{\mathsf{NIZK}}$. It provides $\mathsf{crs}_{\mathsf{NIZK}}$ to $\mathcal{A}$. The adversary $\mathcal{A}$ samples a statement $x \notin \mathcal{L}$ and distinguishes between the two hybrids by constructing two valid accepting transcripts for a^j corresponding to challenges c_0^j and c_1^j. The Sigma protocol witness extractor $\Sigma.\mathsf{Ext}$ always fails to extract a valid witness since x is a false statement for every possible witness w, i.e. $\forall w \in \{0,1\}^*, \mathcal{R}(x, w) = 0$. $\mathcal{A}_\Sigma$ returns $(a^j, c_0^j, z_0^j, c_1^j, z_1^j)$ to the extractor and breaks adaptive special soundness property of the Sigma protocol.

- **Hyb$_3$:** Same as $\mathtt{Hyb}_2$, except the simulator aborts if it fails to extract a valid witness w from any of the τ proofs.

  The simulator fails to extract the correct witness when the corrupt prover P^* finds an $\mathbf{Y}$ which satisfies $f_{\mathsf{sk}}(\mathbf{a}, (\mathbf{a}, \mathbf{Y})) = \mathbf{e}$, such that for every $j \in [\tau], e^j = 0$ iff $\Sigma.\mathsf{V}_2(\mathsf{crs}_\Sigma, x, a^j, c_0^j, z^j) = 1$. In these two hybrids the commitments are extractable if their decommitments are valid. Hence, the only way a corrupt prover distinguishes between the two hybrids if the proof is accepting but the simulator fails to extract a correct witness. This is possible if the adversary breaks the correlation intractability of the hash function H. However, H is statistically correlation intractable for $\mathcal{R}_{\mathsf{sk}} = \{(\mathbf{a}, \mathbf{Y}, \mathbf{e}) : \mathbf{e} = f_{\mathsf{sk}}(\mathbf{a}, \mathbf{Y})\}$ and hence such an $(\mathbf{a}, \mathbf{Y})$ does not exist.

The zero knowledge simulator for Π_{NIZK} in Fig. 13 is secure when the statement is adaptively chosen based on the $\mathsf{crs}_{\mathsf{NIZK}}$ distribution. Adaptive zero knowledge of $\Pi_{\mathsf{NIZK\text{-}FLS}}$ follows from the adaptive zero knowledge property of the Sigma protocol and the adaptive security of the UC commitment scheme. The simulation algorithm and the proof is the same as the security proof of Π_{NIZK} against a corrupt prover. $\qquad\square$

5.4 Instantiations

The adaptively Sigma protocol can be instantiated using Schnorr's protocol, Sigma protocol of FLS, Blum's Hamiltonicity protocol or the GC-based protocol of [HV16]. Detailed overview can be found in Sec. 2.4. The CI hash function can be instantiated from LWE [PS19, CCH+19], or from DDH+LPN assumption [BKM20]. This is discussed below. $\mathcal{F}_{\mathsf{NICOM}}$ is constructed from the UC-commitment scheme of [CF01] in Section. 4 by relying on equivocal commitments and CCA-2 secure public key encryption with oblivious sampleability. The equivocal commitment can be instantiated from Pedersen Commitment and the obliviously sampleable encryption scheme can be instantiated from Cramer Shoup encryption [CS98], yielding a protocol from DDH. Similarly, we can instantiate the equivocal commitment from LWE [CsW19] and the obliviously sampleable encryption scheme from LWE [MP12].

The work of [BKM20] demonstrated that given a commitment scheme Com any commit-and-open Sigma protocol Π_{Com} can be converted to different commit-and-open Sigma protocol Π'_{Com} with the following properties and their results are summarized in Thm. 6.

- If Com is extractable then Π'_{Com} is a trapdoor Sigma protocol.
- If the commitment extraction function $f_{\mathsf{td}}(\mathbf{a}) = \mathsf{Com.Ext}(\mathsf{td}, \mathbf{a})$ has probabilistic constant-degree representation, then so does the trapdoor function BadChallenge, corresponding to Π'_{Com} and, therefore $\mathcal{R}_\Sigma(\Pi_{\mathsf{Com}}')$ is probabilistically searchable by constant-degree polynomials.

Theorem 6 ([BKM20]). *Let $\Pi_{Com} = (Gen, P_1, P_2, V)$ be a trapdoor Sigma protocol for Language $\mathcal{L}$, where the output $\mathbf{a}$ of P_1 is of length $\ell = \ell(\kappa)$. Let Com be a statistically-binding extractable commitment scheme where, for any extraction trapdoor td for the commitment, the function $f_{td}(x) = Com.Ext(td, x)$ has an ϵ-probabilistic representation by c-degree polynomials, for a constant $c \in \mathbb{N}$ and $0 < \epsilon(\kappa) < 1/\ell$. Then, for any polynomial $m := m(\kappa)$, there exists a trapdoor Sigma-protocol Π'_{Com} for $\mathcal{L}$ with soundness 2^{-m} such that $\mathcal{R}_\Sigma(\Pi_{Com}')$ is ϵ'-probabilistically searchable by $6cc'$-degree polynomials, where $c' \in \mathbb{N}$ is an arbitrary constant and $\epsilon' = \ell \cdot \epsilon + 2^{-c'}$.*

Next, they showed that the Fiat-Shamir transform can be applied using a correlation approx hash function H_{Apx} to obtain a NIZK protocol. Moreover, if Π_{Com} is input-delayed then the NIZK protocol satisfies adaptive soundness and adaptive zero knowledge.

Theorem 7 ([BKM20]). *(Sufficient Conditions for NIZK for NP). The following conditions are sufficient to obtain a NIZK argument system for NP with adaptive soundness and adaptive zero-knowledge from Π'_{Com}:*

1. *A trapdoor-Sigma protocol Π_{Com} satisfying delayed-input property.*
2. *A statistically-binding extractable commitment scheme where, for any td, the function $f_{td}(x) = Com.Ext(td, x)$ has an ϵ-probabilistic representation by c-degree polynomials, for a constant $c \in \mathbb{N}$ and $0 < \epsilon(\kappa) < 1/\ell(\kappa)$ for an arbitrarily large polynomial ℓ.*
3. *A programmable correlation intractable hash family $\mathcal{H}_{Apx}$ for relations ϵ probabilistically searchable by c'-degree polynomials, for some constant $\epsilon > 0$ and arbitrarily large constant $c' \in \mathbb{N}$.*

They instantiate the correlation approx hash function H_{Apx} from trapdoor hash [DGI+19] under DDH, QR, DCR and LWE assumptions. The commitment scheme is instantiated under LPN assumption s.t. its extractor algorithm can be probabilistically represented by a constant degree polynomial. We refer to their paper [BKM20] for more details and formal definitions.

Our adaptively secure Sigma protocol with special soundness protocol can be shown to be a trapdoor Sigma protocol in the $\mathcal{F}_{\mathsf{PCOM}}$ model. $\mathcal{F}_{\mathsf{PCOM}}$ can be instantiated based on DDH assumption. The statistically-binding extractable commitment scheme with a constant degree extractor function is obtained using the LPN-based commitment scheme of [BKM20]. By applying the Fiat-Shamir transform using H_{Apx} as the hash function and applying the result of Thm. 7 we obtain a triply adaptive NIZK protocol from LPN+DDH assumption.

We provide a brief overview of adapting our compiler such that it is compatible with the approximate CI hash function. In the protocol, the verification relation of the bad challenge function is converted into a 3CNF formulae using Construction 3.1 of [BKM20] by applying the Cook-Levin theorem. The intermediate steps in the Pedersen commitment computation (which implements $\mathcal{F}_{\mathsf{PCOM}}$) to commit to the first message of the Sigma protocol are also encrypted using the LPN-based commitment scheme. Encrypting the intermediate steps in the commitment computation helps in regenerating the commitments using a 3CNF. This allows regenerating the committed view of the prover and computing the bad challenge function using a 3CNF formulae. Lemma 3.18 of [BKM20] shows that any 3CNF formulae can be approximated using a constant degree polynomial. By combining the above results and applying the lemma,

we obtain a probabilistic representation of the bad challenge function using a constant degree polynomial. This has been proven by [BKM20] in Lemma 3.19. Finally, we can apply the Fiat-Shamir transform using the CI-Apx hash function $\mathcal{H}_{\mathsf{Apx}}$ to obtain the NIZK from DDH+LPN assumption.

6   Triply Adaptive NIZK Argument in the short crs model

Let $\mathsf{crs}_{\mathsf{NIZK}}$ denote the setup string for our NIZK protocol Π_{NIZK}(Fig. 10,11). Currently the $\mathsf{crs}_{\mathsf{NIZK}}$ contains the public hash key k which depends on the circuit length. We reduce this to $\mathrm{poly}(\kappa)$ by applying a compiler which compiles any single-prover multi-proof NIZK protocol Π_{NIZK} in the $\mathsf{crs}_{\mathsf{NIZK}}$ model to a NIZK protocol Π_{sNIZK} in the short $\mathsf{crs}_{\mathsf{sNIZK}}$ model, where $|\mathsf{crs}_{\mathsf{sNIZK}}| = \mathrm{poly}(\kappa)$, assuming additively homomorphic equivocal commitment Com and a PKE with oblivious ciphertext sampling algorithm. Our compiler is inspired from the work of GOS and it can be instantiated from DDH or LWE.

Given the witness y and the statement x for a language $\mathcal{L}$, the prover computes a circuit $\mathcal{C}$ s.t. $\mathcal{C}(y) = \mathcal{R}(x, y)$ where $\mathcal{R}$ is the NP verification relation. Let $y = \{y_i\}_{i \in [|y|]}$. The prover commits to each bit y_i as $c_i = \mathsf{Com}(y_i; r_i)$ and encrypts the corresponding randomness as $e_{i,y_i} = \mathsf{Enc}(\mathsf{pk}, r_i; s_i)$ while $e_{i,\overline{y_i}}$ is sampled obliviously. The output wire is committed as $\mathsf{Com}(1; 1)$. Using Π_{NIZK} the prover proves that each c_i is a commitment to 0 or 1 and the underlying commitment randomness is also encrypted correctly in $e_{i,0}$ or $e_{i,1}$. For each jth NAND gate with input wires α and β and output wires Γ, it computes $C_j = c_\alpha + c_\beta + 2c_\Gamma - 2\mathsf{Com}(1; 0)$ and proves using Π_{NIZK} that C_j is a commitment to 0 or 1. The GOS protocol showed that if the order of the message domain of Com is at least 4 then C_j will always be a commitment to 0 or 1. The verifier verifies the proofs and checks that the commitment corresponding to the output wire is $\mathsf{Com}(1; 1)$.

Adaptive Soundness and Proof of Knowledge. The distribution of the crs is identical in the real and ideal world. A corrupt prover P^* can adaptively chose the statement based on the crs distribution. If P^* constructs a proof for a statement $x \notin \mathcal{L}$, then he must have broken the binding property of the commitment scheme or the soundness of Π_{NIZK}. Else if P^* constructs a proof for a statement $x \in \mathcal{L}$ then the simulator can extract the witness bits y_i from the individual proofs by invoking the simulator of Π_{NIZK}. Note that we encrypt the randomness r_i for each commitment c_i for reduction in the security proof. If a corrupt prover computes two valid openings of c_i then those openings can be decrypted and used to break the binding property of Com.

Adaptive Zero Knowledge. Zero-knowledge is ensured since a ZK simulator can construct the c_i commitments in the equivocal mode, i.e. $c_i = \mathsf{Com}(0; r_i) = \mathsf{Com}(1; r_i')$, and set the encryptions as $(e_{i,0}, e_{i,1}) = (\mathsf{Enc}(\mathsf{pk}, r_i; s_i), \mathsf{Enc}(\mathsf{pk}, r_i'; s_i'))$. He sets the output wire commitment as $\mathsf{Com}(1; 1)$. He invokes the ZK simulator of Π_{NIZK} to produce the proofs for each wire and each NAND gate. In the real world one of the encryptions corresponding to a commitment is honestly generated while the other is obliviously sampled. In the ideal world both encryptions are honestly generated. The two cases are indistinguishable due to oblivious ciphertext sampling property of PKE. Thus, ZK follows from the ZK of Π_{NIZK}, hiding of Com and oblivious ciphertext sampling property of PKE. A statically corrupt verifier can also choose the statement adaptively based on the crs distribution. Adaptive ZK of this protocol is ensured since Π_{NIZK} supports adaptive ZK, the Com is hiding and PKE provides oblivious ciphertext sampling property.

Security against Adaptive Corruptions. When the prover gets corrupted and it obtains the witness w it can compute y. Suppose $y_i = 0$, then he opens c_i and $e_{i,0}$ as $(0, r_i, s_i)$ and claims that $e_{i,1}$ was obliviously sampled. It invokes the Π_{NIZK} simulator of wire i with input witness (y_i, r_i) to obtain randomness for the NIZK proof corresponding to wire i. Similar steps are repeated to obtain randomness for each NAND gate j and the corresponding NIZK proof. The encryption of r_i' (in ideal world) is indistinguishable from an obliviously sampled ciphertext (in real world) due to the oblivious sampling property. Thus, security against adaptive corruption is ensured due to adaptive security of Π_{NIZK}, equivocal property of Com and oblivious sampleability of PKE.

Multi-proof. The protocol Π_{sNIZK} also allows the prover to prove multiple statements using the same $\mathsf{crs}_{\mathsf{sNIZK}}$. If a corrupt party breaks the security of the protocol in one of the proof then that party can be used to either break the multi-proof security of Π_{NIZK} or the security of Com. We discuss triple adaptive security proof of Π_{sNIZK} by proving Theorem 8.

Theorem 8. *Assuming* PKE *is a public key encryption scheme with oblivious ciphertext sampling,* **Com** *is an equivocal additively homomorphic commitment scheme in the reusable* **crs**$_{Com}$ *model and* Π_{NIZK} *implements* $\mathcal{F}_{NIZK}^m$ *against adaptive corruption of parties, then* Π_{sNIZK} *UC-securely implements* $\mathcal{F}_{NIZK}$ *functionality for NP languages against adaptive adversaries in the* **crs** *model where* $|crs| = poly(\kappa)$. Π_{sNIZK} *is also adaptively sound and adaptively zero knowledge.*

Proof. The work of [GOS12] showed that if the order of the message domain of Com is atleast 4 and α, β are the input values of a NAND gate and Γ is the output value then,

$$\Gamma = \alpha \odot \beta \text{ if and only if } \alpha + \beta + 2\Gamma - 2 \in \{0, 1\}$$

And if the order is 3 then

$$\Gamma = \alpha \odot \beta \text{ if and only if } \alpha + \beta + 2\Gamma - 2 \in \{0, 1\} \text{ and } \alpha + \beta + \Gamma - 1 \in \{0, 1\}.$$

We assume the order is 4 (for the sake of simplicity) and correctness for the NAND gates follows from this observation. It can be modified when the order is 3. Next, we present two possible corruption cases.

P is statically corrupt and V is honest. In this case, the adversary can break the soundness of the protocol if he can break the binding of the commitment or the soundness of Π_{NIZK}. The simulation algorithm is presented in Fig. 15. We present the formal hybrids and prove indistinguishability as follows:

- **Hyb$_0$:** Real world.
- **Hyb$_1$:** Same as Hyb_0, except the simulator decrypts $(r_{i,0}, r_{i,1})$ from $(E_{i,0}, E_{i,1})$ for all $i \in [n]$. He aborts if $\exists i \in [n]$ (resp. $\exists j \in [m]$) s.t c_i (resp C_j) opens to both 0 and 1 using randomness $r_{i,0}$ (resp. $R_{j,0}$) and $r_{i,1}$ (resp. $R_{j,1}$).
  The adversary can distinguish between the hybrids if he computes valid decommitments for both 0 and 1. In such a case, the simulator can extract the randomness for both decommitments and he can be used to break the binding of the commitment scheme.
- **Hyb$_2$:** Same as Hyb_1, except the simulator aborts if the simulator of Π_{NIZK} fails to extract a correct witness from a single proof.
  Indistinguishability follows from the multi-proof security of Π_{NIZK}.

Fig. 14. Triply Adaptively Secure NIZK Protocol in the short crs model

---

Π_{sNIZK}

- **Primitives:** Non-interactive equivocal additively homomorphic commitment scheme $\text{Com} = (\text{Gen}, \text{Com}, \text{Ver}, \text{Equiv})$. Public key encryption scheme $\text{PKE} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ with oblivious ciphertext sampling algorithm oEnc. Adaptively secure NIZK protocol $\Pi_{\text{NIZK}} = (\text{Gen}, \text{P}, \text{V})$.
- **Public Inputs:** Common reference string $\text{crs}_{\text{sNIZK}} = (\text{crs}_{\text{Com}}, \text{pk}, \text{crs}_{\text{NIZK}})$ where $(\text{crs}_{\text{Com}}, \text{td}) \leftarrow \text{Com.Gen}(1^\kappa)$, $(\text{crs}_{\text{NIZK}}, \text{td}_{\text{NIZK}}) \leftarrow \Pi_{\text{NIZK}}.\text{Gen}(1^\kappa)$ and $(\text{pk}, \text{sk}) \leftarrow \text{PKE.KeyGen}(1^\kappa)$.
- **Notations:** If $x \in \mathcal{L}$ and w is a valid witness then $\mathcal{R}(x, w) = 1$. Circuit $\mathcal{C}$ computes $\mathcal{R}(x)$ s.t. $\mathcal{C}(y_1 y_2 \ldots y_n) = 1$ iff $\mathcal{R}(x, w) = 1$ where $y_1 y_2 y_{|w|} = w$ and $\mathcal{C}$ has m NAND gates and n wires, where the nth wire is the output wire of $\mathcal{C}$. $\text{Com.Com}(\alpha; r_\alpha) + \text{Com.Com}(\beta; r_\beta) = \text{Com.Com}(\alpha + \beta; r_\alpha + r_\beta)$.

Prove : $\text{P}(x, w)$

- Assign the wire values $y_1 y_2 \ldots y_n$ based on w.
- Commit to each bit y_i as $c_i \leftarrow \text{Com.Com}(y_i; r_i)$ and encrypt the randomness as $E_{i, y_i} = \text{PKE.Enc}(\text{pk}, r_i; s_i)$ and $E_{i, \overline{y_i}} \leftarrow \text{PKE.oEnc}(1^\kappa)$ for $i \in [n-1]$.
- For the output wire let $r_n = 0$ and $c_n = \text{Com.Com}(1; 1)$.
- For all $i \in [n-1]$, prove c_i is a commitment to 0 or 1 by computing proof $\pi_i \leftarrow \Pi_{\text{NIZK}}.\text{P}((c_i, E_{i,0}, E_{i,1}), (w_i, r_i, s_i))$ for the following relation,

$$\mathcal{R}_1((c_i, E_{i,0}, E_{i,1}), (w_i, r_i, s_i)) = (\exists r_i, s_i : (c_i = \text{Com.Com}(0; r_i) \wedge$$

$$E_{i,0} = \text{PKE.Enc}(\text{pk}, r_i; s_i)) \vee (c_i = \text{Com.Com}(1; r_i) \wedge E_{i,1} = \text{PKE.Enc}(\text{pk}, r_i; s_i))).$$

- For the jth ($j \in [m]$) NAND gate with input wires α and β and output wire Γ perform the following:
  - Compute $C_j = c_\alpha + c_\beta + 2c_\Gamma - 2$.
  - Compute the randomness for C_j as $R_j = r_\alpha + r_\beta + 2r_\Gamma - 2$.
  - Prove C_j is a commitment to 0 or 1 by computing proof $\pi'_j \leftarrow \Pi_{\text{NIZK}}.\text{P}(C_j, R_j)$ for the following relation,
$$\mathcal{R}_2(C_j, R_j) = (\exists R_j : C_j = \text{Com.Com}(0; R_j) \vee (C_j = \text{Com.Com}(1; R_j))).$$
- P sends proof $\gamma = \{\{c_i, E_{i,0}, E_{i,1}, \pi_i\}_{i \in [n-1]}, c_n, \{\pi'_j\}_{j \in [m]}\}$ to V.

Verify : $\text{V}(x, \gamma)$

- Parse $\gamma = \{\{c_i, E_{i,0}, E_{i,1}, \pi_i\}_{i \in [n-1]}, c_n, \{\pi'_j\}_{j \in [m]}\}$.
- Verifies $c_n = \text{Com.Com}(1; 0)$.
- Verifies π_i for $i \in [n-1]$ using $\Pi_{\text{NIZK}}.\text{V}((c_i, E_{i,0}, E_{i,1}), \pi_i)$ algorithm.
- For the jth ($j \in [m]$) NAND gate with input wires α and β and output wire Γ compute $C_j = c_\alpha + c_\beta + 2c_\Gamma - 2$ and verify π'_j by running $\Pi_{\text{NIZK}}.\text{V}(C_j, R_j)$.
- If verification succeeds then output ACCEPT else output REJECT.

P is honest and V is statically corrupt. Next, we demonstrate zero knowledge property of Π_{sNIZK} by relying on the ZK simulator of Π_{NIZK} and the equivocal property of Com. The simulation algorithm is presented in Fig. 16. We present the formal hybrids and prove indistinguishability as follows:

- **Hyb$_0$:** Real world.
- **Hyb$_1$:** Same as Hyb$_0$, except the simulator constructs π_i and π'_j using the ZK simulator of Π_{NIZK}.

  Indistinguishability follows due to the multi-proof security of Π_{NIZK}.
- **Hyb$_2$:** Same as Hyb$_1$, except the simulator uses the equivocating trapdoor td (of Com) computes all the commitments c_i as commitment to 0 using randomness r_i and commitment to 1 using randomness r'_i. He encrypts r_i and r'_i to form $E_{i,0}$ and $E_{i,1}$ respectively.

  Indistinguishability follows from the equivocal property of Com. The oblivious ciphertext sampling property of PKE also ensures that for every $i \in [n]$ the following two events are indistinguishable - one of $E_{i,0}$ and $E_{i,1}$ is obliviously sampled (real world view), both $E_{i,0}$ and $E_{i,1}$ are valid encryptions (ideal world view).

Fig. 15. Simulation against a statically corrupt P* by $\mathcal{S}$

- **Public Inputs:** Common reference string $crs_{sNIZK} = (crs_{Com}, pk, crs_{NIZK})$.
- **Simulator Inputs:** Trapdoor of $crs_{sNIZK} = (td, sk, td_{NIZK})$ where sk is the secret key for pk, td is the equivocating trapdoor of Com and td_{NIZK} is the trapdoor of crs_{NIZK}.

Prove :
P* sends proof $\gamma = \{\{c_i, E_{i,0}, E_{i,1}, \pi_i\}_{i \in [n-1]}, c_n, \{\pi'_j\}_{j \in [m]}\}$ to V.
Verify : $\mathcal{S}(x, \gamma)$
- $\mathcal{S}$ runs the honest verifier algorithm to compute C_j for $j \in [m]$ and verify $c_n = \mathsf{Com}(1; 1)$.
- For $i \in [n]$, $\mathcal{S}$ aborts if $\mathsf{PKE.Dec}(sk, E_{i,0})$ and $\mathsf{PKE.Dec}(sk, E_{i,1})$ are valid decommitment randomness for c_i to 0 and 1 respectively.
- $\mathcal{S}$ invokes the simulator of Π_{NIZK} with td_{NIZK} to verify $\{\pi_i\}_{i \in [n-1]}, \{\pi'_j\}_{j \in [m]}$ and extract $y_1 \ldots y_{|w|}$.
- $\mathcal{S}$ invokes $\mathcal{F}_{NIZK}$ with $w = y_1 \ldots y_{|w|}$ to complete the simulation.

Fig. 16. Simulation against a statically corrupt V* by $\mathcal{S}$

- **Public Inputs:** Common reference string $crs_{sNIZK} = (crs_{Com}, pk, crs_{NIZK})$.
- **Simulator Inputs:** Trapdoor of $crs_{sNIZK} = (td, sk, td_{NIZK})$ where sk is the secret key for pk, td is the equivocating trapdoor of Com and td_{NIZK} is the trapdoor of crs_{NIZK}.

Prove : $\mathcal{S}(x)$
- For $i \in [n-1]$, compute $c_i = \mathsf{Com.Com}(0; r_i) = \mathsf{Com}(1; r'_i)$. Compute $E_{i,0} = \mathsf{PKE.Enc}(pk, r_i; s_i)$ and $E_{i,1} = \mathsf{PKE.Enc}(pk, r'_i; s'_i)$. Compute $c_n = \mathsf{Com.Com}(1; 1)$.
- For $j \in [m], i \in [n-1]$, invoke Π_{NIZK} simulator with td_{NIZK} to obtain simulated proofs for π_i and π'_j.
- Send proof $\gamma = \{\{c_i, E_{i,0}, E_{i,1}, \pi_i\}_{i \in [n-1]}, c_n, \{\pi'_j\}_{j \in [m]}\}$ to V.
Verify :
Runs its own adversarial algorithm.

P gets corrupted post-execution. We discuss simulating the view of an honest prover when it gets adaptively corrupted post-execution. In such a case the adaptive simulator obtains the correct $y_1 \ldots y_n$ and he equivocates the c_i s.t. they open to y_i. It claims that $E_{i, \overline{y_i}}$ was randomly sampled owing to the oblivious ciphertext sampling algorithm. π_i and π'_j are simulated by invoking the adaptive simulator for Π_{NIZK} using the respective witnesses. Indistinguishability follows due to the equivocal property of Com which holds in presence of adaptive corruptions, oblivious ciphertext sampling property of PKE and adaptive security for multi-proofs of Π_{NIZK}.

Proof of Adaptive Soundness. In our compiler, the setup string is $crs_{sNIZK} = (crs_{NIZK}, pk, crs_{Com})$ where crs_{NIZK} is the setup string of an adaptively sound NIZK protocol Π_{NIZK}. If an adversary $\mathcal{A}$ breaks adaptive soundness of Π_{sNIZK} then we can use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{COM}$, who breaks the binding property of Com, or an adversary $\mathcal{A}_{NIZK}$, who breaks the adaptive soundness property of Π_{NIZK}. We construct $\mathcal{A}_{COM}$ and $\mathcal{A}_{NIZK}$ as follows:

- *Constructing $\mathcal{A}_{COM}$:* $\mathcal{A}_{COM}$ obtains crs_{Com} for the commitment and he samples a PKE key pair (pk, sk) and crs_{NIZK} to set $crs_{sNIZK} = (crs_{NIZK}, pk, crs_{Com})$. $\mathcal{A}_{COM}$ invokes $\mathcal{A}$ to obtain a statement and a proof. $\mathcal{A}$ checks if $\exists i \in [n]$, s.t. commitment c_i opens both to 0 and 1 by decrypting r_i and r'_i from $E_{i,0}$ and $E_{i,1}$ given the secret key sk. If there is one such c_i then $\mathcal{A}_{COM}$ outputs $(0, r_i)$ and $(1, r'_i)$ to the commitment challenger and breaks the binding property of Com. Else, $\mathcal{A}$ has broken the adaptive soundness of Π_{NIZK}.
- *Constructing $\mathcal{A}_{NIZK}$:* $\mathcal{A}_{COM}$ obtains crs_{NIZK} for Π_{NIZK}. He samples a PKE key pair (pk, sk) and crs_{Com} to set $crs_{sNIZK} = (crs_{NIZK}, pk, crs_{Com})$. $\mathcal{A}_{COM}$ invokes $\mathcal{A}$ to obtain a statement and a proof. $\mathcal{A}$ checks if $\exists i \in [n]$, s.t. commitment c_i opens both to 0 and 1 by decrypting r_i and r'_i from $E_{i,0}$ and $E_{i,1}$ given the secret key sk. If there is no such c_i then $\mathcal{A}$ has broken adaptive

soundness in atleast one of the $\{\pi_i\}_{i\in[n-1]}$ or $\{\pi_j'\}_{j\in[m]}$. $\mathcal{A}_{\mathsf{NIZK}}$ samples one of those proofs randomly and returns it to the challenger of adaptive soundness game of Π_{NIZK}. If $\mathcal{A}$ wins with probability p and $\mathcal{A}_{\mathsf{COM}}$ wins with probability p_c then $\mathcal{A}_{\mathsf{NIZK}}$ wins with probability $\frac{p-p_c}{m+n-1}$.

Proof of Adaptive ZK. Our ZK simulator in Fig. 16 suffices for adaptive zero knowledge as the statement x is adaptively chosen by the adversary after obtaining $\mathsf{crs}_{\mathsf{sNIZK}}$. □

The homomorphic commitment scheme can be instantiated from DDH (Pedersen commitment or [CSW20]) or LWE [GVW15] asusmptions. The PKE can be instantiated from DDH assumption (Elgamal encryption) or LWE [GSW13] assumptions. This yields our compiler from DDH or LWE assumption.

7 Triply Adaptive, multi-proof UC-NIZK Argument

We add non-malleability to our Π_{sNIZK} protocol to attain UC-security for multiple statements in different subsessions. This is performed in the same way as GOS using tag based simulation-sound trapdoor commitment $\mathsf{Com}_{\mathsf{SST}}$ and strong one-time signature SIG. The prover generates a pair of signature keys $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SIG}.\mathsf{KeyGen}$. It commits to the witness bits w using $\mathsf{Com}_{\mathsf{SST}}$ with the tag being $(\mathsf{vk}, \mathsf{sid}, \mathsf{ssid}, x)$ (where sid is the session ID of the multi-instance NIZK functionality and ssid is the sub-session ID for the particular proof) and encrypts the randomness for the commitments. It proves using Π_{sNIZK} that $\mathcal{R}(x, w) = 1$ and the witness bits are correctly committed to compute a proof π. It signs the proof π using sk and sends the proof π and the signature as the final proof. The signature enables that an adversary cannot forge a signature on a different proof with the same vk. Whereas, $\mathsf{Com}_{\mathsf{SST}}$ and Π_{sNIZK} ensures that an adversary cannot reuse the same proof π in a different session ssid since it is bound to the vk and ssid.

Security against Statically Corrupt Prover. Soundness follows from the binding property of $\mathsf{Com}_{\mathsf{SST}}$, unforgeability of SIG and adaptive soundness of Π_{sNIZK}. The witness can be extracted from the commitments by decrypting the randomness of the commitments from the encryptions using sk. Next, we briefly discuss the different cases for triple adaptive security.

Security against Adaptive Corruption of Prover. The ZK simulator commits to all 0s as witness and invokes the ZK simulator of Π_{sNIZK} to construct the simulated proof. Upon obtaining the witness it can equivocate the commitments using the trapdoor and equivocate the proof by invoking the adaptive simulator of Π_{sNIZK}.

Adaptive Soundness and Adaptive Zero Knowledge. The crs distribution is identical in the real and ideal world for our multi-session UC protocol. Adaptive soundness follows from the unforgeability of signature, binding property of the tag-based commitment scheme and adaptive soundness of underlying single instance NIZK protocol. For adaptive ZK our ZK simulator (mentioned in previous paragraph) suffices.

Theorem 9. *If Π_{sNIZK} UC-realizes $\mathcal{F}_{\mathsf{NIZK}}$ for a single proof, SIG is a strong one-time secure signature scheme, $\mathsf{Com}_{\mathsf{SST}}$ is a tag-based simulation-sound trapdoor commitment and PKE is a public key encryption scheme with oblivious ciphertext sampling property then $\Pi_{\mathsf{UC\text{-}NIZK}}$ UC-securely implements $\mathcal{F}_{\mathsf{NIZK}}$ for multiple instances against adaptive adversaries. In addition, $\Pi_{\mathsf{UC\text{-}NIZK}}$ is adaptively sound and adaptively zero knowledge.*

$\Pi_{\text{UC-NIZK}}$

- **Primitives:** Strong one-time signature scheme $\text{SIG} = (\text{KeyGen}, \text{Sign}, \text{Ver})$. Public key encryption scheme $\text{PKE} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ with oblivious ciphertext sampling algorithm oEnc. Simulation sound tag based commitment $\text{Com}_{\text{SST}} = (\text{KeyGen}, \text{Com}, \text{Dec}, \text{Ver}, \text{TCom}, \text{TOpen})$.
- **Public Inputs:** Common reference string $\text{crs}_{\text{UC-NIZK}} = (\text{crs}_{\text{Com}}, \text{pk}, \text{crs}_{\text{sNIZK}})$ where $(\text{crs}_{\text{Com}}, \text{td}) \leftarrow \text{Com}_{\text{SST}}.\text{KeyGen}(1^\kappa)$, $(\text{crs}_{\text{sNIZK}}, \text{td}_{\text{sNIZK}}) \leftarrow \Pi_{\text{sNIZK}}.\text{Gen}(1^\kappa)$ and $(\text{pk}, \text{sk}) \leftarrow \text{PKE}.\text{KeyGen}(1^\kappa)$. Session id sid and subsession id ssid.

Prove : $\text{P}(x, w, \text{sid}, \text{ssid})$
- Generates signature key pair $(\text{sk}, \text{vk}) \leftarrow \text{SIG}.\text{KeyGen}(1^\kappa)$.
- Commits to the bits of witness w_i using the tag $(\text{vk}, \text{sid}, \text{ssid}, x)$ as $c_i = \text{Com}_{\text{SST}}.\text{Com}(\text{crs}_{\text{Com}}, w_i, (\text{vk}, \text{sid}, \text{ssid}, x); r_i)$ using randomness r_i. Encrypt the randomness as $E_{i,r_i} = \text{PKE}.\text{Enc}(\text{pk}, r_i; s_i)$ and $E_{i, \overline{y_i}} \leftarrow \text{PKE}.\text{oEnc}(1^\kappa)$ for $i \in [n-1]$.
- Computes proof $\pi \leftarrow \Pi_{\text{sNIZK}}.\text{P}((x, \text{vk}, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]}), (w, \text{sk}, \{r_i, s_i\}_{i \in [|w|]}))$ for the following relation:
$$\mathcal{R}'(x', w') = ((x, \text{vk}, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]}), (w, \text{sk}, \{r_i, s_i\}_{i \in [|w|]}) : \mathcal{C}(x, w) = 1 \wedge$$
$$(\forall i \in [|w|], c_i = \text{Com}_{\text{SST}}.\text{Com}(\text{crs}_{\text{Com}}, w_i, (\text{vk}, \text{sid}, \text{ssid}, x); r_i) \wedge$$
$$(E_{i,0} = \text{PKE}.\text{Enc}(\text{pk}, r_i; s_i) \vee E_{i,1} = \text{PKE}.\text{Enc}(\text{pk}, r_i; s_i))).$$
- Signs the proof $s \leftarrow \text{SIG}.\text{Sign}(\text{sk}, \pi, \text{sid}, \text{ssid}, x)$. Sends the proof $\gamma = (\pi, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]}, s, \text{vk})$.

Verify : $\text{V}(x, \gamma, \text{sid}, \text{ssid})$
- The verifier outputs ACCEPT if $\Pi_{\text{sNIZK}}.\text{V}((x, \text{vk}, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]}, \pi) = 1$ and $\text{SIG}.\text{Ver}(\text{vk}, (\pi, x, \text{sid}, \text{ssid}), s) = 1$. Else, output REJECT.

Proof. We consider that there are s subsessions and out of that the prover is statically corrupt in s' of those subsessions and he gets adaptively corrupted in $s - s'$ subsessions. In s' of those subsessions the adversary can break the security of the protocol without breaking the security of the single-session NIZK protocol if he can forge a signature or if he can break the binding property of the simulation sound tag-based commitment scheme Com_{SST}. This is captured by a sequence of hybrids for each such subsession. For the rest $s - s'$ subsessions, the environment $\mathcal{Z}$ can distinguish a real and ideal world view either by distinguishing the views of the single-session NIZK protocol or by breaking the equivocal property of Com_{SST}(or oblivious ciphertext sampling property of PKE). In Fig. 18 we capture our ideal world simulation against adaptive corruption of parties for multi-subsessions. We consider $5s + 5$ hybrids where in hybrid $5j$ (for $j \in [s]$) the first $j - 1$ subsessions are simulated and the rest are real executions. The jth subsession is modified in hybrids Hyb_{5j+1}- Hyb_{5j+3} if the prover is statically corrupted. Else, if the prover is honest then the simulator simulates on behalf of the prover and that is captured in hybrids Hyb_{5j+4} and Hyb_{5j+5}. In Hyb_{5j+5} the first j subsessions are simulated and the rest $s - j$ subsessions consist of real executions.

- **Hyb_0:** Real world execution of all the subsessions.
  $\vdots$
- **Hyb_{5j}:** The simulator $\mathcal{S}$ simulates the first $j - 1$ subsessions by invoking the simulator $\Pi_{\text{sNIZK}}.\mathcal{S}$ and the rest $s - j + 1$ subsessions are simulated by running the real protocol with the honest prover's input witness.
- **Hyb_{5j+1}:** Same as Hyb_{5j}, except if the prover is statically corrupt in subsession j and the reduction aborts if the corrupt prover has successfully generated a signature s using vk in the proof for subsession j, where vk was used to generate a proof γ' by an honest prover in another subsession.
  The adversary $\mathcal{A}$ can distinguish between the two hybrids if he forges a signature s on $m = (\pi, x, \text{sid}, \text{ssid})$ using vk where vk has been used to generate a proof in a different NIZK

44

subsession. One can build an adversary for the strong one-time signature scheme $\mathcal{A}_s$ using $\mathcal{A}$ where $\mathcal{A}_s$ outputs (m, s) as the forgery.

- **Hyb$_{5j+2}$:** Same as Hyb_{5j+1}, except if the prover is statically corrupt in subsession j and the reduction aborts if c_i (in the maliciously constructed proof for subsession j) opens to both 0 and 1 using randomness $r_{i,0}$ and $r_{i,1}$ from $E_{i,0}$ and $E_{i,1}$ respectively.

  The adversary can distinguish between the hybrids if he computes valid decommitments for both 0 and 1. In such a case, the simulator can extract the randomness for both decommitments as the adversary can be used to break the simulation soundness property of Com_{SST} for the challenge tag $(\text{vk}, \text{sid}, \text{ssid}, x)$.

- **Hyb$_{5j+3}$:** Same as Hyb_{5j+2}, except if the prover is statically corrupt in subsession j then the simulator constructs the ideal world view by invoking the single session simulator of Π_{sNIZK}.

  Indistinguishability between Hyb_{5j+2} and Hyb_{5j+3} follows from the security of Π_{sNIZK}. This completes simulating the jth subsession for static corruption of prover.

- **Hyb$_{5j+4}$:** Same as Hyb_{5j+3}, except if the prover is honest in the beginning of susbsession j then the simulator simulates an honest prover by computing the commitments using TCom and encrypts the corresponding randomness values for 0 and 1. The proof is generated honestly using the knowledge of the witness and adaptive corruption of prover is simulated by opening the honest prover randomness.

  Indistinguishability follows from the security of Com_{SST} and oblivious ciphertext sampling property of PKE.

- **Hyb$_{5j+5}$:** Same as Hyb_{5j+4}, except if the prover is honest in the beginning of susbsession j then the simulator invokes $\Pi_{\text{sNIZK}}.\mathcal{S}(x')$ to simulate the proof. If the prover gets corrupted post-execution with witness w then invoke the adaptive simulator of Π_{sNIZK} with w to obtain randomness consistent with the proof.

  Indistinguishability between Hyb_{5j+4} and Hyb_{5j+5} follows from the adaptive security of Π_{sNIZK}. This completes simulating the jth subsession for adaptive corruption of prover.

  $\vdots$

- **Hyb$_{5s+5}$:** This is the ideal world execution of the protocol where all the subsessions are simulated. The ideal world adversary view consists of the combined ideal world views of all the subsessions and this is forwarded to the environment $\mathcal{Z}$.

Proof of Adaptive Soundness. If adversary $\mathcal{A}$ breaks adaptive soundness of $\Pi_{\text{UC-NIZK}}$ for a subsession j then one can either build an adversary $\mathcal{A}_\text{S}$, forging signature, or an adversary $\mathcal{A}_{\text{COM}}$, breaking soundness of Com_{SST}, or an adversary $\mathcal{A}_{\text{sNIZK}}$ breaking adaptive soundness of Π_{sNIZK} as follows:

- *Constructing $\mathcal{A}_\text{S}$:* $\mathcal{A}_\text{S}$ initiates a multi-session UC protocol with $\mathcal{A}$. $\mathcal{A}_\text{S}$ samples (pk, sk), crs_{Com} and $\text{crs}_{\text{sNIZK}}$ s.t. he knows the trapdoors of $\text{crs}_{\text{UC-NIZK}} = (\text{crs}_{\text{Com}}, \text{pk}, \text{crs}_{\text{sNIZK}})$. $\mathcal{A}$ obtains $\text{crs}_{\text{UC-NIZK}}$ as the setup string. $\mathcal{A}_\text{S}$ has oracle access to obtain signatures using challenge verification key vk. He incorporates the vk as part of the tag in a subsession $j' \neq j$, where the prover is honest. $\mathcal{A}_\text{S}$ computes the proof for subsession j' using trapdoors of crs_{Com} and $\text{crs}_{\text{sNIZK}}$. $\mathcal{A}_\text{S}$ signs the proof using oracle access to the signing algorithm. $\mathcal{A}_\text{S}$ sends this proof to $\mathcal{A}$ as the proof for subsession j'. $\mathcal{A}_\text{S}$ simulates other subsessions where the prover is honest, by invoking the ZK simulator using the trapdoors of $\Pi_{\text{UC-NIZK}}$. $\mathcal{A}$ returns a statement and a proof for subsession j where he breaks adaptive soundness. If $\mathcal{A}$ has used the same vk as part of the proof in subsession j then $\mathcal{A}_\text{S}$ forwards this proof as a forgery using vk to the signing algorithm. The proofs in subsession j and subsession j'

Fig. 18. Simulation against adaptive corruption of parties in $\Pi_{\text{UC-NIZK}}$

- **Public Inputs:** Common reference string $\text{crs}_{\text{UC-NIZK}} = (\text{crs}_{\text{Com}}, \text{pk}, \text{crs}_{\text{sNIZK}})$.
- **Simulator Inputs:** Trapdoor of $\text{crs}_{\text{UC-NIZK}} = (\text{td}, \text{sk}, \text{td}_{\text{sNIZK}})$ where td is the trapdoor for Com_{SST}, sk is the secret key of pk and td_{sNIZK} is trapdoor of $\text{crs}_{\text{sNIZK}}$.

The simulator $\mathcal{S}$ simulates multiple NIZK subsessions concurrently using the trapdoor td of crs. $\mathcal{S}$ maintains a list $\mathcal{L}$ of subsession ids, corresponding statements x'' and proofs with entries of the form - $(\text{ssid}'', x'', \gamma'')$. Upon obtaining a request for ideal world adversary view in subsession ssid from the environment $\mathcal{Z}$ with statement x, the simulator $\mathcal{S}$ returns γ to $\mathcal{Z}$ if $(\text{ssid}, x, \gamma) \in \mathcal{L}$. If $(\text{ssid}, x'', \_) \in \mathcal{L}$ and $x \neq x''$: $\mathcal{S}$ returns $\perp$ to $\mathcal{Z}$. Else, $\mathcal{S}$ generates the ideal world view as described below for ssid based on the corruption of prover.

Case 1: If the prover is statically corrupt in subsession ssid :
Prove :
$\mathcal{S}$ invokes the dummy adversary for subsession ssid with statement x to obtain $\gamma = (\pi, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]}, s, \text{vk})$.
Verify : $\mathcal{S}(x, \gamma, \text{sid}, \text{ssid})$

- Abort if vk was the verification key of an honestly generated proof in a different subsession $\text{ssid}' \neq \text{ssid}$ and $\text{SIG.Ver}(\text{vk}, (\pi, x, \text{sid}, \text{ssid}), s) = 1$.
- For $i \in [|w|]$, $\mathcal{S}$ aborts if $\text{PKE.Dec}(\text{sk}, E_{i,0})$ and $\text{PKE.Dec}(\text{sk}, E_{i,1})$ are valid decommitment randomness for c_i to 0 and 1 respectively.
- Extracts witness w by invoking the simulator algorithm for Π_{sNIZK} with trapdoor td_{sNIZK} of $\text{crs}_{\text{sNIZK}}$ for statement $x' = (x, \text{vk}, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]})$, i.e. $w = \Pi_{\text{sNIZK}}.\mathcal{S}(x', \pi)$.
- If $w = \perp$ then output REJECT. Else, invoke $\mathcal{F}_{\text{NIZK}}$ with w and output ACCEPT.

Case 2: If the prover is honest at the beginning of subsession ssid :
Prove : $\mathcal{S}(x, \text{sid}, \text{ssid})$

- Generates signature key pair (sk, vk).
- For $i \in [|w|]$, $\mathcal{S}$ computes $(c_i, r_i') \leftarrow \text{Com}_{\text{SST}}.\text{TCom}(\text{crs}_{\text{Com}}, \text{td}, (\text{vk}, x, \text{sid}, \text{ssid}))$.
- Compute randomness $r_{i,0} \leftarrow \text{Com}_{\text{SST}}.\text{TOpen}(\text{crs}_{\text{Com}}, c, 0, r_i')$ and $r_{i,1} \leftarrow \text{Com}_{\text{SST}}.\text{TOpen}(\text{crs}_{\text{Com}}, c, 1, r_i')$. Encrypt $E_{i,0} = \text{PKE.Enc}(\text{pk}, r_{i,0})$ and $E_{i,1} = \text{PKE.Enc}(\text{pk}, r_{i,1})$.
- $\mathcal{S}$ invokes $\pi \leftarrow \Pi_{\text{sNIZK}}.\mathcal{S}(x')$ with trapdoors td_{sNIZK} for statement $x' = (x, \text{vk}, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]})$ to obtain π.
- Signs the proof $s \leftarrow \text{SIG.Sign}(\text{sk}, \pi, \text{sid}, \text{ssid}, x)$.
- $\mathcal{S}$ sends the proof $\gamma = (\pi, \{c_i, E_{i,0}, E_{i,1}\}_{i \in [|w|]}, s, \text{vk})$ to verifier.

Verify :
Performs its own algorithm.
Post-Execution Corruption of Prover :
$\mathcal{S}$ computes the correct decommitment randomness r_{i,w_i} and its encryption randomness s_i. It invokes the adaptive simulator $\Pi_{\text{sNIZK}}.\mathcal{S}$ on $(w, \text{sk}, r_{i,w_i}, s_i)$ to obtain randomness for π. It opens c_i and E_{i,w_i} using randomness r_{i,w_i} and s_i respectively and claims that $E_{i,\overline{w_i}}$ is randomly sampled.

are ensured to be different because the subsession ids are different and hence the tags are different for each subsession. Thus, the proof for subsession j with same vk will act as a forgery.

- *Constructing $\mathcal{A}_{COM}$:* $\mathcal{A}_{COM}$ initiates a multi-session UC protocol with $\mathcal{A}$. $\mathcal{A}_{COM}$ receives crs_{Com} from the challenger of the binding game. $\mathcal{A}_{COM}$ samples (pk, sk) and crs_{sNIZK} and sets $crs_{UC\text{-}NIZK} = (crs_{Com}, pk, crs_{sNIZK})$. $\mathcal{A}$ obtains $crs_{UC\text{-}NIZK}$ as the setup string. If $\mathcal{A}$ breaks adaptive soundness of subsession j then $\mathcal{A}_{COM}$ checks if $\exists i \in [[w]]$, s.t. commitment c_i opens both to 0 and 1 by decrypting r_i and r_i' from $E_{i,0}$ and $E_{i,1}$ given the secret key sk. He returns r_i and r_i' to the challenger of Com to break the binding property.

- *Constructing $\mathcal{A}_{sNIZK}$:* $\mathcal{A}_{sNIZK}$ obtains crs_{sNIZK} from the challenger of adaptive soundness for Π_{sNIZK}. $\mathcal{A}_{sNIZK}$ initiates a multi-session UC protocol with $\mathcal{A}$. $\mathcal{A}_{sNIZK}$ samples (pk, sk) and crs_{Com} and sets $crs_{UC\text{-}NIZK} = (crs_{Com}, pk, crs_{sNIZK})$. $\mathcal{A}$ obtains $crs_{UC\text{-}NIZK}$ as the setup string. $\mathcal{A}_{sNIZK}$ simulates subsessions where the prover is honest by invoking the ZK simulator using the trapdoors of crs_{Com}. If $\mathcal{A}$ returns a statement and a proof for subsession j then $\mathcal{A}_{sNIZK}$ checks that $\nexists i \in [[w]]$, s.t. commitment c_i opens both to 0 and 1 by decrypting r_i and r_i' from $E_{i,0}$ and $E_{i,1}$ given the secret key sk. Also, $\mathcal{A}_{sNIZK}$ verifies that the vk contained in the tag for session j is not used in any previous subsessions where the prover was honest. These two checks rule out the possibility of $\mathcal{A}$ breaking the binding property of Com and forging a signature. After these checks, it is ensured that $\mathcal{A}$ succeeds in breaking adaptive soundness in subsession j by breaking adaptive soundness of Π_{sNIZK}. $\mathcal{A}_{sNIZK}$ forwards the statement and the proof (without the signature) forwarded by $\mathcal{A}$ in subsession j.

Proof of Adaptive Zero Knowledge. Our simulator for an honest prover provides adaptive zero knowledge even when a statically corrupt verifier adaptively chooses the statement to be proven based on the $crs_{UC\text{-}NIZK}$ distribution. □

References

AF07. Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 118–136. Springer, Heidelberg, February 2007.

AMPS21. Navid Alamati, Hart Montgomery, Sikhar Patranabis, and Pratik Sarkar. Two-round adaptively secure MPC from isogenies, lpn, or CDH. In *Advances in Cryptology - ASIACRYPT 2021*, volume 13091 of *Lecture Notes in Computer Science*, pages 305–334. Springer, 2021.

BCG$^+$14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

BFM90. Manuel Blum, Paul Feldman, and Silvio Micali. Proving security against chosen cyphertext attacks. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 256–268. Springer, Heidelberg, August 1990.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

BKM06. Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 60–79. Springer, Heidelberg, March 2006.

BKM20. Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2020, Part III*, LNCS, pages 738–767. Springer, Heidelberg, August 2020.

Blu86. Manuel Blum. How to prove a theorem so no one else can claim it. 1986.

BMW03. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

BSMP91. Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

Can01.   Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CCH$^+$19.   Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

CD00.   Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 331–345. Springer, Heidelberg, December 2000.

CF01.   Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

CGH98.   Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

CGPS21.   Suvradip Chakraborty, Chaya Ganesh, Mahak Pancholi, and Pratik Sarkar. Reverse firewalls for adaptively secure MPC without setup. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 335–364. Springer, 2021.

CJJ21.   Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. LNCS, pages 394–423. Springer, Heidelberg, 2021.

CLOS02.   Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

CPS$^+$16.   Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 63–92. Springer, Heidelberg, May 2016.

CPV20.   Michele Ciampi, Roberto Parisella, and Daniele Venturi. On adaptive security of delayed-input sigma protocols and fiat-shamir nizks. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020,*, pages 670–690, 2020.

CS98.   Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, August 1998.

CsW19.   Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 30–60. Springer, Heidelberg, August 2019.

CSW20.   Ran Canetti, Pratik Sarkar, and Xiao Wang. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 277–308. Springer, 2020.

DDN91.   Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd ACM STOC*, pages 542–552. ACM Press, May 1991.

DGI$^+$19.   Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019.

DN00.   Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, Heidelberg, August 2000.

FLS99.   Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

FS87.   Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GGI$^+$15.   Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, October 2015.

GMW87.   Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

GOS06.   Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

GOS12.   Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.

GR13.   Oded Goldreich and Ron D. Rothblum. Enhancements of trapdoor permutations. *Journal of Cryptology*, 26(3):484–512, July 2013.

GSW13.   Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

GVW15.   Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.

HL18.   Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 850–858. IEEE Computer Society, 2018.

HLR21.   Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-shamir via list-recoverable codes (or: parallel repetition of gmw is not zero-knowledge). In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, 2021*, pages 750–760, 2021.

HV16.   Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On the power of secure two-party computation. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 397–429. Springer, Heidelberg, August 2016.

KKK21.   Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Composition with knowledge assumptions. LNCS, pages 364–393. Springer, Heidelberg, 2021.

KNYY19. Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Exploring constructions of compact NIZKs from various assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 639–669. Springer, Heidelberg, August 2019.

KNYY20. Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Compact NIZKs from standard assumptions on bilinear maps. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, LNCS, pages 379–409. Springer, Heidelberg, May 2020.

KZM+15. Ahmed E. Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T.-H. Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. How to use snarks in universally composable protocols. *IACR Cryptol. ePrint Arch.*, 2015:1093, 2015.

LP09.   Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

LZ09.   Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 183–201. Springer, Heidelberg, March 2009.

MP12.   Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

NY90.   Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

PS19.   Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019.

Sch90.   Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

SCO+01. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 566–598, 2001.

Yao86.   Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A   Equivalence of UC-Commitment Functionalities

In this section, we prove that our new functionality $\mathcal{F}_{\mathsf{NICOM}}$ implies the old UC commitment functionality of [CF01] by constructing a protocol in Fig. 20 that implements the $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ functionality (Fig. 19) in the $\mathcal{F}_{\mathsf{NICOM}}$ model. We prove security by proving the following theorem.

Theorem 10. *The protocol in Fig. 20 securely implements the $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ functionality (Fig. 19) in the $\mathcal{F}_{\mathsf{NICOM}}$ model.*

Fig. 19. Non-Interactive Commitment Functionality $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ [CF01]

$\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ proceeds as follows, running with parties (P, V) and an adversary $\mathcal{S}$.

- Upon receiving a value $(\mathsf{Com}, \mathsf{ssid}, m)$ from P, record the tuple (ssid, m) and send the message $(\mathsf{Receipt}, \mathsf{ssid})$ to V and $\mathcal{S}$. Ignore subsequent $(\mathsf{Com}, \mathsf{ssid}, \ldots)$ values.
- Upon receiving a value $(\mathsf{Open}, \mathsf{ssid})$ from P, proceed as follows: If the tuple (ssid, m) is recorded then send the message $(\mathsf{verified}, \mathsf{ssid}, m)$ to V and $\mathcal{S}$. Otherwise, do nothing.
- When receiving $(\mathsf{Corrupt}, \mathsf{ssid})$ from $\mathcal{S}$, mark ssid as corrupted. Send the stored tuple of the form $(\mathsf{ssid}, \ldots)$ to $\mathcal{S}$. If there does not exist any tuple then send $(\mathsf{ssid}, \perp)$ to $\mathcal{S}$.

Proof. We describe the ideal-model adversary $\mathcal{S}$. This adversary runs an execution with the environment $\mathcal{Z}$ and, in parallel, simulates a virtual copy of the real-life dummy adversary $\mathcal{A}$ in a black-box way. $\mathcal{S}$ acts as an interface between $\mathcal{A}$ and $\mathcal{Z}$ by imitating a copy of a real execution of the protocol for $\mathcal{A}$, incorporating $\mathcal{Z}$'s ideal-model interactions and vice versa forwarding $\mathcal{A}$'s messages to $\mathcal{Z}$. More precisely, firstly we consider the case where the committer is honest and gets corrupted post-execution. Secondly, we consider the case where the committer is statically corrupt.

Fig. 20. Protocol implementing $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ in $\mathcal{F}_{\mathsf{NICOM}}$ model

---
The parties participate in a session and subsession with identifiers sid and ssid respectively.

- $\mathsf{Com}(\mathsf{ssid}, m)$: The committer invokes $\mathcal{F}_{\mathsf{NICOM}}$ with input $(\mathsf{Com}, \mathsf{ssid}, m)$ to obtain $(\mathsf{Receipt}, \mathsf{ssid}, \pi, d)$. Send π as the commitment to verifier V and store d as the decommitment. Upon receiving π, the verifier V outputs $(\mathsf{Receipt}, \mathsf{ssid})$.
- $\mathsf{Open}(\mathsf{ssid})$ Phase: The committer sends (m, d) to the verifier. Upon obtaining (m, d), the verifier invokes $\mathcal{F}_{\mathsf{NICOM}}$ with input $(\mathsf{Open}, \mathsf{ssid}, m, \pi, d)$ to obtain (v, ssid) where $v \in \{\mathsf{verified}, \mathsf{verification\text{-}failed}\}$. If $(v == \mathsf{verified})$ the verifier outputs m, otherwise do nothing.
---

1. If at some point in the execution the environment $\mathcal{Z}$ writes a message of the form - $(\mathsf{Com}, \mathsf{ssid}, m)$ on the tape of the uncorrupted committer P and copies this to the functionality $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$, then the ideal-model simulator $\mathcal{S}$ who cannot read m, but is informed about the commitment by receiving $(\mathsf{Receipt}, \mathsf{ssid})$ from $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$. The simulator obtains $\mathcal{S}_C$ from $\mathcal{Z}$ and activates $\mathcal{F}_{\mathsf{NICOM}}$ with $\mathcal{S}_C$ algorithm if this is the first activation. Else, the simulator invokes $\mathcal{S}_C$ with input $(\mathsf{Com}, \mathsf{ssid})$ to obtain (π, st). $\mathcal{S}$ sends π internally to the simulated verifier. If the verifier is corrupted by $\mathcal{A}$, then $\mathcal{S}$ directly sends π to $\mathcal{A}$.
2. When $\mathcal{Z}$ instructs the adversary to corrupt the uncorrupted committer, the ideal world adversary $\mathcal{S}$ writes a message $(\mathsf{Corrupt}, \mathsf{ssid})$ on the tape of the uncorrupted committer P and copies this to the functionality $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$, then the functionality marks ssid as corrupted and sends (ssid, m) to the ideal world adversary $\mathcal{S}$. If $(m \neq \perp)$, $\mathcal{S}$ invokes $\mathcal{S}_C$ with input

(Equiv, ssid, π, st, m) to obtain (d, st) and the simulator invokes $\mathcal{F}_{\mathsf{NICOM}}$ functionality with input (Open, ssid, m, π, d) to obtain (v, ssid). $\mathcal{S}$ forwards the corruption request by $\mathcal{Z}$ to the dummy adversary $\mathcal{A}$ to corrupt the committer in the simulated execution.

3. When the environment $\mathcal{Z}$ instructs the dummy adversary $\mathcal{A}$ to corrupt the prover and sends a commitment π to the ideal-world adversary simulator, $\mathcal{S}$ performs the following: The simulator obtains $\mathcal{S}_C$ from $\mathcal{Z}$ and activates $\mathcal{F}_{\mathsf{NICOM}}$ with $\mathcal{S}_C$ algorithm if this is the first activation. The simulator instructs the dummy adversary to corrupt the prover and send π to the simulated verifier. Upon obtaining π from the corrupt prover, the simulator invokes $\mathcal{S}_C(\mathsf{Ext}, \mathsf{ssid}, \pi)$ to obtain (m, st). If $m \neq \bot$, then $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ with input (Com, ssid, m). If $m == \bot$, then do nothing.

    When $\mathcal{A}$ opens the commitment π by providing input (m', d') then the simulator invokes $\mathcal{F}_{\mathsf{NICOM}}$ with input (Open, ssid, m, d) to obtain v from the functionality. If ($v ==$ verified) then $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ with input (Open, ssid).

At the end of the execution, the dummy adversary forwards its view to $\mathcal{S}$, who forwards its view to the environment $\mathcal{Z}$ as the ideal world adversary view.

Indistinguishability. We prove that the real world adversary view is indistinguishable from the ideal world adversary view in the $\mathcal{F}_{\mathsf{NICOM}}$ model as follows:

1. In the real world, the verifier obtains a proof π obtained from $\mathcal{F}_{\mathsf{NICOM}}$ which is generated by $\mathcal{S}_C$. In the ideal world the simulator invokes $\mathcal{S}_C$ to generate the simulated proof π and send it to the simulated verifier or $\mathcal{A}$ (if the verifier is corrupted). The two proofs are identically distributed in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

2. In the real world, when the prover is uncorrupted during proof generation $\mathcal{S}_C$ generates (π, st) independent of the message and then returns the opening d upon being invoked on message m. The tuple $(\mathsf{ssid}, m, \pi, d, \mathsf{st})$ is stored by $\mathcal{F}_{\mathsf{NICOM}}$. When $\mathcal{Z}$ instructs the $\mathcal{A}$ to corrupt prover in subsession ssid, $\mathcal{F}_{\mathsf{NICOM}}$ returns $(\mathsf{ssid}, m, \pi, d, \mathsf{st})$ to $\mathcal{A}$. In the ideal world, upon obtaining the corruption request the simulator obtains (π, st) by invoking the simulation algorithm $\mathcal{S}_C(\mathsf{Com}, \mathsf{ssid})$ and sets it as the simulated proof. When $\mathcal{A}$ corrupts the prover, the simulator obtains m and equivocates π by invoking $\mathcal{S}_C(\mathsf{Equiv}, \mathsf{ssid}, \pi, \mathsf{st}, m)$ to obtain (d, st). The simulator forces $\mathcal{F}_{\mathsf{NICOM}}$ to store this tuple by invoking $\mathcal{F}_{\mathsf{NICOM}}$ with (Open, ssid, m, π, d). This is an unregistered tuple and so $\mathcal{F}_{\mathsf{NICOM}}$ stores it once its gets validated internally (in $\mathcal{F}_{\mathsf{NICOM}}$) by the $\mathcal{S}_C$ algorithm. Finally, the simulator forwards the corruption request (from $\mathcal{Z}$) to the dummy $\mathcal{A}$ to corrupt the prover in subsession ssid. The two views are identical in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

3. In the real world, the corrupt prover sends a commitment π to the honest verifier. When the prover opens the commitment by sending (m, π, d), the verifier is supposed to verify the commitment by invoking $\mathcal{F}_{\mathsf{NICOM}}$ with command (Open, ssid, m, π, d). In the ideal world, the simulator extracts the committed message from π by invoking $\mathcal{S}_C(\mathsf{Ext}, \mathsf{ssid}, \pi)$. When the dummy adversary provides the opening, the simulator invokes $\mathcal{F}_{\mathsf{NICOM}}$ with command (Open, ssid, m, π, d) to verify the opening. If the verification is successful then the simulator invokes $\mathcal{F}_{\mathsf{COM}}^{\mathsf{CF}}$ with command (Open, ssid). The output of the honest verifier is identically distributed in the two worlds in the $\mathcal{F}_{\mathsf{NICOM}}$ model.

$\square$