One-Sided Adaptively Secure Two-Party Computation

Carmit Hazay* Arpita Patra†

Abstract

Adaptive security is a strong security notion that captures additional security threats that are not addressed by static corruptions. For instance, it captures real-world scenarios where "hackers" actively break into computers, possibly while they are executing secure protocols. Studying this setting is interesting from both theoretical and practical points of view. A primary building block in designing adaptively secure protocols is a non-committing encryption (NCE) that implements secure communication channels in the presence of adaptive corruptions. Current constructions require a number of public key operations that grows linearly with the length of the message. Furthermore, general two-party protocols require a number of NCE calls that is dependent both on the circuit size and the security parameter.

In this paper we study the two-party setting in which at most one of the parties is adaptively corrupted, which we believe is the right security notion in the two-party setting. We study the feasibility of (**1**) NCE with constant number of public key operations for large message spaces. (**2**) Oblivious transfer with constant number of public key operations for large sender's input spaces, and (**3**) constant round secure computation protocols with an overall number of public key operations that is linear in the circuit size. Our study demonstrates that such primitives indeed exist in the presence of single corruptions, while this is not known for fully adaptive security (where both parties may get corrupted).

Keywords: Secure Two-Party Computation, Adaptive Security, Non-Committing Encryption, Oblivious Transfer

*Department of Computer Engineering, Bar-Ilan University, Israel. Email: carmit.hazay@biu.ac.il.

†Applied Statistics Unit, ISI Kolkata, India. Email: arpitapatra10@gmail.com.

1 Introduction

1.1 Background

Secure two-party computation. In the setting of secure two-party computation, two parties with private inputs wish to jointly compute some function of their inputs while preserving certain security properties like privacy, correctness and more. In this setting, security is formalized by viewing a protocol execution as if the computation is executed in an ideal setting where the parties send inputs to a trusted party that performs the computation and returns its result (also known by simulation-based security). Starting with the work of [Yao82, GMW87], it is by now well known that (in various settings) any polynomial-time function can be compiled into a secure function evaluation protocol with practical complexity; see [BDOZ11, LP12, DPSZ12, NNOB12] for a few recent works. The security proofs of these constructions assume that a party is statically corrupted. Meaning, corruptions take place at the outset of the protocol execution and the identities of the corrupted parties are fixed throughout the computation. Adaptive security is a stronger notion where corruptions takes place *at any point* during the course of the protocol execution. That is, upon corruption the adversary sees the internal data of the corrupted party which includes its input, randomness and the incoming messages. This notion is much stronger than static security due to the fact that the adversary may choose at any point which party to corrupt, even after the protocol is completed! It therefore models more accurately real world threats.

Typically, when dealing with adaptive corruptions we distinguish between corruptions with erasures and without erasures. In the former case honest parties are trusted to erase data if are instructed to do so by the protocol, whereas in the latter case no such assumption is made. This assumption is often problematic since it relies on the willingness of the honest parties to carry out this instruction without the ability to verify its execution. In settings where the parties are distrustful it may not be a good idea to base security on such an assumption. In addition, it is generally unrealistic to trust parties to fully erase data since this may depend on the operating system. Nevertheless, assuming that there are no erasures comes with a price since the complexity of adaptively secure protocols without erasures is much higher than the analogue complexity of protocols that rely on erasures. In this paper we do *not* rely on erasures.

Adaptive security. It is known by now that security against adaptive attacks captures important real-world concerns that are not addressed by static corruptions. For instance, such attacks capture scenarios where "hackers" actively break into computers, possibly while they are running secure protocols, or when the adversary learns from the communication which parties are worth to corrupt more than others. This later issue can be demonstrated by the following example. Consider a protocol where some party (denoted by the dealer) shares a secret among a public set of $\sqrt{n}$ parties, picked at random from a larger set of n parties. This scheme is insecure in the adaptive model if the adversary corrupts $\sqrt{n}$ parties since it can always corrupt the particular set of parties that share the secret. In the static setting the adversary corrupts the exact same set of parties that share the secret with a negligible probability in n.

Other difficulties also arise when proving security. Consider the following protocol for transferring a message: A receiver picks a public key and sends it to a sender that uses it to encrypt its message. Then, security in the static model is simple and relies on the semantic security of the underlying encryption scheme. However, this protocol is insecure in the adaptive model since standard semantically secure encryption binds the receiver to a single message (meaning, given the public key, a ciphertext can only be decrypted into a single value). Thus, upon corrupting the receiver at the end of the protocol execution it would not be possible to "explain" the simulated ciphertext with respect to the real message. This implies that adaptive security is much harder to achieve.

Adaptively secure two-party computation. In the two-party setting there are scenarios where the system is comprised from two devices communicating between themselves without being part of a bigger system. For instance, consider a scenario where two devices share an access to an encrypted database that contains highly sensitive data (like passwords). Moreover, the devices communicate via secure computation but do not communicate with other devices due to high risk of breaking into the database. Thus, attacking one of the devices does not disclose any useful information about the content of the database, while attacking both devices is a much harder task. It is reasonable to assume that the devices are not necessarily statically corrupted since they are protected by other means, while attackers may constantly try to break into these devices (even while running secure computation).

In 2011, RSA secureID authentication products were breached by hackers that leveraged the stolen information from RSA in order to attack the U.S. defense contractor Lockheed Martin. The attackers targeted SecurID data as part of a broader scheme to steal defense secrets and related intellectual property. Distributing the SecureID secret keys between two devices potentially enables to defend against such an attack since in order to access these keys the attackers need to *adaptively* corrupt both devices, which is less likely to occur. Many other applications face similar threats when attempt to securely protect their databases.

We therefore focus on a security notion that seems the most appropriate in this context. In this paper, we study secure two-party computation with single adaptive corruptions in the non-erasure model where at most one party is adaptively corrupted. To distinguish this notion from fully adaptive security, where both parties may get corrupted, we denote it by *one-sided* adaptive security. Our goal in this work is to make progress in the study of the efficiency of two-party protocols with one-sided security. Our measure of efficiency is the number of public key encryption (PKE) operations. Loosely speaking, our primitives are parameterized by a public key encryption scheme for which we count the number of key generation/encryption/decryption operations. More concretely, these operations are captured by the number of exponentiations in several important groups (e.g., groups where the DDH assumption is hard and composite order groups where the assumptions DCR and QR are hard), and further considered in prior works such as [GWZ09]. Finally, our proofs are given in the universal composable (UC) setting [Can01] with a common reference string (CRS) setup. The reductions of our non-committing encryption and oblivious transfer with one-sided security are tight. The reductions of our general two-party protocols are tighter than proofs in prior works; see more details below. All our theorems *are not* known to hold in the fully adaptive setting.

1.2 Our Results

One-sided NCE with constant overhead. A non-committing encryption (NCE) scheme [CFGN96] implements secure channels in the presence of adaptive corruptions and is an important building block in designing adaptively secure protocols. In [DN00], Damgård and Nielsen presented a theoretical improvement in the one-sided setting by designing an NCE under strictly weaker assumptions than simulatable public key encryption scheme (the assumption for fully adaptive NCE). Nevertheless, all known one-sided [CFGN96, DN00] and fully adaptive NCE constructions [DN00, CDSMW09a] require $\mathcal{O}(1)$ PKE operations for each transmitted bit. It was unknown whether this bound can be reduced for one-sided NCEs and even matched with the overhead of standard PKEs.

We suggest a new approach for designing NCEs secure against one-sided adaptive attacks. Our protocols are built on two cryptographic building blocks that are non-committing with respect to a single party. We denote these by NCE for the sender and NCE for the receiver. *Non-committing for the receiver* (NCER) implies that one can efficiently generate a secret key that decrypts a simulated ciphertext into any plaintext. Whereas *non-committing for the sender* (NCES) implies that one can efficiently generate randomness for any plaintext for proving that a ciphertext, encrypted under a fake key, encrypts this plaintext. A core building block in our one-sided construction is (a variant) of the following protocol, in which the receiver

generates two sets of public/secret keys; one pair of keys for each public key system, and sends these public keys to the sender. Next, the sender partitions its message into two shares and encrypts the distinct shares under the distinct public keys. Finally, the receiver decrypts the ciphertexts and reconstructs the message. Both NCES and NCER are semantically secure PKEs that are as efficient as standard PKEs. Informally, we prove that,

Theorem 1.1 (Informal) *Assume the existence of NCER and NCES with constant number of PKE operations for message space $\{0,1\}^q$ and simulatable PKE. Then there exists a one-sided NCE with constant number of PKE operations for message space $\{0,1\}^q$, where $q = O(n)$ and n is the security parameter.*

Importantly, the security of this protocol only works if the simulator knows the identity of the corrupted party since fake public keys and ciphertexts cannot be explained as valid ones. We resolve this issue by slightly modifying this protocol using somewhat NCE [GWZ09] in order to encrypt only three bits. Namely, we use somewhat NCE to encrypt the *choice* of having fake/valid keys and ciphertexts (which only requires a single non-committing bit per choice). This enables the simulator to "explain" fake keys/ciphertext as valid and vice versa using only a constant number of asymmetric operations. In this work we consider two implementations of NCER and NCES. For polynomial-size message spaces the implementations are secure under the DDH assumption, whereas for exponential-size message spaces security holds under the DCR assumption. The NCER implementations are taken from [JL00, CHK05]. NCES was further discussed in [FHKW10] and realized under the DDH assumption in [BHY09] using the closely related notion of lossy encryption.[1] In this paper we realize NCES under the DCR assumption.

One-sided oblivious transfer with constant overhead. We use our one-sided NCEs to implement 1-out-of-2 oblivious transfer (OT) between a sender and a receiver. We consider a generic framework that abstracts the statically secure OT of [PVW08] that is based on a dual-mode PKE primitive, while encrypting only a small portion of the communication using our one-sided NCE. Our construction requires a constant number of PKE operations for an input space $\{0,1\}^q$ of the sender, where $q = O(n)$. This is significantly better than the fully adaptively secure OT of [GWZ09] (currently the most efficient fully adaptive construction), that requires $\mathcal{O}(q)$ such operations. We prove that,

Theorem 1.2 (Informal) *Assume the existence of one-sided NCE with constant number of PKE operations for message space $\{0,1\}^q$ and dual-mode PKE. Then there exists a one-sided OT with constant number of PKE operations for sender's input space $\{0,1\}^q$, where $q = O(n)$ and n is the security parameter.*

We build our one-sided OT based on the PVW protocol with the following modifications. (**1**) First, we require that the sender sends its ciphertexts via a one-sided non-committing channel (based on our previous result, this only inflates the overhead by a constant). (**2**) We fix the common parameters of the dual-mode PKE in a single mode (instead of alternating between two modes as in the [GWZ09] protocol). To ensure correctness, we employ a special type of ZK PoK which uses a novel technique; see below for more details. Finally, we discuss two instantiations of ZK PoK based on the DDH and QR assumptions.

Constant round one-sided secure computation. Theoretically, it is well known that any statically secure protocol can be transformed into a one-sided adaptively secure protocol by encrypting the *entire* communication using NCE. This approach, adopted by [KO04], implies that the number of PKE operations grows

[1] This notion differs from NCES by not requiring an efficient opening algorithm that enables to equivocate the ciphertext's randomness. We further observe that the notion of NCES is also similar to mixed commitments [DN02].

linearly with the circuit size times a computational security parameter.[2] A different approach in the OT-hybrid model was taken in [IPS08] and achieved a similar overhead as well.

In this work we demonstrate the feasibility of designing better generic constant round protocols based on Yao's garbled circuit technique with one-sided security, tolerating semi-honest and malicious attacks. Our main observation implies that one-sided security can be obtained if the keys corresponding to the inputs and output wires, as well as the garbled circuit are communicated via a one-sided adaptively secure channel. Using our one-sided secure primitives we obtain protocols that outperform the constant round one-sided constructions of [KO04, IPS08] and all known generic fully adaptively secure two-party protocols. Specifically, we prove that

Theorem 1.3 (Informal) *Under the assumptions of achieving statically secure two-party computation and one-sided OT with constant number of PKE operations for sender's input space $\{0,1\}^q$, where $q = O(n)$ and n is the security parameter, there exists a* constant round *one-sided semi-honest adaptively secure two-party protocol that requires $\mathcal{O}(|C|) + |\mathsf{input}| + |\mathsf{output}|)$ public key operations.*

In order to obtain one-sided security against malicious attacks we adapt the cut-and-choose based protocol introduced in [LP12]. The idea of the cut-and-choose technique is to ask one party to send s garbled circuits and later open half of them by the choice of the other party. This ensures that with very high probability the majority of the unopened circuits are valid. Proving security in the one-sided setting requires dealing with new subtleties and requires a modified cut-and-choose OT protocol, since [LP12] defines the public parameters of their cut-and-choose OT protocol in a way that precludes the equivocation of the receiver's input. Our result in the malicious setting follows.

Theorem 1.4 (Informal.) *Under the assumptions of achieving static security in [LP12], one-sided cut-and-choose OT with constant number of PKE operations for sender's input space $\{0,1\}^q$, where $q = O(n)$ and n is the security parameter, and simulatable PKE, there exists a* constant round *one-sided malicious adaptively secure two-party protocol that requires $\mathcal{O}(s(|C| + |\mathsf{input}| + |\mathsf{output}|))$ public key operations where s is a statistical parameter that determines the cut-and-choose soundness error.*

Our protocols are simpler than the prior protocols [KO04, IPS08].

Witness equivocal UC ZK PoK for compound statements. As a side result, we demonstrate a technique for efficiently generating statically secure UC ZK PoK for known Σ-protocols. Our protocols use a new approach where the prover commits to an additional transcript which enables to extract the witness with a constant overhead in the CRS setting. We further focus on compound statements (where the statement is comprised of sub-statements for which the prover only knows a subset of the witnesses), and denote a UC ZK PoK by *witness equivocal* if the simulator knows the witnesses for *all* sub-statements but not which subset is known to the real prover. We extend our proofs for this notion to the adaptive setting as well. In particular, the simulator must be able to convince an adaptive adversary that it does *not know* a different subset of witnesses. This notion is weaker than the typical one-sided security notion (that requires simulation without the knowledge of any witness), but is still meaningful in designing one-sided secure protocols. In this work, we build witness equivocal UC ZK PoKs for a class of fundamental compound Σ-protocols, without relying on NCE. Our protocols are round efficient and with a negligible soundness error and UC secure [Can01].

To conclude, our results may imply that one-sided security is strictly easier to achieve than fully adaptive security, and for some applications this is indeed the right notion to consider. We leave open the efficiency of

[2] We note that this statement is valid regarding protocols that do not employ fully homomorphic encryptions (FHE). To this end, we only consider protocols that do not take the FHE approach. As a side note, it was recently observed in [KTZ13] that adaptive security is impossible for FHE satisfying compactness.

constant round one-sided secure protocols in the multi-party setting. Currently, it is not clear how to extend our techniques beyond the two-party setting (such as for the [BMR90] protocol). Another open problem whether it is feasible is to achieve secure constructions with a number of PKE operations that is strictly less than what we achieve here.

1.3 Prior Work

We describe prior work on NCE, adaptively secure OT and two-party computation.

Non-committing encryption. One-sided NCE was introduced in [CFGN96] which demonstrated feasibility of the primitive under the RSA assumption. Next, NCE was studied in [DN00, CDSMW09a]. The construction of [DN00] requires constant rounds on the average and is based on simulatable PKE, whereas [CDSMW09a] presents an improved expected two rounds NCE based on a weaker primitive. [DN00] further presented a one-sided NCE based on a weakened simulatable PKE notion. The computational overhead of these constructions is $\mathcal{O}(1)$ PKE operations for each transmitted bit. An exception is the somewhat NCE introduced in [GWZ09] (see Section 3.3 for more details). This primitive enables to send arbitrarily long messages at the cost of $\log \ell$ PKE operations, where ℓ is the equivocality parameter that determines the number of messages the simulator needs to explain. This construction improves over NCEs for sufficiently small ℓ's. Finally, in [Nie02] Nielsen proved that adaptively secure non-interactive encryption scheme must have a decryption key that is at least as long as the transmitted message.

Adaptively secure oblivious transfer. [Bea97, CLOS02] designed semi-honest adaptively secure OT (using NCE) and then compiled it into the malicious setting using generic ZK proofs. More recently, in a weaker model that assumes erasures, Lindell [Lin09] used the method of [WW06] to design an efficient transformation from any static OT to a semi-honest composable adaptively secure OT. Another recent work by Garay *et al.* [GWZ09] presented a UC adaptively secure OT, building on the static OT of [PVW08] and somewhat NCE. This paper introduces an OT protocol with security under a weaker *semi-adaptive* notion, that is then compiled into a fully adaptively secure OT by encrypting the transcript of the protocol using somewhat NCE.[3] Finally, [CDSMW09b] presented an improved compiler for a UC adaptively secure OT in the malicious setting (using NCE as well).

Adaptively secure two-party computation. In the non-erasure model, adaptively secure computation has been extensively studied [CLOS02, DN03, CDD$^+$04, KO04, IPS08, Lin09, CDSMW09a, CDSMW09b, GS12]. Starting with the work of [CLOS02], it is known by now how to adaptively compute any well-formed two-party functionality. The followup work of [DN03] showed how to use a threshold encryption to achieve UC adaptive security but requires honest majority. A generic compiler from static to adaptive security was shown in [CDD$^+$04] (yet without considering post-execution corruptions). The work by Katz and Ostrovsky [KO04] studied the round complexity in the one-sided setting. Their protocol is the first round efficient construction, yet it takes the naive approach of encrypting the entire communication using NCE. Moreover, the work of [IPS08] provided a one-sided adaptively secure protocol, given a honest majority adaptively secure protocol (where the particular instantiation uses the constant rounds protocol from [DI05]), and the overall number of PKE operations is $O(s^3|C|n)$ for a statistical and computational security parameter s and n. Finally, a recent work by Garg and Sahai [GS12] shows adaptively secure constant round protocols tolerating $m-1$ out of m corrupted parties using a non-black box simulation approach. Their approach uses the OT hybrid compiler of [IPS08].

In the erasure model, one of the earliest works by Beaver and Haber [BH92] showed an efficient generic transformation from adaptively secure protocols with ideally secure communication channels, to

[3] We stress that the semi-adaptive notion is incomparable to the one-sided notion since the former assumes that either one party is statically corrupted or none of the parties get corrupted.

adaptively secure protocols with standard (authenticated) communication channels. A more recent work by Lindell [Lin09] presents an efficient semi-honest constant round two-party protocol with adaptive security.

2 Preliminaries

We denote the security parameter by n. A function $\mu(\cdot)$ is *negligible* if for every polynomial $p(\cdot)$ there exists a value N such that for all $n > N$ it holds that $\mu(n) < \frac{1}{p(n)}$. We write PPT for probabilistic polynomial-time. We denote the message spaces of our non-committing encryption schemes and the message space of the sender in our OT protocols by $\{0,1\}^q$ for $q = O(n)$.

We specify the definitions of computational indistinguishability and statistical distance.

Definition 2.1 (Computational indistinguishability by circuits) *Let $X = \{X_n(a)\}_{n\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y_n(a)\}_{n\in\mathbb{N},a\in\{0,1\}^*}$ be distribution ensembles. We say that X and Y are* computationally indistinguishable*, denoted $X \approx_c Y$, if for every family $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ of polynomial-size circuits, there exists a negligible function $\mu(\cdot)$ such that for all $a \in \{0,1\}^*$,*

$$|\Pr[\mathcal{C}_n(X_n(a)) = 1] - \Pr[\mathcal{C}_n(Y_n(a)) = 1]| < \mu(n).$$

Definition 2.2 (Statistical distance) *Let X_n and Y_n be random variables accepting values taken from a finite domain $\Omega \subseteq \{0,1\}^n$. The* statistical distance *between X_n and Y_n is*

$$SD(X_n, Y_n) = \frac{1}{2}\sum_{\omega\in\Omega} |\Pr[X_n = \omega] - \Pr[Y_n = \omega]|.$$

*We say that X_n and Y_n are ϵ-*close *if their statistical distance is at most $SD(X_n, Y_n) \leq \epsilon(n)$. We say that X_n and Y_n are* statistically close*, denoted $X_n \approx_s Y_n$, if $\epsilon(n)$ is negligible in n.*

2.1 Public Key Encryption Scheme

We specify the definitions of public key encryption and IND-CPA.

Definition 2.3 (PKE) *We say that $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a* public-key encryption scheme *if $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ are polynomial-time algorithms specified as follows:*

- Gen*, given a security parameter n (in unary), outputs keys $(\mathrm{PK}, \mathrm{SK})$, where PK is a public key and SK is a secret key. We denote this by $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^n)$.*
- Enc*, given the public key PK and a plaintext message m, outputs a ciphertext c encrypting m. We denote this by $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m)$; and when emphasizing the randomness r used for encryption, we denote this by $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m; r)$.*
- Dec*, given the public key PK, secret key SK and a ciphertext c, outputs a plaintext message m s.t. there exists randomness r for which $c = \mathsf{Enc}_{\mathrm{PK}}(m; r)$ (or $\perp$ if no such message exists). We denote this by $m \leftarrow \mathsf{Dec}_{\mathrm{PK},\mathrm{SK}}(c)$.*

For a public key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a non-uniform probabilistic adversary $\text{ADV} = (\text{ADV}_1, \text{ADV}_2)$, we consider the following *IND-CPA game*:

$(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^n)$.
$(m_0, m_1, history) \leftarrow \text{ADV}_1(\text{PK})$, s.t. $|m_0| = |m_1|$.
$c \leftarrow \mathsf{Enc}_{\text{PK}}(m_b)$, where $b \in_R \{0,1\}$.
$b' \leftarrow \text{ADV}_2(c, history)$.
ADV wins if $b' = b$.

Denote by $\mathbf{Adv}_{\Pi,\text{ADV}}(n)$ the probability that ADV wins the IND-CPA game.

Definition 2.4 (IND-CPA) *A public key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *has* indistinguishable encryptions under chosen plaintext attacks (IND-CPA)*, if for every non-uniform probabilistic adversary* $\text{ADV} = (\text{ADV}_1, \text{ADV}_2)$ *there exists a negligible function* negl *such that* $\mathbf{Adv}_{\Pi,\text{ADV}}(n) \leq \frac{1}{2} + \mathsf{negl}(n)$.

We say that a protocol π *realizes functionality* $\mathcal{F}$ *with* t *PKE operations* (relative to Π) if the number of calls π makes to either one of $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is at most t. Importantly, this definition is not robust in the sense that one might define an encryption algorithm Enc' that consists of encrypting n times in parallel using Enc. In this work we do not abuse this definition and achieve a single basic operation relative to algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, which are implemented by $O(1)$ group exponentiations in various group descriptions.

2.2 Simulatable Public Key Encryption

A simulatable public key encryption scheme is an IND-CPA secure PKE with four additional algorithms. I.e., an oblivious public key generator $\widetilde{\mathsf{Gen}}$ and a corresponding key faking algorithm $\widetilde{\mathsf{Gen}}^{-1}$, and an oblivious ciphertext generator $\widetilde{\mathsf{Enc}}$ and a corresponding ciphertext faking algorithm $\widetilde{\mathsf{Enc}}^{-1}$. Intuitively, the key faking algorithm is used to explain a legitimately generated public key as an obliviously generated public key. Similarly, the ciphertext faking algorithm is used to explain a legitimately generated ciphertext as an obliviously generated one.

Definition 2.5 (Simulatable PKE [DN00]) *A Simulatable PKE is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Gen}}^{-1}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Enc}}^{-1})$ *that satisfy the following properties:*

- **IND-CPA.** $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CPA secure as in Definition 2.4.*
- **Oblivious public key generation.** *Consider the experiment* $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^n)$, $r \leftarrow \widetilde{\mathsf{Gen}}^{-1}(\text{PK})$ *and* $\text{PK}' \leftarrow \widetilde{\mathsf{Gen}}(r')$. *Then,* $(r, \text{PK}) \approx_c (r', \text{PK}')$.
- **Oblivious ciphertext generation.** *For any message* m *in the appropriate domain, consider the experiment* $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^n)$, $c \leftarrow \mathsf{Enc}_{\text{PK}}(m)$, $r \leftarrow \widetilde{\mathsf{Enc}}^{-1}(c)$, $c' \leftarrow \widetilde{\mathsf{Enc}}_{\text{PK}}(r')$. *Then* $(\text{PK}, r, c) \approx_c (\text{PK}, r', c')$.

The El Gamal PKE [Gam85] is one example for simulatable PKE.

2.3 Dual-Mode PKE

A dual-mode PKE Π_{DUAL} is specified by the algorithms $(\mathsf{Setup}, \mathsf{dGen}, \mathsf{dEnc}, \mathsf{dDec}, \mathsf{FindBranch}, \mathsf{TrapKeyGen})$ described below.

- Setup is the system parameters generator algorithm. Given a security parameter n and a mode $\mu \in \{0,1\}$, the algorithm outputs (CRS, t). The CRS is a common string for the remaining algorithms, and t is a trapdoor value that is given to either $\mathsf{FindBranch}$ or $\mathsf{TrapKeyGen}$, depends on the mode. The setup algorithms for messy and decryption modes are denoted by $\mathsf{SetupMessy}$ and $\mathsf{SetupDecryption}$, respectively; namely $\mathsf{SetupMessy} := \mathsf{Setup}(1^n, 0)$ and $\mathsf{SetupDecryption} := \mathsf{Setup}(1^n, 1)$.
- dGen is the key generation algorithm that takes a bit α and the CRS as input. If $\alpha = 0$, then it generates left public and secret key pair. Otherwise, it creates right public and secret key pair.
- dEnc is the encryption algorithm that takes a bit β, a public key PK and a message m as input. If $\beta = 0$, then it creates the left encryption of m, else it creates the right encryption.
- dDec decrypts a message given a ciphertext and a secret key SK.
- $\mathsf{FindBranch}$ finds whether a given public key (in messy mode) is left key or right key given the messy mode trapdoor t.
- $\mathsf{TrapKeyGen}$ generates a public key and two secret keys using the decryption mode trapdoor t such that both left encryption as well as the right encryption using the public key can be decrypted using the secret keys.

Definition 2.6 (Dual-mode PKE) *A dual-mode PKE is a tuple of algorithms described above that satisfy the following properties:*

1. **Completeness.** *For every mode* $\mu \in \{0,1\}$*, every* $(\mathrm{CRS}, t) \leftarrow \mathsf{Setup}(1^n, \mu)$*, every* $\alpha \in \{0,1\}$*, every* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{dGen}(\alpha)$*, and every* $m \in \{0,1\}^\ell$*, decryption is correct when the public key type matches the encryption type, i.e.,* $\mathsf{dDec}_{\mathrm{SK}}(\mathsf{dEnc}_{\mathrm{PK}}(m, \alpha)) = m$.
2. **Indistinguishability of modes.** *The CRS generated by* $\mathsf{SetupMessy}$ *and* $\mathsf{SetupDecryption}$ *are computationally indistinguishable, i.e.,* $\mathsf{SetupMessy}(1^n) \approx_c \mathsf{SetupDecryption}(1^n)$.
3. **Trapdoor extraction of key type (messy mode).** *For every* $(\mathrm{CRS}, t) \leftarrow \mathsf{SetupMessy}(1^n)$ *and every (possibly malformed)* PK*,* $\mathsf{FindBranch}(t, \mathrm{PK})$ *outputs the public key type* $\alpha \in \{0,1\}$*. Encryption at branch* $1-\alpha$ *is then message-lossy; namely, for every* $m_0, m_1 \in \{0,1\}^\ell, \mathsf{dEnc}_{\mathrm{PK}}(m_0, 1-\alpha) \approx_s \mathsf{dEnc}_{\mathrm{PK}}(m_1, 1-\alpha)$.
4. **Trapdoor generation of keys decrypt both branches (decryption mode).** *For every* $(\mathrm{CRS}, t) \leftarrow \mathsf{SetupDecryption}(1^n)$*,* $\mathsf{TrapKeyGen}(t)$ *outputs* $(\mathrm{PK}, \mathrm{SK}_0, \mathrm{SK}_1)$ *such that for every* α*,* $(\mathrm{PK}, \mathrm{SK}_\alpha) \approx_c \mathsf{dGen}(\alpha)$.

2.4 Hardness Assumptions

Our constructions rely on the following hardness assumptions.

Definition 2.7 (DDH) *We say that* the decisional Diffie-Hellman (DDH) problem is hard relative to $\mathcal{G}$ *if for all polynomial-sized circuits* $\mathcal{C} = \{\mathcal{C}_n\}$ *there exists a negligible function* negl *such that*

$$\left| \Pr\left[\mathcal{C}_n(\mathbb{G}, q, g, g^x, g^y, g^z) = 1\right] - \Pr\left[\mathcal{C}_n(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1\right] \right| \leq \mathsf{negl}(n),$$

where $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$ *and the probabilities are taken over the choices of* g *and* $x, y, z \in \mathbb{Z}_q$.

We require the DDH assumption to hold for prime order groups. In a few places we use a different version of the DDH assumption: for random generators $g, h \in \mathbb{G}$ and for distinct but otherwise random $a, b \in \mathbb{Z}_q$, the tuples (g, h, g^a, h^a) and (g, h, g^a, h^b) are computationally indistinguishable. This version of the DDH assumption is equivalent to the common form discussed above.

Definition 2.8 (DCR) *We say that* the Decisional Composite Residuosity (DCR) problem is hard relative to $\mathcal{G}$ *if for all polynomial-sized circuits* $\mathcal{C} = \{\mathcal{C}_n\}$ *there exists a negligible function* negl *such that*

$$\left| \Pr\left[\mathcal{C}_n(N, z) = 1 |\ z = y^N \bmod N^2\right] - \Pr\left[\mathcal{C}_n(N, z) = 1 |\ z = (1+N)^r \cdot y^N \bmod N^2\right] \right| \leq \mathsf{negl}(n),$$

where $N \leftarrow \mathcal{G}(1^n)$, N *is a random* n*-bit RSA composite,* r *is chosen at random in* $\mathbb{Z}_N$ *and the probabilities are taken over the choices of* N, y *and* r.

Definition 2.9 (QR) *We say that* the Quadratic Residuosity (QR) problem is hard relative to $\mathcal{G}$ *if for all polynomial-sized circuits* $\mathcal{C} = \{\mathcal{C}_n\}$ *there exists a negligible function* negl *such that*

$$\left| \Pr\left[\mathcal{C}_n(N, z) = 1 |\ z \leftarrow \mathbb{QR}_N\right] - \Pr\left[\mathcal{C}_n(N, z) = 1 |\ z \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\right] \right| \leq \mathsf{negl}(n),$$

where $N \leftarrow \mathcal{G}(1^n)$, N *is a random* n*-bit RSA composite,* $\mathbb{J}_N$ *denote the group of Jacobi symbol (+1) elements of* $\mathbb{Z}_N^*$, $\mathbb{QR}_N = \{x^2 : x \in \mathbb{Z}_N^*\}$ *denote* $\mathbb{J}_N$*'s subgroup of quadratic residues and the probabilities are taken over the choices of* N, z.

2.5 Zero-knowledge Proofs and Proofs of Knowledge

Our protocols employ zero-knowledge proofs (of knowledge) for assuring correct behavior. We formally define zero-knowledge and knowledge extraction as stated in [Gol01]. We then conclude with a definition of a Σ-protocol which constitutes a zero-knowledge proof of a special type.

Definition 2.10 (Interactive proof system) *A pair of* PPT *interactive machines* (P, V) *is called an* interactive proof system for a language L *if there exists a negligible function* negl *such that the following two conditions hold:*

1. COMPLETENESS: *For every* $x \in L$,

$$\Pr[\langle \mathsf{P}, \mathsf{V} \rangle(x) = 1] \geq 1 - \mathsf{negl}(|x|).$$

2. SOUNDNESS: *For every* $x \notin L$ *and every interactive* PPT*machine* B,

$$\Pr[\langle B, \mathsf{V} \rangle(x) = 1] \leq \mathsf{negl}(|x|).$$

Definition 2.11 (Zero-knowledge) *Let* (P, V) *be an interactive proof system for some language* L. *We say that* (P, V) *is* computational zero-knowledge *if for every* PPT *interactive machine* V^* *there exists a* PPT *algorithm* M^* *such that*

$$\{\langle \mathsf{P}, \mathsf{V}^* \rangle(x)\}_{x \in L} \approx_c \{\langle M^* \rangle(x)\}_{x \in L}$$

where the left term denote the output of V^* *after it interacts with* P *on common input* x *whereas, the right term denote the output of* M^* *on* x.

Definition 2.12 (Knowledge extraction) *Let R be a binary relation and $\kappa \to [0,1]$. We say that an interactive function* V *is* a knowledge verifier for the relation R with knowledge error κ *if the following two conditions holds:*

NON-TRIVIALITY: *There exists an interactive machine* P *such that for every $(x,y) \in \mathcal{R}$, (implying that $x \in L_{\mathcal{R}}$), all possible interactions of* V *with* P *on common input x and auxiliary input y are accepting.*

VALIDITY (WITH ERROR κ): *There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine K such that for every interactive function* P*, every $x \in L_{\mathcal{R}}$, and every machine K satisfies the following condition:*

> *Denote by $p(x,y,r)$ the probability that the interactive machine* V *accepts, on input x, when interacting with the prover specified by* $\mathsf{P}_{x,y,r}$ *that uses randomness r (where the probability is taken over the coins of* V*). If $p(x,y,r) > \kappa(|x|)$, then, on input x and with access to oracle* $\mathsf{P}_{x,y,r}$*, machine K outputs a solution $s \in \mathcal{R}(x)$ within an expected number of steps bounded by*
>
> $$\frac{q(|x|)}{p(x,y,r) - \kappa(|x|)}$$
>
> *The oracle machine K is called a universal knowledge extractor.*

Definition 2.13 (Σ-protocol) *A protocol π is a* Σ-protocol *for relation R if it is a 3-round public-coin protocol and the following requirements hold:*

- COMPLETENESS: *If* P *and* V *follow the protocol on input x and private input w to* P *where $(x,w) \in \mathcal{R}$, then* V *always accepts.*
- SPECIAL SOUNDNESS: *There exists a polynomial-time algorithm A that given any x and any pair of accepting transcripts $(a,e,z),(a,e',z')$ on input x, where $e \neq e'$, outputs w such that $(x,w) \in \mathcal{R}$.*
- SPECIAL HONEST-VERIFIER ZERO KNOWLEDGE: *There exists a* PPT*algorithm M^* such that*

$$\left\{\langle \mathsf{P}(x,w), \mathsf{V}(x,e)\rangle\right\}_{x \in L_{\mathcal{R}}} \approx_c \left\{M(x,e)\right\}_{x \in L_{\mathcal{R}}}$$

where $M(x,e)$ denotes the output of M upon input x and e, and $\langle \mathsf{P}(x,w), \mathsf{V}(x,e)\rangle$ denotes the output transcript of an execution between P *and* V*, where* P *has input (x,w),* V *has input x, and* V*'s random tape (determining its query) equals e.*

2.6 Security Definitions

In the following, we formalize the notion of UC one-sided adaptive security [Can01]. Formally, a two-party computation protocol is cast by specifying the participating parties P_0 and P_1 and a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_0, f_1)$ mapping pairs of inputs to pairs of outputs (one for each party). That is, for every pair of inputs $x_0, x_1 \in \{0,1\}^n$ the output pair is a random variable $(f_0(x_0,x_1), f_1(x_0,x_1))$ ranging over pair of strings. The first party with input x_0 wishes to receive $f_0(x_0,x_1)$, while the second party with input x_1 wishes to obtain $f_1(x_0,x_1)$.

2.6.1 One-sided Adaptive Security

In the two-party setting, a real execution of some protocol Π_f that implements f is run between two parties P_0 and P_1 in the presence of an adversary ADV and an environment ENV (that is given an input z, a random tape r_{ENV} and a security parameter n), and is modeled as a sequence of activations of the entities. ENV

is activated first and generates the inputs for the other entities. Then the protocol proceeds by having the parties communicating with each other, and ADV exchanges messages with ENV. Upon completing the real execution ENV outputs a bit.

In the ideal model, the computation involves an incorruptible trusted third party $\mathcal{F}_f$ which receives the parties' inputs, computes the function f on these inputs and returns to each party its respective output. The parties are replaced by dummy parties that do not communicate with each other, such that whenever a dummy party is activated it sends its input to the ideal functionality. Upon completing the ideal execution ENV outputs a bit. We say that a protocol Π_f UC realizes functionality $\mathcal{F}_f$ if for any real world adversary ADV there is a ideal world adversary SIM such that no ENV can tell with non-negligible probability whether it is interacting with ADV and the parties running Π_f in a real execution or with SIM and the dummy parties in an ideal execution. Details follow.

Execution in the real model. We now proceed with a real world execution, where a real two-party protocol is executed. Whenever ENV is activated, it first fixes input $x_i \in \{0,1\}^*$ for party P_i. Each party P_i then starts the execution with an input $x_i \in \{0,1\}^*$, a random tape r_i and a security parameter n. A *one-sided* adversary ADV is a probabilistic polynomial-time interactive Turing machine that is given a random tape r_{ADV} and a security parameter n. At the outset of the protocol, ADV receives some initial information from ENV. Then the computation proceeds in rounds such that in each round ADV sees all the messages sent between the parties. At the beginning of each round, ADV may choose to corrupt P_{i^*} for $i^* \in \{0,1\}$. Upon corrupting P_{i^*}, ADV learns its input and the random tape, and obtains some auxiliary information from ENV. In case ADV is malicious P_{i^*} follows ADV's instructions from the time it is corrupted. At the end of the protocol execution the honest parties locally compute their outputs and output the value specified by the protocol, whereas the corrupted party outputs a special symbol $\perp$. The adversary ADV outputs an arbitrary function of its internal state that includes, r_{ADV}, the messages received from ENV and the corrupted party's view. Next, a post-execution corruption process begins where ENV first learns the outputs. Then, ADV and ENV interact in at most one additional round. If none of the parties is corrupted yet, ENV can ask ADV to corrupt P_{i^*} for $i^* \in \{0,1\}$, receiving back the state of this party. At the end ENV outputs a bit.

Let f be as specified above and Π_f be a two-party protocol that computes f. We denote by the variable $\mathbf{OREAL}_{\Pi_f,\text{ADV},\text{ENV}}(n, x_0, x_1, z, \vec{r})$ the output of ENV on input z, random tape r_{ENV} and a security parameter n upon interacting with ADV and parties P_0, P_1 that engage in protocol Π_f on inputs r_{ADV} and $(x_0, r_0), (x_1, r_1)$, respectively, where $\vec{r} = (r_{\text{ENV}}, r_{\text{ADV}}, r_0, r_1)$. Let $\mathbf{OREAL}_{\Pi_f,\text{ADV},\text{ENV}}(n, x_0, x_1, z)$ denote a random variable describing $\mathbf{OREAL}_{\Pi_f,\text{ADV},\text{ENV}}(n, x_0, x_1, z, \vec{r})$ where the random tapes are chosen uniformly. Let $\mathbf{OREAL}_{\Pi_f,\text{ADV},\text{ENV}}$ denote the distribution ensemble:

$$\{\mathbf{OREAL}_{\Pi_f,\text{ADV},\text{ENV}}(n, x_0, x_1, z)\}_{x_0,x_1,z\in\{0,1\}^*,n\in\mathbb{N}}.$$

Execution in the ideal model. A one-sided ideal world adversary SIM is a probabilistic polynomial-time interactive Turing machine that is given a random tape r_{SIM} and a security parameter n. The ideal process is defined with respect to a trusted party that implements functionality $\mathcal{F}_f$ as follows:

First corruption phase: SIM receives some auxiliary information from ENV. Next, SIM may decide whether to corrupt party P_{i^*} for $i^* \in \{0,1\}$. Upon corrupting party P_{i^*}, SIM learns its input x_{i^*}. In addition, ENV hands some auxiliary information to SIM.

Computation phase: In the semi-honest setting, each party forwards its input to the trusted party. In the malicious settings, the corrupted party hands $\mathcal{F}_f$ the values handed to it by SIM. The trusted party computes $(y_0, y_1) = f(x_0, x_1)$ and hands each P_i the value y_i. SIM receives the output of the corrupted party.

Second corruption phase: SIM continues to another corruption phase, where it might choose to corrupt P_{i^*} for $i^* \in \{0,1\}$ (in case it did not corrupt any party in the first corruption phase), where this choice is made based on SIM's random tape and all the information gathered so far. Upon corrupting P_{i^*}, SIM learns its input x_{i^*}. ENV hands SIM some auxiliary information.

Output: The uncorrupted party P_{1-i^*} outputs y_{1-i^*} and the corrupted party outputs $\perp$. SIM outputs an arbitrary efficient function of its view. ENV learns all the outputs.

Post-execution corruption phase: After the outputs are generated, SIM proceeds with ENV in at most one round of interaction, where ENV can instruct SIM to corrupt P_{i^*} for $i^* \in \{0,1\}$ (if none of the parties are corrupted yet). SIM generates some arbitrary answer and might choose to corrupt P_{i^*}. The interaction continues until ENV halts with an output.

We denote by $\mathbf{OIDEAL}_{\mathcal{F}_f,\text{SIM},\text{ENV}}(n, x_0, x_1, z, \vec{r})$ the output of ENV on input z, random tape r_{ENV} and security parameter n upon interacting with SIM and parties P_0, P_1, running an ideal process with inputs r_{SIM} and x_0, x_1, respectively, where $\vec{r} = (r_{\text{ENV}}, r_{\text{SIM}})$. Let $\mathbf{OIDEAL}_{\mathcal{F}_f,\text{SIM},\text{ENV}}(n, x_0, x_1, z)$ denote a random variable describing $\mathbf{OIDEAL}_{\mathcal{F}_f,\text{SIM},\text{ENV}}(n, x_0, x_1, z, \vec{r})$ when the random tapes r_{ENV} and r_{SIM} are chosen uniformly. Let $\mathbf{OIDEAL}_{\mathcal{F}_f,\text{SIM},\text{ENV}}$ denote the distribution ensemble:

$$\{\mathbf{OIDEAL}_{\mathcal{F}_f,\text{SIM},\text{ENV}}(n, x_0, x_1, z)\}_{x_0,x_1,z\in\{0,1\}^*,n\in\mathbb{N}}$$

Then we define security as follows.

Definition 2.14 *Let $\mathcal{F}_f$ and Π_f be as defined above. Protocol Π_f UC realizes $\mathcal{F}_f$ in the presence of one-sided semi-honest/malicious adversaries if for every non-uniform probabilistic polynomial-time one-sided semi-honest/malicious adversary* ADV*, there exists a non-uniform probabilistic polynomial-time ideal adversary* SIM *such that:*

$$\begin{aligned}&\{\mathbf{OIDEAL}_{\mathcal{F}_f,\text{SIM},\text{ENV}}(n, x_0, x_1, z)\}_{x_0,x_1,z\in\{0,1\}^*,n\in\mathbb{N}}\\ \approx_c\ &\{\mathbf{OREAL}_{\Pi_f,\text{ADV},\text{ENV}}(n, x_0, x_1, z)\}_{x_0,x_1,z\in\{0,1\}^*,n\in\mathbb{N}}\end{aligned}$$

where $|x_0| = |x_1|$.

Composition. In order to simplify our security proofs we consider a hybrid setting where the parties implement some functionalities using ideals calls. We rely on the composition theorem introduced by Canetti [Can01] in the adaptive setting. (Note that we are only interested in cases where the same party is corrupted with respect to all composed protocols.)

2.6.2 Concrete Functionalities

We specify the definition of three important functionalities for this work.

Secure communication (SC). We define the functionality $\mathcal{F}_{\text{SC}} : (m, -) \mapsto (-, m)$ for securely communicating a message m from P_0 to P_1.

Oblivious transfer (OT). The 1-out-of-2 oblivious transfer functionality is defined by $\mathcal{F}_{\text{OT}} : ((x_0, x_1), \sigma)) \mapsto (-, x_\sigma)$. In a bit OT $x_0, x_1 \in \{0,1\}$, whereas in a string OT $x_0, x_1 \in \{0,1\}^n$.

Zero-knowledge proofs of knowledge (ZK PoK). Let $\mathcal{NP}$ relation $\mathcal{R}$ associated with the language $L_{\mathcal{R}} = \{x| \exists w \ s.t. \ (x,w) \in \mathcal{R}\}$. Then, we define the ZK PoK functionality for $\mathcal{R}$ by $\mathcal{F}^{\mathcal{R}}_{\text{ZKPoK}} : ((x,w),x) \mapsto (-,(x,b))$ where $b = \texttt{Accept}$ if $\mathcal{R}(x,w) = 1$ and $b = \texttt{Reject}$ if $\mathcal{R}(x,w) = 0$.

3 Different Notions of Non-Committing Encryptions (NCE)

3.1 NCE for the Receiver

An NCE for the receiver is a semantically secure PKE with an additional property that enables generating a secret key that decrypts a *simulated* (i.e., fake) ciphertext into any plaintext. Specifically, the scheme operates in two modes. The "real mode" enables to encrypt and decrypt as in the standard definition of PKE. The "simulated mode" enables to generate simulated ciphertexts that are computationally indistinguishable from real ciphertexts. Moreover, using a special trapdoor one can produce a secret key that decrypts a fake ciphertext into any plaintext. Intuitively, this implies that simulated ciphertexts are generated in a lossy mode where the plaintext is not well defined given the ciphertext and the public key. This leaves enough entropy for the secret key to be sampled in a way that determines the desired plaintext. We continue with a formal definition of NCE for the receiver.

Definition 3.1 (NCE for the receiver (NCER)) *An NCE for the receiver encryption scheme is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Dec}, \mathsf{Equivocate})$ *specified as follows:*

- $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ *are as specified in Definition 2.3.*
- Enc^**, given the public key* PK *output a ciphertext* c^* *and a trapdoor* t_{c^*}*.*
- $\mathsf{Equivocate}$*, given the secret key* SK*, trapdoor* t_{c^*} *and a plaintext* m*, output* SK^* *such that* $m \leftarrow \mathsf{Dec}_{\text{SK}^*}(c^*)$.

Definition 3.2 (Secure NCER) *An NCE for the receiver* $\Pi_{\text{NCR}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Enc}^*, \mathsf{Equivocate})$ *is* secure *if it satisfies the following properties:*

- $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ *imply an IND-CPA secure encryption scheme as in Definition 2.4.*
- *The following* ciphertext indistinguishability *holds for any plaintext* m*:* $(\text{PK}, \text{SK}^*, c^*, m)$ *and* $(\text{PK}, \text{SK}, c, m)$ *are computationally indistinguishable, for* $(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^n)$*,* $(c^*, t_{c^*}) \leftarrow \mathsf{Enc}^*(\text{PK})$*,* $\text{SK}^* \leftarrow \mathsf{Equivocate}(\text{SK}, c^*, t_{c^*}, m)$ *and* $c \leftarrow \mathsf{Enc}_{\text{PK}}(m)$.

We now review two implementations of NCER under the DDH and DCR assumptions.

NCER under the DDH assumption for polynomial-size message spaces. NCER for polynomial message space can be constructed under the DDH assumption [JL00, CHK05]. For simplicity we consider a binary plaintext space. Let $(g_0, g_0, p) \leftarrow \mathcal{G}(1^n)$ be an algorithm that given a security parameter n, returns a group description $\mathbb{G} = \mathbb{G}_{g_0,g_1,p}$ specified by its generators g_0, g_1 and its order p. Then define $\Pi^{\text{DH}}_{\text{NCER}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Dec}, \mathsf{Equivocate})$ as follows.

- Gen, given the security parameter n, set $(g_0, g_1, p) \leftarrow \mathcal{G}(1^n)$. Choose uniformly random $x, y \leftarrow \mathbb{Z}_p^2$ and compute $h = g_0^x g_1^y$. Output the secret key $\text{SK} = (x, y)$ and the public key $\text{PK} = (g_0, g_1, h)$.
- Enc, given the public key PK and a plaintext $m \in \mathbb{G}$, choose a uniformly random $r \leftarrow \mathbb{Z}_p^*$. Output the ciphertext $(g_0^r, g_1^r, g_0^m \cdot h^r)$.

– Enc^*, given the public key PK choose uniformly random $r_1, r_2, r_3 \leftarrow \mathbb{Z}_p^*$. Output the fake ciphertext $(g_0^{r_1}, g_0^{r_2}, g_0^{r_3})$ and trapdoor $t_{c^*} = (r_1, r_2, r_3)$.

– Dec, given the secret key (x, y) and a ciphertext (c_0, c_1, Φ), output $\Phi \cdot (c_0^x c_1^y)^{-1}$.

– $\mathsf{Equivocate}$, given (x, y), a simulated ciphertext c^*, trapdoor $t_{c^*} = (r_1, r_2, r_3)$ and a plaintext $m \in \mathbb{G}$ output $\mathrm{SK}^* = (x^*, y^*)$ by solving the system of linear equations induced by the exponents of the public key and ciphertext $c^* = (g_0^{r_1}, g_0^{r_2}, g_0^{r_3})$.[4]

Proposition 3.1 ([JL00, CHK05]) *Assume that the DDH assumption is hard in $\mathbb{G}$. Then $\Pi_{\mathrm{NCER}}^{\mathrm{DH}}$ is a secure NCER.*

It is easy to verify that real and simulated ciphertexts are computationally indistinguishable under the DDH assumption since the only difference is with respect to the first two group elements, the third group element induces a linear combination of the first two elements and the secret key in the exponent.

NCER under the DCR assumption for exponential-size message spaces. NCER for exponential message space can be constructed under the DCR assumption [CHK05]. Let $(p', q') \leftarrow \mathcal{G}(1^n)$ be an algorithm that given a security parameter n returns two random n bit primes p' and q' such that $p = 2p' + 1$ and $q = 2q' + 1$ are also primes. Let $N = pq$ and $N' = p'q'$. Define $\Pi_{\mathrm{NCER}}^{\mathrm{DCR}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Dec}, \mathsf{Equivocate})$ as follows.

– Gen, given the security parameter n, run $(p', q') \leftarrow \mathcal{G}(1^n)$ and set $p = 2p' + 1$, $q = 2q' + 1$, $N = pq$ and $N' = p'q'$. Choose random $x_0, x_1 \leftarrow \mathbb{Z}_{N^2/4}$ and a random $g' \in \mathbb{Z}_{N^2}^*$ and compute $g_0 = g'^{2N}$, $h_0 = g^{x_0}$ and $h_1 = g_0^{x_1}$. Output public key $\mathrm{PK} = (N, g_0, h_0, h_1)$ and secret key $\mathrm{SK} = (x_0, x_1)$.

– Enc, given the public key PK and a plaintext $m \in \mathbb{Z}_N$, choose a uniformly random $t \leftarrow \mathbb{Z}_{N/4}$ and output ciphertext

$$c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m; t) = \left(g_0^t \bmod N^2, (1+N)^m h_0^t \bmod N^2, h_1^t \bmod N^2\right).$$

– Enc^*, given the public key PK choose uniformly random $t \leftarrow \mathbb{Z}_{\phi(N)/4}$, compute the fake ciphertext

$$c^* \leftarrow (c_0^*, c_1^*, c_2^*) = \left((1+N) \cdot g_0^t \bmod N^2, (c_0^*)^{x_0} \bmod N^2, (c_0^*)^{x_1} \bmod N^2\right).$$

– Dec, given the secret key (x_0, x_1) and a ciphertext (c_0, c_1, c_2), check whether $c_0^{2x_1} = (c_2)^2$; if not output $\perp$. Then set $\hat{m} = (c_1/c_0^{x_0})^{N+1}$. If $\hat{m} = 1 + mN$ for some $m \in \mathbb{Z}_N$, then output m; else output $\perp$.

– $\mathsf{Equivocate}$, given N', (x_0, x_1), a ciphertext (c_0, c_1, c_2) and a plaintext $m \in \mathbb{Z}_N$, output $\mathrm{SK}^* = (x_0^*, x_1)$, where $x_0^* \leftarrow \mathbb{Z}_{NN'}$ is the unique solution to the equations $x_0^* = x \bmod N'$ and $x_0^* = x_0 - m \bmod N$. These equations have a unique solution due to the fact that $gcd(N, N') = 1$ and the solution can be obtained employing Chinese Remainder Theorem.

  It can be verified that the secret key SK^* matches the public key PK and also decrypts the 'simulated' ciphertext to the required message m. The first and third component of PK remains the same since

[4] In order to compute such a secret key the $\mathsf{Equivocate}$ algorithm has to know $\log_{g_0} g_1$ and $\log_{g_0} m$. The requirement of knowing $\log_{g_0} m$ makes this scheme work only for messages from polynomial-size spaces.

x_1 has not been changed. Now $g^{x_0^*} = g^{x_0^* \bmod N'} = g^{x_0 \bmod N'} = g^{x_0} = h_0$. Using the fact that the order of $(1+N)$ in $\mathbb{Z}^*_{N^2}$ is N, we have

$$\left(\frac{c_1}{c_0^{x_0^*}}\right)^{N+1} = \left(\frac{(1+N)^{x_0} g_0^{tx_0}}{(1+N)^{x_0^*} g_0^{tx_0^*}}\right)^{N+1}$$
$$= \left((1+N)^{x_0 - x_0^* \bmod N}\right)^{N+1} = (1+N)^m = (1+mN).$$

Proposition 3.2 ([CHK05]) *Assume that the DCR assumption is hard in $\mathbb{Z}^*_{N^2}$. Then $\Pi^{\mathrm{DCR}}_{\mathrm{NCR}}$ is a secure NCER.*

It is easy to verify that real and simulated ciphertexts are computationally indistinguishable under the DCR assumption since the only difference is with respect to the first element (which is an $2N$th power in a real ciphertext and not an $2N$th power in a simulated ciphertext). The other two elements are powers of the first element. Furthermore $\mathrm{SK} = (x_0, x_1)$ and $\mathrm{SK}^* = (x_0^*, x_1)$ are statistically close since $x_0 \leftarrow \mathbb{Z}_{N^2/4}$ and $x_0^* \leftarrow \mathbb{Z}_{NN'}$ and the uniform distribution over $\mathbb{Z}_{NN'}$ and $\mathbb{Z}_{N^2/4}$ is statistically close.

3.2 NCE for the Sender

NCE for the sender is a semantically secure PKE with an additional property that enables generating a fake public key, such that any ciphertext encrypted under this key can be viewed as the encryption of any message together with the matched randomness. Specifically, the scheme operates in two modes. The "real mode" that enables to encrypt and decrypt as in standard PKEs and the "simulated mode" that enables to generate simulated public keys and an additional trapdoor, such that the two modes keys are computationally indistinguishable. In addition, given this trapdoor and a ciphertext generated using the simulated public key, one can produce randomness that is consistent with any plaintext. We continue with a formal definition.

Definition 3.3 (NCE for the sender (NCES)) *An NCE for the sender encryption scheme is a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Gen}^*, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Equivocate})$ *specified as follows:*

- $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ *are as specified in Definition 2.3.*
- Gen^* *generates public key* PK^* *and a trapdoor* t_{PK^*}.
- $\mathsf{Equivocate}$*, given a ciphertext c^* computed using* PK^**, a trapdoor t_{PK^*} and a plaintext m, output r such that $c^* \leftarrow \mathsf{Enc}(m; r)$.*

Definition 3.4 (Secure NCES) *An NCE for the sender* $\Pi_{\mathrm{NCES}} = (\mathsf{Gen}, \mathsf{Gen}^*, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Equivocate})$ *is* secure *if it satisfies the following properties:*

- $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$ *imply an IND-CPA secure encryption scheme as in Definition 2.4.*
- *The following* public key indistinguishability *holds for any plaintext m:* $(\mathrm{PK}^*, r^*, m, c^*)$ *and* (PK, r, m, c) *are computationally indistinguishable, for* $(\mathrm{PK}^*, t_{\mathrm{PK}^*}) \leftarrow \mathsf{Gen}^*(1^n)$*,* $c^* \leftarrow \mathsf{Enc}_{\mathrm{PK}^*}(m'; r')$*,* $r^* \leftarrow \mathsf{Equivocate}(c^*, t_{\mathrm{PK}^*}, m)$ *and* $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m; r)$.

We review the DDH based implementation from [BHY09] and then present our DCR based implementation.

NCES under the DDH assumption for polynomial-size message spaces. For simplicity we consider a binary plaintext space. Let $(g_0, g_1, p) \leftarrow \mathcal{G}(1^n)$ be an algorithm that given a security parameter n returns a group description $\mathbb{G} = \mathbb{G}_{g_0,g_1,p}$ specified by its generators g_0, g_1 and its order p. Then define $\Pi^{\mathrm{DH}}_{\mathrm{NCES}} = (\mathsf{Gen}, \mathsf{Gen}^*, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Equivocate})$ as follows.

- Gen, given the security parameter n, set $(g_0, g_1, p) \leftarrow \mathcal{G}(1^n)$. Choose uniformly random $x \leftarrow \mathbb{Z}_p$ and compute $h_i = g_i^x$ for all $i \in \{0, 1\}$. Output the secret key $\mathrm{SK} = x$ and the public key $\mathrm{PK} = (g_0, g_1, h_0, h_1)$.
- Gen^*, given the security parameter n, set $(g_0, g_1, p) \leftarrow \mathcal{G}(1^n)$. Choose uniformly random $x_0, x_1 \leftarrow \mathbb{Z}_p^2$ and $h_i = g_i^{x_i}$ for all $i \in \{0, 1\}$. Output the trapdoor $t_{\mathrm{PK}^*} = (x_0, x_1)$ and the public key $\mathrm{PK}^* = (g_0, g_1, h_0, h_1)$.
- Enc, given the public key PK (or PK^*) and a plaintext $m \in \mathbb{G}$, choose a uniformly random $s, t \leftarrow \mathbb{Z}_p$. Output the ciphertext $\left(g_0^s g_1^t, g_0^m \cdot (h_0^s h_1^t)\right)$.
- Dec, given the secret key x and a ciphertext (g_c, h_c), output $h_c \cdot (g_c^x)^{-1}$.
- $\mathsf{Equivocate}$, given the fake key $\mathrm{PK}^* = (g_0, g_1, h_0, h_1)$, trapdoor t_{PK^*}, a ciphertext $c^* = \mathsf{Enc}_{\mathrm{PK}^*}(m; (s', t'))$ and a plaintext m, output (s, t) such that $c^* \leftarrow \mathsf{Enc}(m; (s, t))$ by solving the system of linear equations induced by the exponents of the two elements in ciphertext.[5]

Proposition 3.3 *Assume that the DDH assumption is hard in $\mathbb{G}$. Then $\Pi^{\mathrm{DH}}_{\mathrm{NCES}}$ is a secure NCES.*

Proof: It is easy to verify that real and simulated public keys are computationally indistinguishable under the DDH assumption even in the presence of the randomness of the encryption. Proving indistinguishability of a real and simulated public key in the presence of the randomness used for encryption is sufficient since a ciphertext is a linear combination of the public key using the randomness for encryption both in the real as well as in the simulated world. The proof follows by a reduction to the DDH assumption as follows. Given a tuple (g_0, g_1, h_0, h_1) adversary $\mathrm{ADV}_{\mathrm{DH}}$ that attempts to decide whether the given tuple is a DH tuple or not, fixes this tuple as the public key, encrypts a message m under this key and output the public key, ciphertext, m and the randomness it used to generate the ciphertext. It then invokes the adversary $\mathrm{ADV}^{\mathrm{DH}}_{\mathrm{NCES}}$ for our scheme that can tell apart (with non-negligible probability) a simulated public key from a valid public key. $\mathrm{ADV}_{\mathrm{DH}}$ outputs what $\mathrm{ADV}^{\mathrm{DH}}_{\mathrm{NCES}}$ outputs. Clearly, $\mathrm{ADV}_{\mathrm{DH}}$ can decide if a given tuple is a DH tuple. ∎

NCES under the DCR assumption for exponential-size message spaces. In what follows, we introduce a new NCE for the sender based on the security of the DCR assumption. This allows us to fix the composite as part of the CRS when appropriate. Our scheme is based on the PKE from [CHK05], building on earlier work by Cramer and Shoup [CS02]. Let $N = pq$ be an RSA modulus, then define $\Pi^{\mathrm{DCR}}_{\mathrm{NCES}} = (\mathsf{Gen}, \mathsf{Gen}^*, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Equivocate})$ an NCES as follows.

- Gen, given the security parameter n, generate an RSA modulus $N = pq$ with $p = 2p' + 1$ and $q = 2q' + 1$ where p, q, p', q' are primes. Pick $g' \leftarrow \mathbb{Z}^*_{N^2}$ and $\alpha \leftarrow \mathbb{Z}_{N^2/4}$ and set $g_0 = g'^{2N} \bmod N^2$ and $h_0 = g_0^\alpha \bmod N^2$. Choose a random $r \leftarrow \mathbb{Z}_{N/4}$ and compute $g_1 = g_0^r \bmod N^2, h_1 = ((1 + N) \cdot h_0^r) \bmod N^2$. Output $\mathrm{PK} = (N, g_0, h_0, g_1, h_1)$ and secret key $\mathrm{SK} = \alpha$.
- Gen^*, given the security parameter n, generate N, g_0, h_0 as in Gen. Choose a random $r \leftarrow \mathbb{Z}_{N/4}$ and compute $g_1 = g_0^r \bmod N^2, h_1 = h_0^r \bmod N^2$. Output $\mathrm{PK}^* = (N, g_0, h_0, g_1, h_1)$ and trapdoor $t_{\mathrm{PK}^*} = r$.

[5] In order to compute such randomness the $\mathsf{Equivocate}$ algorithm has to know $\log_{g_0} g_1$ and $\log_{g_0} m$. The requirement of knowing $\log_{g_0} m$ makes this scheme work only for messages from polynomial-size spaces.

– Enc, given the public key $\mathrm{PK} = (N, g_0, h_0, g_1, h_1)$ (or PK^*) and a message $m \in \mathbb{Z}_N$, choose a random $t \leftarrow \mathbb{Z}_{N/4}$ and output the ciphertext

$$c \leftarrow \mathsf{Enc}(m;t) = \left((g_1^m g_0^t) \bmod N^2, (h_1^m h_0^t) \bmod N^2\right).$$

– Dec, given the public key $\mathrm{PK} = (N, g_0, h_0, g_1, h_1)$, secret key $\mathrm{SK} = \alpha$ and ciphertext $c = (g_c, h_c)$, compute $\hat{m}$ as follows and output $m \in \mathbb{Z}_N$ such that $\hat{m} = 1 + mN$.

$$\hat{m} = (h_c / g_c^{\alpha})^{N+1} = [(1+N)^m]^{N+1} = (1+N)^m.$$

– Equivocate, given $\Phi(N)$, the fake key $\mathrm{PK}^* = (N, g_0, h_0, g_1, h_1)$, trapdoor $t_{\mathrm{PK}^*} = r$, a ciphertext $c^* \leftarrow \mathsf{Enc}_{\mathrm{PK}^*}(m;t) = (g_c, h_c)$ and a message m', output $t' = (rm + t - rm') \bmod \Phi(N)/4$. It is easy to see that

$$\begin{aligned}\mathsf{Enc}_{\mathrm{PK}^*}(m';t') &= \left((g_1^{m'} g_0^{t'}), h_1^{m'} h_0^{t'})\right) \\ &= \left((g_0^{rm'} g_0^{(rm+t-rm')}), (h_0^{rm'} h_0^{(rm+t-rm')})\right) = c.\end{aligned}$$

Next, we show that this scheme meets Definitions 3.4.

Proposition 3.4 *Assume that the DCR assumption is hard in $\mathbb{Z}_{N^2}^*$. Then $\Pi_{\mathrm{NCES}}^{\mathrm{DCR}}$ is a secure NCES.*

Proof: Given the public key $\mathrm{PK} = (N, g_0, h_0, g_1, h_1)$ and two messages $m, m' \in \mathbb{Z}_N$, we show that encryptions of m and m' are computationally close. Namely tuples, (1) $\left((g_1^m g_0^t) \bmod N^2, (h_1^m h_0^t) \bmod N^2\right)$ and (2) $\left((g_1^{m'} g_0^t) \bmod N^2, (h_1^{m'} h_0^t) \bmod N^2\right)$ are computationally close. This follows immediately from the IND-CPA security proof for the modified scheme in [DJ03] (cf. Theorem 2 of [DJ03]).

The fact that fake and valid public keys are computationally indistinguishable follows from the IND-CPA security of [CHK05] and [CS02] and the former proof (for the DDH based scheme). As the former key is an encryption of zero whereas the latter key is an encryption of one. ■

3.3 Somewhat Non-Committing Encryption [GWZ09]

The idea of somewhat NCE is to exploit the fact that it is often unnecessary for the simulator to explain a fake ciphertext for *any* plaintext. Instead, in many scenarios it suffices to explain a fake ciphertext with respect to a small set of size ℓ determined in advance (where ℓ might be as small as 2). Therefore there are two parameters that are considered here: a plaintext of bit length l and an equivocality parameter ℓ which is the number of plaintexts that the simulator needs to explain a ciphertext for (namely, the non-committed domain size). Note that for fully NCE $\ell = 2^l$. Somewhat NCE typically improves over fully NCE whenever ℓ is very small but the plaintext length is still large, say $O(n)$ for n the security parameter.

[GWZ09] design somewhat NCE using three primitives: simulatable PKE (cf. Definition 2.2), NCE (for the purpose of sending a short index of length $\log \ell$), and a secret key encryption (SKE). Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Gen}}^{-1}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Enc}}^{-1})$ be a simulatable PKE and $(\mathsf{Gen}^{\mathrm{SKE}}, \mathsf{Enc}^{\mathrm{SKE}}, \mathsf{Dec}^{\mathrm{SKE}})$ be an SKE in which the ciphertexts are indistinguishable from uniformly random strings of the same length. In more details,

Setup Phase.

- SEN sends a random index $i \in [\ell]$ using NCE.
- SEN generates ℓ public keys. For $j \in \{1, \ldots, \ell\} \setminus \{i\}$, the keys $\mathrm{PK}_j \leftarrow \widetilde{\mathsf{Gen}}(1^n)$ are generated obliviously while $(\mathrm{PK}_i, \mathrm{SK}_i) \leftarrow \mathsf{Gen}(1^n)$. SEN sends the keys $\mathrm{PK}_1, \ldots, \mathrm{PK}_\ell$ and stores SK_i.

- REC generates $K \leftarrow \mathsf{Gen}^{\mathrm{SKE}}(1^n)$ and computes $c_i \leftarrow \mathsf{Enc}_{\mathsf{PK}_i}(K)$. REC also generates $c_j \leftarrow \widetilde{\mathsf{Enc}}_{\mathsf{PK}_i}(1^n)$ obliviously for $j \in \{1,\ldots,\ell\} \setminus \{i\}$. Finally it sends ciphertexts $c_1,\ldots,c_\ell$.
- SEN decrypts the key $K \leftarrow \mathsf{Dec}_{\mathsf{SK}_i}(c_i)$. Both parties store K, i.

Transfer Phase.

- SEN computes $C_i \leftarrow \mathsf{Enc}_K^{\mathrm{SKE}}(m)$ and chooses C_j for $j \in \{1,\ldots,\ell\} \setminus \{i\}$ uniformly at random of the appropriate length. S sends $(C_1,\ldots,C_\ell)$.
- REC ignores everything other than C_i and decrypts $m \leftarrow \mathsf{Dec}_K^{\mathrm{SKE}}(C_i)$.

In case of no corruptions the simulator simulates both parties as follows: it first generates the keys and the ciphertexts in the setup phase using Gen and Enc, and generates ℓ keys for a symmetric key encryption. In the transfer phase the simulator uses these symmetric keys to encrypt all ℓ plaintexts. Assume that the adversary corrupts a party at the end of this phase then the simulator obtains the message m, say the kth element in $[\ell]$. It then explains the index of this element via the NCE and uses the key and ciphertext faking algorithms to explain the other public key/ciphertext pairs as being obliviously generated. Given that the kth simulatable PKE ciphertext encrypts the secret key K, it holds that the kth SKE ciphertext encrypts the message m. This simulation strategy works for corruption at any point of the execution.

4 One-sided Adaptively Secure NCE

In this section we design one-sided NCE, building on NCE for the sender (NCES) and NCE for the receiver (NCER). Namely, NCER implies that for any plaintext there exists an efficiently generated secret key that decrypts a fake ciphertext into that plaintext (see Definition 3.1). Furthermore, NCES implies that for any plaintext there exists efficiently generated randomness for proving that a ciphertext, encrypted under a fake key, encrypts that plaintext (see Definition 3.3).

The idea of our protocol is to have the receiver create two public/secret key pairs where the first pair is for NCES and the second pair is for NCER. The receiver sends the public keys and the sender encrypts two shares of its message m, each share with a different key. Upon receiving the ciphertexts the receiver recovers the message by decrypting the ciphertexts. Therefore, equivocality of the sender's input can be achieved if the public key of the NCES is fake, whereas, equivocality of the receiver's input can be achieved if the ciphertext of the NCER is fake. Nevertheless, this idea only works if the simulator is aware of the identity of the corrupted party prior to the protocol execution in order to decide whether the keys or the ciphertexts should be explained as valid upon corruption (since it cannot explain fake keys/ciphertext as valid). We resolve this problem using somewhat NCE in order to commit to *the choice* of having fake/valid keys and ciphertexts. Specifically, it enables the simulator to "explain" fake keys/ciphertext as valid and vice versa using only a constant number of asymmetric operations, as each such non-committing bit requires an equivocation space of size 2. (An overview of somewhat NCE is given in Section 3.3.)

Let $\Pi_{\mathrm{NCES}} = (\mathsf{Gen}, \mathsf{Gen}^*, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Equivocate})$ and $\Pi_{\mathrm{NCER}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Dec}, \mathsf{Equivocate})$ denote secure NCES and NCER for a message space $\{0,1\}^q$. Consider the following one-sided protocol to realize the message communication functionality $\mathcal{F}_{\mathrm{SC}} : (m, -) \mapsto (-, m)$.

Protocol 1 (One-sided NCE (Π_{OSC}))

- **Inputs:** *Sender* SEN *is given input message* $m \in \{0,1\}^q$. *Both parties are given security parameter* 1^n.
- **The Protocol:**

  1. **Message from the receiver.** REC *invokes* $\mathsf{Gen}(1^n)$ *of* Π_{NCES} *and* Π_{NCER} *and obtains two public/secret key pairs* $(\mathrm{PK}_0, \mathrm{SK}_0)$ *and* $(\mathrm{PK}_1, \mathrm{SK}_1)$*, respectively.* REC *sends* PK_1 *on clear and* PK_0 *using somewhat NCE with equivocality parameter* $\ell = 2$.

2. **Message from the sender.** *Upon receiving* PK_0 *and* PK_1, SEN *creates two shares of* m, m_0 *and* m_1, *such that* $m = m_0 \oplus m_1$. *It then encrypts each* m_i *using* PK_i, *creating ciphertext* c_i, *and sends* c_0 *and* c_1 *using two instances of somewhat NCE with equivocality parameter* $\ell = 2$.
3. **Output.** *Upon receiving* c_0, c_1, REC *decrypts* c_i *using* SK_i *and outputs the bitwise XOR of the decrypted plaintexts.*

Note that the message space of our one-sided NCE is equivalent to the message space of the NCES/NCER schemes, where q can be as large as n. Therefore, our protocol transmits q-bits messages using a *constant number of PKE operations*, as opposed to fully adaptive NCEs that require $O(q)$ such operations. We provide two instantiations for the above protocol. One for polynomial-size message spaces using DDH based NCES and NCER, and another for exponential-size message spaces using DCR based NCES and NCER. We conclude with the following theorem and the complete proof.

Theorem 4.1 *Assume the existence of NCER and NCES with constant number of PKE operations for message space* $\{0,1\}^q$ *and simulatable PKE. Then Protocol 1 UC realizes* $\mathcal{F}_{\mathrm{SC}}$ *in the presence of one-sided adaptive malicious adversaries with constant number of PKE operations for message space* $\{0,1\}^q$, *where* $q = O(n)$ *and* n *is the security parameter.*

Intuitively, security follows due to the fact that the simulator is not committed to either valid keys or valid ciphertexts. Thus, upon corrupting one of the parties it is able to explain that party's internal state, while equivocating the communication and make it consistent with message m. For instance, if the sender is corrupted after the simulator sends the message on the sender's behalf, the simulator can explain PK_0 as a fake key. Thus, the ciphertext encrypted under this key can be explained with respect to any plaintext. Details follow.

Proof: Let ADV be a malicious probabilistic polynomial-time adversary attacking Protocol 1 by adaptively corrupting one of the parties. We construct an adversary SIM for the ideal functionality $\mathcal{F}_{\mathrm{SC}}$ such that no environment ENV distinguishes with a non-negligible probability whether it is interacting with ADV in the real setting or with SIM in the ideal setting. We recall that SIM interacts with the ideal functionality $\mathcal{F}_{\mathrm{SC}}$ and the environment ENV. We refer to the interaction of SIM with $\mathcal{F}_{\mathrm{SC}}$ and ENV as the external interaction. The interaction of SIM with the simulated ADV is the internal interaction. We explain the strategy of the simulation for all corruption cases.

Simulating the communication with ENV. Every input value received by the simulator from ENV is written on ADV's input tape. Likewise, every output value written by ADV on its output tape is copied to the simulator's output tape (to be read by the environment ENV).

SEN is corrupted at the onset of the protocol. SIM begins by activating ADV and emulates the honest receiver by sending ADV, PK_0 using the somewhat NCE and PK_1 in clear as the honest receiver would do. Upon receiving two ciphertexts c_0 and c_1 from ADV, SIM extracts m by computing $\mathsf{Dec}_{\mathrm{SK}_0}(c_0) \oplus \mathsf{Dec}_{\mathrm{SK}_1}(c_1)$. SIM externally forwards m to the ideal functionality $\mathcal{F}_{\mathrm{SC}}$. Since corruption takes place at the onset of the protocol execution the simulator is able to perfectly emulate the sender's view.

REC is corrupted at the onset of the protocol. SIM begins by activating ADV and obtains REC's output m from $\mathcal{F}_{\mathrm{SC}}$. SIM invokes ADV and receives PK_0 from ADV via the somewhat NCE and PK_1 in clear. Next, SIM completes the execution playing the role of the honest sender on input m. Note that it does not make a difference whether REC generates valid or invalid public keys since SIM knows m and thus perfectly emulates the receiver's view.

If none of the parties is corrupted as above, SIM emulates the receiver's message as follows. It creates public/secret key pair $(\mathrm{PK}_1, \mathrm{SK}_1)$ for Π_{NCER} and sends the public key in clear. It then creates a valid public/secret key pair $(\mathrm{PK}_0, \mathrm{SK}_0)$ and a fake public key with a trapdoor $(\mathrm{PK}_0^*, t_{\mathrm{PK}_0^*})$ for Π_{NCES} (using Gen and Gen^*, respectively). SIM sends $(\mathrm{PK}_0, \mathrm{PK}_0^*)$ using somewhat NCE. Namely, the simulator does not send the valid PK_0 as the honest receiver would do, rather it encrypts both valid and invalid keys within the somewhat NCE.

SEN is corrupted between Steps 1 and 2. Since no message was sent yet on behalf of the sender, SIM completes the simulation exactly as it does in the previous case when SEN was corrupted at the outset.

REC is corrupted between Steps 1 and 2. Upon receiving m, SIM explains the receiver's internal state which is independent of the message m so far. Specifically, it reveals the randomness for generating $\mathrm{PK}_0, \mathrm{SK}_0$ and $\mathrm{PK}_1, \mathrm{SK}_1$ and presents the randomness for the valid key PK_0 being the message sent by the somewhat NCE. SIM plays the role of the honest sender with input m as the message.

In these cases the only difference between the real and simulated executions is with respect to the somewhat NCE channel that delivers the public key of NCES. Specifically, in the real execution this channel always delivers a valid public key and the rest of the computation is computed honestly, whereas in the simulated execution it delivers both valid and fake keys (for which later it is decided which key will be used). Therefore the difference between the real and simulated views is reduced to the difference between two invocations of somewhat NCE channel and security follows from the security of this channel. Using the construction from [GWZ09], security is reduced to the security of the underlying simulatable PKE and SKE (with ciphertexts that are indistinguishable from uniformly random strings). The proof is straightforward and thus omitted.

If none of the above corruption cases occur, SIM emulates the sender's message as follows. It first chooses two random shares m_0', m_1' and generates a pair of ciphertexts (c_0', c_0^*) for Π_{NCES} that encrypts m_0' using PK_0 and PK_0^*. It then generates a pair of ciphertexts (c_1', c_1^*) for Π_{NCER} such that c_1' is a valid encryption of m_1' using the public key PK_1, and c_1^* is a fake ciphertext generated using Enc^* and PK_1. SIM sends (c_0', c_0^*) and (c_1^*, c_1') via two instances of somewhat NCE.

SEN is corrupted after Step 2 is concluded. Upon receiving a corruption instruction from ENV, SIM corrupts the ideal SEN and obtains SEN's input m. It then explains the sender's internal state as follows. It explains PK_0^* for being the public key sent by the receiver using the somewhat NCE. Furthermore, it presents the randomness for c_0^* and c_1' being the ciphertexts sent via the somewhat NCE. Finally, it computes $r'' \leftarrow \mathsf{Equivocate}(c_0^*, t_{\mathrm{PK}_0^*}, m_0'')$ for m_0'' such that $m = m_0'' \oplus m_1'$, and presents r'' as the randomness used to generate c_0^* that encrypts m_0''. The randomness used for generating c_1' is revealed honestly.

Consider the adversary's view which is comprised from three invocations of somewhat non-committing channel and a public key PK_1 that was honestly generated. The message received from the receiver via the first channel is a fake public key PK_0^*. Moreover, the remaining two invocations of the somewhat NCE are regarding sending the ciphertexts encrypted under keys PK_0^* and PK_1. We can reduce the security of this case to the security of Π_{NCES} and somewhat NCE.

Let $\mathcal{H}_0$ denote the simulation. Then, define a sequence of three hybrid games $\{\mathcal{H}_i\}_{i=1}^3$ where in $\mathcal{H}_i$ simulator SIM_i invokes the first i executions of the somewhat NCE channel as in the real execution, whereas the remaining $3 - i$ invocations are computed as in the simulation. Specifically, in the first hybrid game the simulator SIM_1 sends a valid public key PK_0 using the first somewhat NCE channel and two pairs of ciphertexts $(c_0', c_0^*), (c_1', c_1^*)$ using the last two executions of the somewhat NCE. Moreover, the only difference in the second hybrid game is that the simulator SIM_2 sends c_0 encrypted

under PK_0 within the second execution of the somewhat NCE. Finally, in the third hybrid game the difference is that the simulator SIM_3 sends c_1 using the somewhat NCE channel. The messages encrypted in c_0 and c_1 when xored result in the real message m.

Note that the difference with respect to the adversary's view in games $\mathcal{H}_0$ and $\mathcal{H}_1$ is due to the messages sent within the somewhat NCE channel. That is, in game $\mathcal{H}_0$ the simulator sends a pair of keys $(\mathrm{PK}_0, \mathrm{PK}_0^*)$ such that one is valid and the other one is fake, through this channel and reveals the latter key, whereas in game $\mathcal{H}_1$ the simulator sends just a valid key PK_0. Security follows due to the security of the somewhat NCE channel and the indistinguishability property of Π_{NCES}. To prove that, we consider an intermediate game $\mathcal{H}'$ for which the simulator SIM' sends a pair of valid keys $(\mathrm{PK}_0, \mathrm{PK}_0')$ rather than $(\mathrm{PK}_0, \mathrm{PK}_0^*)$ (as done in game $\mathcal{H}_0$). We first show that the adversary's view within this game is indistinguishable from its view in game $\mathcal{H}_0$. Here the difference boils down to the key indistinguishability property of Π_{NCES}. Given m and a tuple of one of these forms $(\mathrm{PK}^*, r^*, \tilde{m}, c^*)$ or $(\mathrm{PK}, r, \tilde{m}, c)$ for an arbitrary $\tilde{m}$, as specified in Definition 3.4, a distinguisher $\mathbf{Adv}_{\text{NCES}}$ continues as follows. Say it was given $(\mathrm{PK}', r', \tilde{m}, c')$, then $\mathbf{Adv}_{\text{NCES}}$ encrypts PK' together with a valid key PK_0 using somewhat NCE and sends a valid key PK_1 in the clear. It then sends, using somewhat NCE, c' together with another ciphertext c_0 that encrypts $\tilde{m}$ under PK_0. Finally, it sends, using somewhat NCE, a valid and a fake ciphertext (c_1', c_1^*) that encrypt $m \oplus \tilde{m}$ under PK_1. The distinguisher then explains the adversary's internal state as in game $\mathcal{H}_0$ explaining that PK_0' has been received while using r' to explain the randomness used for computing c'. Clearly, any advantage in distinguishing the views generated in games $\mathcal{H}_0$ and $\mathcal{H}'$ can be reduced to breaking the indistinguishability of Π_{NCES} since $\mathbf{Adv}_{\text{NCES}}$ generates a view that is distributed according to one of these games.

Next, we prove indistinguishability relative to games $\mathcal{H}'$ and $\mathcal{H}_1$. Here the difference boils down to the difference relative to the unopened (symmetric key) ciphertexts within the somewhat NCE channel (where the adversary plays the role of the receiver). This is because the adversary receives in both executions a valid key, yet in game $\mathcal{H}'$ the simulator sends two valid keys $\mathrm{PK}_0, \mathrm{PK}_0'$ while in the latter game the simulator sends just one valid key over the somewhat NCE channel honestly. Security here follows immediately due to the security of the channel. A similar argument holds for the indistinguishability between games $\mathcal{H}_1$ and $\mathcal{H}_2$, and games $\mathcal{H}_2$ and $\mathcal{H}_3$ (where here the adversary is the sender via the somewhat NCE channel). This concludes the proof since game $\mathcal{H}_3$ is identical to the real execution.

REC is corrupted after Step 2 is concluded. Upon receiving a corruption instruction from ENV, SIM corrupts the ideal REC and obtains REC's output m. It then explains the receiver's internal state as follows. It presents the randomness for PK_0 for being the public key sent via the somewhat NCE and presents the randomness for generating $(\mathrm{PK}_0, \mathrm{SK}_0)$. It then explains c_0 and c_1^* for being received via the somewhat NCE. Finally, it generates a secret key SK_1^* so that $m_1'' \leftarrow \mathsf{Dec}_{\mathrm{SK}_1^*}(c_1^*)$ and $m_1'' \oplus m_0' = m$. That is, it explains $(\mathrm{PK}_1, \mathrm{SK}_1^*)$ as the other pair of keys generated by the receiver.

Security is proven similarly to the case the sender is corrupted at the end. Namely, the receiver's view is comprised from three invocations of the somewhat NCE channel and ciphertexts c_0, c_1^*. Security follows due to the security of somewhat NCE and Π_{NCER}, where a distinguisher $\mathbf{Adv}_{\text{NCER}}$ that wishes to distinguish between a fake and a real ciphertext receives either $(\mathrm{PK}, \mathrm{SK}_1^*, c_1^*, m_1'')$ or $(\mathrm{PK}, \mathrm{SK}, c_1, m_1'')$.

■

5 One-Sided Adaptively Secure OT

$\mathcal{F}_{\text{OT}} : ((x_0, x_1), \sigma) \mapsto (-, x_b)$ is one of the fundamental functionalities in secure computation. A common approach to design an adaptive OT [Bea98, CLOS02] is by having the receiver generate two public keys $(\text{PK}_0, \text{PK}_1)$ such that it only knows the secret key associated with PK_σ. The sender then encrypts x_0, x_1 under these respective keys so that the receiver decrypts the σth ciphertext. The security of this protocol in the adaptive setting holds if the underlying encryption scheme is an *augmented non-committing encryption scheme* [CLOS02]. In this section we follow the approach from [GWZ09] and build one-sided OT based on the static OT from [PVW08], which is based on a primitive called *dual-mode PKE*.

The [PVW08] OT. Dual-mode PKE is an IND-CPA secure encryption scheme that is initialized with system parameters of two types. For each type one can generate two types of public/secret key pair, labeled by the left key pair and the right key pair. Similarly, the encryption algorithm generates a left or a right ciphertext. Moreover, if the key label matches the ciphertext label (i.e., a left ciphertext is generated under the left public key), then the ciphertext can be correctly decrypted. (A formal definition of dual-mode PKE is given in Section 2.3.) This primitive was introduced in [PVW08] which demonstrates its usefulness in designing efficient statically secure OTs under various assumptions. First, the receiver generates a left key if $\sigma = 0$, and a right key otherwise. In response, the sender generates a left ciphertext for x_0 and a right ciphertext for x_1. The receiver then decrypts the σth ciphertext.

The security of dual-mode PKE relies on the two indistinguishable modes of generating the system parameters: *messy* and *decryption* mode. In a messy mode the system parameters are generated together with a messy trapdoor. Using this trapdoor, any public key (even malformed ones) can be labeled as a left or as a right key. Moreover, when the key type does not match the ciphertext type, the ciphertext becomes statistically independent of the plaintext. The messy mode is used to ensure security when the receiver is corrupted since it allows to extract the receiver's input bit while hiding the sender's other input. On the other hand, the system parameters in a decryption mode are generated together with a decryption trapdoor that can be used to decrypt both left and right ciphertexts. Moreover, left public keys are statistically indistinguishable from right keys. The decryption mode is used to ensure security when the sender is corrupted since the decryption trapdoor enables to extract the sender's inputs while statistically hiding the receiver's input. [PVW08] instantiated dual-mode PKE and their generic OT construction based on various assumptions, such as DDH, QR and lattice-based assumptions. We recall their instantiation based on DDH in Appendix A. In a followup work [GWZ09], Garay et al. extended the dual-mode definition into enhanced dual-model PKE in order to enable equivocation of the sender's input relative to the ciphertext that encrypts $x_{1-\sigma}$ in a messy mode. Our protocol implies equivocation for the sender via our one-sided NCE which implies that we can use the simpler dual-mode primitive.

Our construction. We build our one-sided OT based on the PVW protocol considering the following modifications. (**1**) First, we require that the sender sends its ciphertexts using one-sided NCE (see Section 4). (**2**) We fix the system parameters in a decryption mode, which immediately implies extractability of the sender's input and equivocality of the receiver's input. We further achieve equivocality of the sender's input using our one-sided NCE. In order to ensure extractability of the receiver's input we employ a special type of ZK PoK. Namely, this proof exploits the fact that the simulator knows both witnesses for the proof yet it does not know which witness will be used by the real receiver, since this choice depends on σ. Specifically, it allows the simulator to use both witnesses and later convince the adversary that it indeed used a particular witness. In addition, it enables to extract σ since the real receiver does not know both witnesses. We denote these proofs for compound statements by witness equivocal and refer to Section 7.2 for more details.

Our construction is one-sided UC secure in the presence of malicious adversaries, and uses a number of

non-committed bits that is *independent* of the sender's input size or the overall communication complexity. We formally denote the dual-mode PKE of [PVW08] by $\Pi_{\text{DUAL}} = (\mathsf{SetupMessy}, \mathsf{SetupDecryption}, \mathsf{dGen}, \mathsf{dEnc}, \mathsf{dDec}, \mathsf{FindBranch}, \mathsf{TrapKeyGen})$ and describe our construction in the $(\mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}})$-hybrid model, where $\mathcal{F}_{\text{SC}}$ is instantiated with one-sided NCE. Furthermore, the latter functionality is required to ensure the correctness of the public key and is defined for a compound statement that is comprised from the following two relations,

$$\mathcal{R}_{\text{LEFT}} = \big\{(\text{PK}, r_0) \mid (\text{PK}, \text{SK}) \leftarrow \mathsf{dGen}(\text{CRS}, 0; r_0)\big\},$$

where CRS are the system parameters. Similarly, we define $\mathcal{R}_{\text{RIGHT}}$ for the right keys. Specifically, $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}}$ receives a public key PK and randomness r_σ for $\sigma \in \{0,1\}$ and returns `Accept` if either $\sigma = 0$ and $\text{PK} = \mathsf{dGen}(\text{CRS}, 0; r_0)$, or $\sigma = 1$ and $\text{PK} = \mathsf{dGen}(\text{CRS}, 1; r_1)$ holds. Security is proven by implementing this functionality using a witness equivocal ZK PoK that allows the simulator to equivocate the witness during the simulation (i.e., explaining the proof transcript with respect to either r_0 or r_1). We consider two instantiations of dual-mode PKE (based on the DDH and QR assumptions). For each implementation we design a concrete ZK PoK, proving that the prover knows r_σ with respect to $\sigma \in \{0,1\}$; see details below.

We define our OT protocol as follows,

Protocol 2 (One-sided OT (Π_{OT}))

- **Inputs:** *Sender* SEN *has* $x_0, x_1 \in \{0,1\}^q$ *and receiver* REC *has* $\sigma \in \{0,1\}$.
- **CRS:** CRS *such that* $(\text{CRS}, t) \leftarrow \mathsf{SetupDecryption}$.
- **The Protocol:**
  1. REC *sends* SEN PK, *where* $(\text{PK}, \text{SK}) \leftarrow \mathsf{dGen}(\text{CRS}, \sigma; r_\sigma)$. REC *calls* $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}}$ *with* (PK, r_σ).
  2. *Upon receiving* `Accept` *from* $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}}$ *and* PK *from* REC, SEN *generates* $c_0 \leftarrow \mathsf{dEnc}_{\text{PK}}(x_0, 0)$ *and* $c_1 \leftarrow \mathsf{dEnc}_{\text{PK}}(x_1, 1)$. SEN *calls* $\mathcal{F}_{\text{SC}}$ *twice with inputs* c_0 *and* c_1, *respectively.*
  3. *Upon receiving* (c_0, c_1), REC *outputs* $\mathsf{dDec}_{\text{SK}}(c_\sigma)$.

Theorem 5.1 *Assume the existence of one-sided NCE with constant number of PKE operations for message space* $\{0,1\}^q$ *and dual-mode PKE. Then Protocol 2 UC realizes* $\mathcal{F}_{\text{OT}}$ *in the* $(\mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}})$*-hybrid model in the presence of one-sided adaptive malicious adversaries with constant number of PKE operations for sender's input space* $\{0,1\}^q$, *where* $q = O(n)$ *and* n *is the security parameter.*

Proof: Let ADV be a probabilistic polynomial-time malicious adversary attacking Protocol 2 by adaptively corrupting one of the parties. We construct an adversary SIM for the ideal functionality $\mathcal{F}_{\text{OT}}$ such that no environment ENV distinguishes with a non-negligible probability whether it is interacting with ADV in the real setting or with SIM in the ideal setting. We recall that SIM interacts with the ideal functionality $\mathcal{F}_{\text{OT}}$ and the environment ENV. We refer to the interaction of SIM with $\mathcal{F}_{\text{OT}}$ and ENV as the external interaction. The interaction of SIM with the simulated ADV is the internal interaction. Upon computing $(\text{CRS}, t) \leftarrow \mathsf{SetupDecryption}(1^n)$, SIM proceeds as follows:

Simulating the communication with ENV. Every input value received by the simulator from ENV is written on ADV's input tape. Likewise, every output value written by ADV on its output tape is copied to the simulator's output tape (to be read by its environment ENV).

SEN is corrupted at the outset of the protocol. SIM begins by activating ADV and emulates the receiver by running $(\text{PK}, \text{SK}_0, \text{SK}_1) \leftarrow \mathsf{TrapKeyGen}(t)$. It then sends PK and an `Accept` message to ADV on behalf of $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}}$. Whenever ADV returns c_0, c_1 via $\mathcal{F}_{\text{SC}}$, SIM extracts SEN's inputs x_0, x_1 by

invoking $\mathsf{dDec}_{\mathrm{SK}_0}(c_0)$ and $\mathsf{dDec}_{\mathrm{SK}_1}(c_1)$ as in static case. It then sends x_0, x_1 to $\mathcal{F}_{\mathrm{OT}}$ and completes the execution playing the role of the receiver using an arbitrary σ.

Note that, in contrast to the hybrid execution where the receiver uses its real input σ to dGen in order to create public/secret keys pair, the simulator does not know σ and thus creates the keys using $\mathsf{TrapKeyGen}$. Nevertheless, when the CRS is set in a decryption mode the left public key is statistically indistinguishable from right public key. Furthermore, the keys $(\mathrm{PK}, \mathrm{SK}_i)$ that are generated by $\mathsf{TrapKeyGen}$ are statistically close to the keys generated by dGen with input bit i. This implies that the hybrid and simulated executions are statistically close.

REC is corrupted at the outset of the protocol. SIM begins by activating ADV and receives its public key PK and a witness r_σ on behalf of $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{LR}}}$. Given r_σ, SIM checks if PK is the left or the right key and use it to extract the receiver's input σ. If the adversary's message is invalid then SIM aborts, sending $\perp$ to the trusted party. Otherwise, it sends σ to $\mathcal{F}_{\mathrm{OT}}$, receiving back x_σ. Finally, SIM computes the sender's message using x_σ and an arbitrary $x_{1-\sigma}$.

Unlike in the hybrid execution, the simulator uses an arbitrary $x_{1-\sigma}$ instead of the real $x_{1-\sigma}$. However, a decryption mode implies computational privacy of $x_{1-\sigma}$. This follows from the same proof in [PVW08]. Therefore, the hybrid view is also computationally indistinguishable from the simulated view as in the static setting.

If none of the above corruption cases occur SIM invokes $(\mathrm{PK}, \mathrm{SK}_0, \mathrm{SK}_1) \leftarrow \mathsf{TrapKeyGen}(t)$ and sends PK to the sender. Note that the simulator knows a witness r_0 such that $\mathrm{PK} = \mathsf{dGen}(\mathrm{CRS}, 0; r_0)$ and a witness r_1 such that $\mathrm{PK} = \mathsf{dGen}(\mathrm{CRS}, 1; r_1)$.

SEN is corrupted between Steps 1 and 2. SIM trivially explains the sender's internal state since SEN did not compute any message so far. The simulator completes the simulation by playing the role of REC using an arbitrary bit σ as in the case when the sender is corrupted at the outset of the execution.

Indistinguishability for this case follows similarly to the prior corruption case when SEN is corrupted at the outset of the execution since the simulator uses the same simulation strategy as above. Namely, the adversary's simulated view is identically distributed in both simulation cases. This is because this view only contains the public key which is statically independent of σ is a decryption mode.

REC is corrupted between Steps 1 and 2. Upon corrupting the receiver SIM obtains σ, x_σ from $\mathcal{F}_{\mathrm{OT}}$ and explains the receiver's internal state as follows. It first explains r_σ as the witness given to $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{LR}}}$ and PK as the outcome of $\mathsf{dGen}(\mathrm{CRS}, \sigma; r_\sigma)$. The simulator completes the simulation playing the role of the honest sender with x_σ and an arbitrary $x_{1-\sigma}$.

Indistinguishability for this case follows similarly to the prior corruption case since the simulator did not emulate the sender's message yet.

If none of the above corruption cases occur then SIM chooses two arbitrary inputs x'_0, x'_1 for the sender and encrypts them using the dual-mode encryption. Denote these ciphertexts by c'_0, c'_1. SIM emulates the sender that sends these ciphertexts using $\mathcal{F}_{\mathrm{SC}}$.

SEN is corrupted after Step 2. Upon corrupting the sender, SIM obtains (x_0, x_1) from $\mathcal{F}_{\mathrm{OT}}$. It then explains the sender's internal state as follows. It first computes c_0, c_1 that respectively encrypt x_0 and x_1. It then explains c_0 and c_1 as being sent to the receiver using $\mathcal{F}_{\mathrm{SC}}$.

Indistinguishability follows as in the prior corruption case of the sender since the one-sided non-committing channel enables the simulator to "rewind" the simulation back, assuming that the sender is corrupted before simulating its message. Therefore, the same simulation strategy as before, of emulating the sender's incoming message using an arbitrary bit σ works here as well.

REC is corrupted after Step 2. Upon corrupting the receiver, SIM obtains REC's input and output (σ, x_σ) from $\mathcal{F}_{\text{OT}}$. It then explains the receiver's internal state as follows. It first explains r_σ as the witness given to $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{LR}}}$ and PK as the outcome of $\mathsf{dGen}(\text{CRS}, \sigma; r_\sigma)$. Finally, it explains the output of $\mathcal{F}_{\text{SC}}$ as c_0, c_1, so that c_σ is a valid encryption of x_σ under PK and $c_{1-\sigma}$ is an encryption of an arbitrary input computed as in the real execution.

Indistinguishability follows similarly to a static corruption case of the receiver since the simulator forces the second message from the sender to be simulated using an arbitrary $x_{1-\sigma}$ right as in the former receiver corruption.

■

Concrete instantiations. In the DDH-based instantiation the CRS is a Diffie-Hellman tuple (g_0, g_1, h_0, h_1) and the trapdoor is $\log_{g_0} g_1$. Moreover, the concrete ZK PoK functionality is $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DL,OR}}}$ which is invoked with the statement and witness $\big(((g_0 h_0, g_\sigma^r h_\sigma^r), (g_1 h_1, g_\sigma^r h_\sigma^r)), r\big)$, such that $\text{PK} = (g_\sigma^r, h_\sigma^r)$, $\text{SK} = r$ and $r \leftarrow \mathbb{Z}_p$. (See Appendix A for the DDH based OT of [PVW08]).

In the QR-based instantiation the CRS is a quadratic residue y and the trapdoor is s such that $y = s^2 \bmod N$ and N is an RSA composite. The concrete ZK PoK functionality is $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{QR,OR}}}$ which is invoked with the statement and witness $\big((y \cdot \text{PK}, \text{PK}), r\big)$, such that $\text{PK} = r^2/y^\sigma$, $\text{SK} = r$ and $r \leftarrow \mathbb{Z}_N^*$.

6 Constant Round One-Sided Adaptively Secure Computation

In the following we demonstrate the feasibility of one-sided adaptively secure two-party protocols in the presence of semi-honest and malicious adversaries. Our constructions are constant round UC secure, and use a number of non-committed bits that depends on the input and output lengths, as well as the circuit size.

6.1 A High-Level Overview of Yao's Garbling Technique

We briefly describe the garbling technique of Yao as described by Lindell and Pinkas in [LP09]. In this construction, the desired function f is represented by a boolean circuit C that is computed gate by gate from the input wires to the output wires. In the following, we distinguish four different types of wires used in a given boolean circuit: (**a**) circuit-input wires; (**b**) circuit-output wires; (**c**) gate-input wires (that enter some gate g); and (**d**) gate-output wires (that leave some gate g). The underlying idea is to associate every wire w with two random values, say k_w^0, k_w^1, such that k_w^0 represents the bit 0 and k_w^1 represents the bit 1. The garbled table for each gate maps random input values to random output values, with the property that given two input values it is only possible to learn the output value that corresponds to the output bit. This is accomplished by viewing the four potential inputs to the gate k_1^0, k_1^1 (values associated with the first input wire) and k_2^0, k_2^1 (values associated with the second input wire), as encryption keys. So that the output key values k_3^0, k_3^1 are encrypted under the appropriate input keys. For instance, let g be a NAND gate. Then, k_3^1 (that corresponds to bit 1) is encrypted under the pair of keys associated with the values $(0,0),\ (0,1),\ (1,0)$. Whereas, k_3^0 is encrypted under the pair of keys associated with $(1,1)$ which yields the following four ciphertexts

$$\mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^0}(k_3^1)),\ \ \mathsf{Enc}_{k_1^0}(\mathsf{Enc}_{k_2^1}(k_3^1)),\ \ \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^0}(k_3^1))\ \text{ and }\ \mathsf{Enc}_{k_1^1}(\mathsf{Enc}_{k_2^1}(k_3^0)),$$

where $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private key encryption scheme that has *chosen double encryption security* and an *elusive efficiently verifiable range*; see [LP09] for the formal definitions. These ciphertexts are randomly

permuted in order to obtain the garbled table for gate g. Then, given the input wire keys k_1^α, k_2^β that correspond to the bits α and β and the garbled table containing the four encryptions, it is possible to obtain the output wire key $k_3^{g(\alpha,\beta)}$. The description of the garbled circuit is concluded with the *output decryption tables*, mapping the random values on the circuit output wires back to their corresponding boolean values.

A useful lemma. Next, we state a useful lemma regarding garbled circuits taken verbatim from [LP07] and further stated in [Lin09, LP12]. The lemma states that it is possible to build a fake garbled circuit that outputs a fixed value $y = f(x_0, x_1)$ which is indistinguishable relative to an adversary who has only a single set of keys that corresponds to the inputs x_0, x_1. We rely on this lemma in our one-sided security proofs when P_1 is corrupted. Formally,

Lemma 6.1 *Given a circuit C and an output value y (of same length as the output of C) it is possible to construct a garbled circuit $\widetilde{GC}$ such that:*

1. *The output of $\widetilde{GC}$ is always y, regardless of the garbled values that are provided for the input wires of P_0 and P_1, and*
2. *If $y = f(x_0, x_1)$, then no non-uniform PPT adversary* ADV *can distinguish between the distribution ensemble of $\widetilde{GC}$ and a single arbitrary garbled value for every input wire, and the distribution ensemble consisting of a real garbled version of C, together with garbled values that correspond to x_0 for P_0's input wires and to x_1 for P_1's input wires.*

6.2 One-Sided Yao for Semi-Honest Adversaries

Our first construction adapts the semi-honest two-party protocol [Yao82, LP09] into the one-sided adaptive setting at a cost of $\mathcal{O}(|C|)$ private key operations and $\mathcal{O}(|\mathsf{input}| + |\mathsf{output} + |C|)$ public key operations. Namely, we show that one-sided security can be obtained by communicating the keys that correspond to the input/output wires and the garbled circuit via a one-sided non-committing channel.

Informally, the input keys that correspond to P_0's input are transferred to P_1 using one-sided NCE[6], whereas P_1's input keys are sent using one-sided OT. Moreover, the garbled circuit is sent to P_1 using one-sided NCE. P_1 then evaluates the garbled circuit and computes its output. Finally, P_1 sends P_0 the output using one-sided NCE. We note that obtaining the input and output via one-sided primitives is crucial to our proof since it means that P_0 is not committed to these values ahead of time. In addition, we were able to prove indistinguishability between a real and a simulated executions when P_1 is corrupted, only if the garbled circuit is not committed (since the standard reductions to the security of the encryption scheme for garbling do not work in the adaptive setting). For simplicity we only consider deterministic and same-output functionalities. This can be further generalized using the reductions specified in [Gol04]. The formal description of our one-sided semi-honest protocol Π_f^{SH} is given below in the $\{\mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{OT}}\}$-hybrid model.

Protocol 3 (One-sided adaptively secure semi-honest Yao (Π_f^{SH}))

- **Inputs:** *P_0 has $x_0 \in \{0,1\}^n$ and P_1 has $x_1 \in \{0,1\}^n$. Let $x_0 = x_0^1, \ldots, x_0^n$ and $x_1 = x_1^1, \ldots, x_1^n$.*
- **Auxiliary Input:** *A boolean circuit C such that for every $x_0, x_1 \in \{0,1\}^n$, $C(x,y) = f(x,y)$ where $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. Furthermore, we assume that C is such that if a circuit-output wire leaves some gate, then the gate has no other wires leading from it into other gates (i.e. no circuit-output wire is also a gate-output wire). Likewise, a circuit-input wire that is also a circuit-output wire enters no gates.*

[6]The keys can be transferred via somewhat NCE with equivocality parameter $\ell = 2$ to improve the exact efficiency of the two party computation protocol. Since the asymptotic complexity does not change, we prefer to use one-sided NCE to simplify the security proof.

Convention: *Unless specified differently, $i \in [n]$. We further assume that the gates of circuit C induce a topological sort.*

- **The Protocol:**

  1. **Setup and garbled circuit computation.** *P_0 constructs garbled circuit $G(C)$ as described in Section 6.1 subject to the constraint that the keys corresponding to each circuit-output wire have a distinct most significant bit.*
  2. **Transferring the garbled circuit and input keys to $\mathbf{P}_1$.** *Let (k_i^0, k_i^1) be the key pair corresponding to the circuit-input wire that takes the ith bit of x_0 and let (k_{n+i}^0, k_{n+i}^1) be the key pair corresponding to the circuit-input wire that takes the ith bit of x_1. Then,*
     (a) *For all $i \in [1, \ldots, n]$, P_0 sends $k_i^{x_0^i}$ by making an ideal call to $\mathcal{F}_{\text{SC}}$.*
     (b) *For all $i \in [1, \ldots, n]$, P_0 and P_1 call $\mathcal{F}_{\text{OT}}$ with input (k_{n+i}^0, k_{n+i}^1) and x_1^i, respectively. Let $k_{n+i}^{x_1^i}$ denotes P_1's ith output.*
     (c) *P_0 sends P_0 the garbled circuit $G(C)$ via $\mathcal{F}_{\text{SC}}$.*
  3. **Circuit evaluation.** *P_1 evaluates $G(C)$ on the above input keys and obtains its output $y = f(x_0, x_1)$.*
  4. **Output communication.** *P_1 sends y via $\mathcal{F}_{\text{SC}}$.*

Theorem 6.1 (One-sided semi-honest) *Let f be a deterministic same-output functionality and assume that the encryption scheme for garbling has indistinguishable encryptions under chosen plaintext attacks, and an elusive and efficiently verifiable range. Furthermore, assume that $\mathcal{F}_{\text{OT}}$ is realized in the presence of one-sided semi-honest adversaries with constant number of PKE operations for sender's input space $\{0,1\}^q$, where $q = O(n)$ and n is the security parameter. Then Protocol 3 UC realizes $\mathcal{F}_f$ in the presence of one-sided semi-honest adversaries at a cost of $\mathcal{O}(|C|)$ private key operations and $\mathcal{O}(|\mathsf{input}| + |\mathsf{output}| + |C|)$ public key operations.*

It is important to note that a simpler variant of static OT can be used in our protocol. In particular, the only input to the OT that is needed to be equivocated in Yao's protocol is P_1's input. This is because in our simulation P_0 always enters the correct input keys. Therefore, the [PVW08] OT protocol can be used here when the CRS is set in a decryption mode (which ensures the equivocation of the receiver's input), together with a ZK proof of knowledge that allows extraction; see Section 5 for further details. Nevertheless, a different simulation strategy may require the stronger security notion with the ability to equivocate either one of the parties.

Proof: Our proof is shown in the $(\mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{OT}})$-hybrid model. Let ADV be a probabilistic polynomial-time semi-honest adversary attacking Protocol 3 by adaptively corrupting one of the parties. We construct an adversary SIM for the ideal functionality $\mathcal{F}_f$ such that no environment ENV distinguishes with a non-negligible probability whether it is interacting with ADV in the real setting or with SIM in the ideal setting. We recall that SIM interacts with the ideal functionality $\mathcal{F}_f$ and the environment ENV. We refer to the interaction of SIM with $\mathcal{F}_f$ and ENV as the external interaction. The interaction of SIM with the simulated ADV is the internal interaction. We now explain the actions of the simulation for the following corruption cases: **(1)** No corruption takes place; **(2)** Corruption takes place at the outset. **(3)** Corruption takes place between Steps 2 and 3. **(4)** Corruption takes place at the end. We now describe the simulator for all these cases considering the corruption of each party. These cases cover all potential cases of corruption.

No corruption. When no corruption takes place the simulator simulates both P_0 and P_1 as follows:

1. **Setup and garbling circuit C.** No communication is carried out in this step. Internally, SIM chooses $2n$ random keys for P_0's input denoted by $k_1^0, k_1^1, \ldots, k_n^0, k_n^1$, $2n$ random keys for P_1's input denoted by $k_{n+1}^0, k_{n+1}^1, \ldots, k_{2n}^0, k_n^1$ and $2n$ random keys for the output denoted by $k_{2n+1}^0, k_{2n+1}^1, \ldots, k_{3n}^0, k_{3n}^1$.

2. **Transferring the garbled circuit and input keys to $\mathbf{P}_1$.** No communication is carried out, since ideal calls to $\mathcal{F}_{\text{SC}}$ are made for sending the garbled inputs of P_0 and the garbled circuit.

3. **Circuit evaluation.** No communication is carried out in this phase.

4. **Output.** No communication is carried out since an ideal call to $\mathcal{F}_{\text{SC}}$ is made for sending the output.

Note that when the simulator simulates the parties in the ideal calls to $\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{SC}}$, the simulator merely needs to inform ADV about the invocations taking place. The adversary sees nothing during the simulation of such steps. It is simple to verify that an eavesdropper does not learn anything meaningful about the parties' inputs since in the hybrid setting the adversary does not see any communication at all.

Corruption at the outset of the protocol execution.

- P_0 **is corrupted.** SIM receives P_0's input x_0 and its output y. It then plays the role of P_1 as in the no corruption case except that in the final step SIM simulates P_1 sending y via $\mathcal{F}_{\text{SC}}$.

- P_1 **is corrupted.** SIM receives P_1's input x_1 and its output y. It then mimics the [LP09] simulation that is proven for the static case, building and sending a fake circuit that always computes y on the behalf of $\mathcal{F}_{\text{SC}}$.

Security in this case is proven as in the static setting and is identical to the [LP09] proof. (The proof of security when P_1 is corrupted relies on Lemma 6.1.) Specifically, the only difference is that the communication is transferred via non-committing channels. Nevertheless, the elements that are transferred via these channels are as in the static simulation.

Corruption between Steps 2 and 3.

- P_0 **is corrupted.** The simulator simulates the parties' actions as in the first two steps of the no corruption case. Upon corrupting P_0 the simulator receives P_0's input x_0 and its output y. It then explains the internal state of P_0 until Step 2 as follows: SIM completes the construction of the garbled circuit $G(C)$ for C as the honest P_0 would do using the keys picked in step 1 why simulating P_0. It then explains the keys and the garbled circuit construction. Moreover, P_0's input to the ith $\mathcal{F}_{\text{OT}}$ call are explained as the pair of input keys $\{(k^0_{n+i}, k^1_{n+i})\}$. Finally, P_0's input to the ith invocation of $\mathcal{F}_{\text{SC}}$ is explained as $k_i^{x_0^i}$ where x_0^i is the ith bit of x_0. SIM completes the simulation playing the role of P_1 as in the case when P_0 is corrupted in the outset.

  In $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{SC}}$ hybrid model, the view of ADV for this corruption case is identically distributed to its view when corrupting P_0 in the outset.

- P_1 **is corrupted.** SIM simulates the parties' actions as in the first two steps of the no corruption case. Upon corrupting P_1 the simulator receives P_1's input x_1 and its output y. It then explains the internal state of P_1 as follows. P_1's input and output pair for the ith $\mathcal{F}_{\text{OT}}$ call are explained as $(x_1^i, k_{n+i}^{x_1^i})$. Moreover, the outcome from the ith call to $\mathcal{F}_{\text{SC}}$ for transferring P_0's input keys is explained as k_i^0 (the key corresponding to bit 0), whereas the outcome from the instance of $\mathcal{F}_{\text{SC}}$ that carries the garbled circuit is explained as a fake garbled circuit $\widetilde{GC}$ that always outputs y, exactly as in the simulation of [LP09]. This implies that the proof can be reduced to the proof of [LP09] for the case that P_1 is statically corrupted since the adversary's view in the hybrid execution is identical to its view in [LP09]. Then SIM completes the simulation playing the role of P_0 as in the case when P_1 is corrupted in the outset. Finally, SIM outputs whatever the adversary does.

In more details, let $\mathcal{H}_0$ denote the simulated game where the simulator explains that the fake circuit $\widetilde{GC}$ (that always outputs y) as being sent to P_1 via $\mathcal{F}_{\mathrm{SC}}$. Moreover, let $\mathcal{H}_1$ denote the hybrid execution where the circuit is garbled correctly. Since the garbled circuit and its inputs are transferred using ideal calls to $\mathcal{F}_{\mathrm{OT}}$ and $\mathcal{F}_{\mathrm{SC}}$ (and so the adversary does not see any communication), the difference between the above two hybrids games relative to the adversary's point of view is regarding the way the circuit is garbled as well as the garbled inputs keys that are associated with the input of P_0. The indistinguishability of the above two hybrids executions can be immediately proven relying on Lemma 6.1. This concludes the proof for the current corruption case.

Post execution corruption. The simulator completes the simulation as in the no corruption case. Upon corrupting one of the parties the simulator explains the adversary's internal state as in the previous corruption case after receiving the input and the output of the corrupted party. Finally, the simulator explains that P_1 has sent y via $\mathcal{F}_{\mathrm{SC}}$ in the case P_1 is corrupted. Similarly, the simulator explains that P_0 has received y in the last step in the case P_0 is corrupted. Since y is transferred through $\mathcal{F}_{\mathrm{SC}}$, the above does not make a difference between the current two corruption cases and the cases considering the corruption occurs between steps 2 and 3 when we are in $\mathcal{F}_{\mathrm{SC}}$ hybrid model. Therefore the security for this case can be easily achieved given the previous proof. ■

6.3 Security against Malicious Adversaries

Next, we modify Π_f^{SH} and adapt the cut-and-choose OT protocol introduced in [LP12] in order to achieve security against malicious adversaries. The idea of the cut-and-choose technique is to ask P_0 to send s garbled circuits and later open half of them (aka, *check circuits*) by the choice of P_1. This ensures that with very high probability the majority of the unopened circuits (aka, *evaluation circuits*) are valid. The cut-and-choose OT primitive of [LP12] allows P_1 to choose a secret random subset $\mathcal{J}$ of size $s/2$ for which it learns both keys for each input wire that corresponds to the check circuits, and the keys associated with its input with respect to the evaluation circuits. As in the semi-honest setting, the garbled circuits are encrypted using one-sided NCE.

In order to ensure that P_0 hands P_1 consistent input keys for all the circuits, the [LP12] protocol ensures that the keys associated with P_0's input are obtained via a Diffie-Hellman pseudorandom synthesizer [NR95]. Namely, P_0 chooses values $g^{a_1^0}, g^{a_1^1}, \ldots, g^{a_n^0}, g^{a_n^1}$ and $g^{c_1}, \ldots, g^{c_s}$, where n is the input/output length, s is the cut-and-choose parameter and g is a generator of a prime order group $\mathbb{G}$. So that the pair of keys associated with the ith input of P_0 in the jth circuit is $(g^{a_i^0 c_j}, g^{a_i^1 c_j})$.[7] Given values $\{g^{a_i^0}, g^{a_i^1}, g^{c_j}\}$ and any subset of keys associated with P_0's input, the remaining keys associated with its input are pseudorandom by the DDH assumption. Furthermore, when the keys are prepared this way P_0 can efficiently prove that it used the same input for all circuits. P_1 then evaluates the evaluation circuits and takes the majority value for the final output. In Section 6.3.1 we demonstrate how to adapt the cut-and-choose OT protocol into the one-sided setting using the building blocks introduced in this paper. This requires dealing with new subtleties regarding the system parameters and the ZK proofs.

6.3.1 One-sided Single Choice Cut-and-Choose OT

We describe next the single choice cut-and-choose OT functionality $\mathcal{F}_{\mathrm{CCOT}}$ from [LP12] and present a protocol that implements this functionality with UC one-sided malicious security. In Section 6.3.2 we briefly

[7] The actual key pair used in the circuit garbling is derived from $(g^{a_i^0 c_j}, g^{a_i^1 c_j})$ using an extractor. A universal hash function is used in [LP12] for this purpose, where the seeds for the function are picked by P_0 before it knows $\mathcal{J}$.

describe our batch single choice cut-and-choose OT construction using a single choice cut-and-choose OT, which is used as a building block in our two-party protocol. Formally, $\mathcal{F}_{\mathrm{CCOT}}$ is defined as follows

1. **Inputs:**
   (a) SEN inputs a vector of pairs $\{(x_0^j, x_1^j)\}_{j=1}^s$.
   (b) REC inputs a bit σ and a set of indices $\mathcal{J} \subset [s]$ of size exactly $s/2$.
2. **Output:** If $\mathcal{J}$ is not of size $s/2$, then SEN and REC receive $\perp$ as output. Otherwise,
   (a) For all $j \in \mathcal{J}$, REC obtains the pair (x_0^j, x_1^j).
   (b) For all $j \notin \mathcal{J}$, REC obtains x_σ^j.

This functionality is implemented in [LP12] by invoking the DDH based [PVW08] OT s times (see Appendix A for the complete details), where the receiver generates the system parameters in a decryption mode for $s/2$ indices corresponding to $\mathcal{J}$ and the remaining system parameters are generated in a messy mode. The decryption mode trapdoor enables the receiver to learn both sender's inputs for the instances corresponding to $\mathcal{J}$. This idea is coupled with two proofs that are run by the receiver: (**i**) a ZK PoK for proving that half of the system parameters set is in a messy mode which essentially boils down to a ZK PoK realizing functionality $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DH,COMP}(s,s/2)}}$ (namely, the statement is a set of s tuples and the prover proves the knowledge of $s/2$ Diffie-Hellman tuples within this set). (**ii**) A ZK PoK to ensure that the same input bit σ has been used for all s instances which boils down to a ZK proof realizing functionality $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DH,OR}(s)}}$ (namely, the statement contains two sets of tuples, each of size s, for which the prover proves that one of the sets contains DH tuples). See Section 7.2 for more details on the ZK PoK functionalities.

Our first step towards making the [LP12] construction one-sided adaptively secure is to invoke our one-sided OT scheme s times with all system parameters in a decryption mode. Notably, we cannot use the messy mode for the $s/2$ instances not in $\mathcal{J}$ as in the static settings since that would preclude the equivocation of the receiver's bit. Similarly to [LP12], our constructions have two phases; a *setup phase* and a *transfer phase*. In the setup phase, the receiver generates the system parameters in a decryption mode for the $s/2$ OTs corresponding to indices in $\mathcal{J}$, while the remaining system parameters are generated in the same mode but in a way that does not allow REC to learn the trapdoor. This is obtained by fixing two random generators g_0, g_1, so that the receiver sets the first component of every CRS from the system parameters to be g_0. Moreover, the second component in positions $j \notin \mathcal{J}$ is a power of g_1, else this element is a power of g_0. Note that REC does not know $\log_{g_0} g_1$ which is the decryption mode trapdoor for $j \notin \mathcal{J}$. To ensure correctness, REC proves that it knows the discrete logarithm of the second element with respect to g_1 of at least $s/2$ pairs. This is achieved using a witness equivocal proof for functionality $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DL,COMP}(s,s/2)}}$ described in Section 7.2.

In the transfer phase, the receiver uses these system parameters to create a public/secret key pair for each OT execution, for keys not in the set $\mathcal{J}$. For the rest of the OT executions the receiver invokes the $\mathsf{TrapKeyGen}$ algorithm of the dual-mode PKE and obtains a public key and two secret keys that enable it to decrypt both of the sender's inputs. In order to ensure that the receiver uses the same input bit σ for all OTs the receiver proves its behavior using the proof specified above. Finally, we prove the equivocality of the sender's input and the receiver's output based on our one-sided NCE.

Formally, let the DDH based dual-mode PKE of [PVW08] be specified by $\Pi_{\mathrm{DUAL}} = (\mathsf{SetupMessy}, \mathsf{SetupDecryption}, \mathsf{dGen}, \mathsf{dEnc}, \mathsf{dDec}, \mathsf{FindBranch}, \mathsf{TrapKeyGen})$. We denote our one-sided OT by Π_{CCOT} and present it in the $(\mathcal{F}_{\mathrm{SC}}, \mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DL,COMP}(s,s/2)}}, \mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DH,OR}(s)}})$-hybrid model.

Protocol 4 (One-sided adaptive single choice cut-and-choose OT (Π_{CCOT}))

- **Inputs:** SEN *inputs a vector of pairs* $\{(x_0^i, x_1^i)\}_{i=1}^s$ *and* REC *inputs a bit* σ *and a set of indices* $\mathcal{J} \subset [s]$ *of size exactly* $s/2$.

- **Auxiliary Inputs:** *Both parties hold a security parameter 1^n and $\mathbb{G}, p$, where $\mathbb{G}$ is an efficient representation of a group of order p and p is of length n.*
- **CRS:** *The CRS consists of a pair of random group elements g_0, g_1 from $\mathbb{G}$.*
- **Setup phase:**
  1. REC *chooses a random $x_j \in \mathbb{Z}_p$ and sets $g_1^j = g_0^{x_j}$ for all $j \in \mathcal{J}$ and $g_1^j = g_1^{x_j}$ otherwise.*
     For all j, REC *chooses a random $y_j \in \mathbb{Z}_p$ and sets* $\mathrm{CRS}_j = \left(g_0, g_1^j, h_0^j = (g_0)^{y_j}, h_1^j = (g_1^j)^{y_j}\right)$.
     Finally, for all $j \in \mathcal{J}$, REC *stores trapdoor $t_j = x_j$. It then sends $\{\mathrm{CRS}_j\}_{j=1}^s$ to* SEN.
  2. REC *calls $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DL,COMP}(s,s/2)}}$ with $(\{g_1, g_1^j\}_{j=1}^s, \{x_j\}_{j\in\mathcal{J}})$ to prove the knowledge of the discrete logarithms of $s/2$ values within the second element in $\{\mathrm{CRS}_j\}_j$ and with respect to g_1.*
- **Transfer phase (repeated in parallel for all j):**
  1. *For all $j \notin \mathcal{J}$,* REC *computes* $(\mathrm{PK}_j, \mathrm{SK}_j) = ((g_j, h_j), r_j) \leftarrow \mathsf{dGen}(\mathrm{CRS}_j, \sigma)$.
     For all $j \in \mathcal{J}$, REC *computes* $(\mathrm{PK}_j, \mathrm{SK}_j^0, \mathrm{SK}_j^1) = ((g_j, h_j), r_j, r_j/t_j) \leftarrow \mathsf{TrapKeyGen}(\mathrm{CRS}_j, t_j)$.
     Finally, REC *sends the set $\{\mathrm{PK}_j\}_{j=1}^s$ and stores the secret keys.*
  2. REC *calls $\mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DH,OR}(s)}}$ with input $((\{(g_0, h_0^j, g_j, h_j)\}_{j=1}^s, \{(g_1^j, h_1^j, g_j, h_j)\}_{j=1}^s), \{r_j\}_{j=1}^s)$ to prove that all the tuples in one of the sets $\{(g_0, h_0^j, g_j, h_j)\}_{j=1}^s$ or $\{(g_1^j, h_1^j, g_j, h_j)\}_{j=1}^s$ are DH tuples.*
  3. *For all j,* SEN *generates $c_0^j \leftarrow \mathsf{dEnc}_{\mathrm{PK}_j}(x_0^j, 0)$ and $c_1^j \leftarrow \mathsf{dEnc}_{\mathrm{PK}_j}(x_1^j, 1)$. Let $c_0^j = (c_{00}^j, c_{01}^j)$ and $c_1^j = (c_{10}^j, c_{11}^j)$.* SEN *calls $\mathcal{F}_{\mathrm{SC}}$ with c_{01}^j and c_{11}^j.*
- **Output:** *Upon receiving (c_{01}^j, c_{11}^j) from $\mathcal{F}_{\mathrm{SC}}$,*
  1. REC *outputs $x_\sigma^j \leftarrow \mathsf{dDec}_{\mathrm{SK}_j}(c_\sigma^j)$ for all $j \notin \mathcal{J}$.*
  2. REC *outputs $(x_0^j, x_1^j) \leftarrow (\mathsf{dDec}_{\mathrm{SK}_j^0}(c_0^j), \mathsf{dDec}_{\mathrm{SK}_j^1}(c_1^j))$ for all $j \in \mathcal{J}$.*

Theorem 6.2 *Assume that the DDH assumption is hard in $\mathbb{G}$. Then Protocol 4 UC realizes $\mathcal{F}_{\mathrm{CCOT}}$ in the $(\mathcal{F}_{\mathrm{SC}}, \mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DL,COMP}(s,s/2)}}, \mathcal{F}_{\mathrm{ZKPoK}}^{\mathcal{R}_{\mathrm{DH,OR}(s)}})$-hybrid model in the presence of one-sided malicious adversaries.*

Proof: Let ADV be a probabilistic polynomial-time malicious adversary attacking Protocol 4 by that adaptively corrupting one of the parties. We construct an adversary SIM for the ideal functionality of a single choice cut-and-choose oblivious transfer $\mathcal{F}_{\mathrm{CCOT}}$ such that no environment ENV distinguishes with a non-negligible probability whether it is interacting with ADV in the real setting or with SIM in the ideal setting. We recall that SIM interacts with the ideal functionality and the environment ENV. We refer to the interaction of SIM with the ideal functionality $\mathcal{F}_{\mathrm{CCOT}}$ and ENV as the external interaction. The interaction of SIM with the simulated ADV is the internal interaction. We describe the simulator's strategy for all corruption cases. SIM begins by creating a CRS (g_0, g_1) and storing $x = \log_{g_0} g_1$. It proceeds as follows:

Simulating the communication with ENV. Every input value received by the simulator from ENV is written on ADV's input tape. Likewise, every output value written by ADV on its output tape is copied to the simulator's output tape (to be read by its environment ENV).

The sender is corrupted at the onset of the protocol. SIM begins by activating ADV and emulates the receiver as follows. In the setup phase it picks s system parameters in a decryption mode in which it knows their trapdoors. Namely for each $j = 1, \ldots, s$, it creates $\mathrm{CRS}_j = (g_0, g_1^j, h_0^j, h_1^j)$ where $g_1^j = (g_0)^{x_j}$, $h_0^j = (g_0)^{y_j}$ and $h_1^j = (g_1^j)^{y_j} = (g_0^{x_j})^{y_j}$ for random x_j's and y_j's, and records the trapdoor $t_j = x_j$. The simulator further computes $x_j' = \log_{g_1} g_1^j$ for all j using the knowledge of $x = \log_{g_0} g_1$. It then chooses an arbitrary set $\mathcal{J}'$ of size $s/2$ and sends `Accept` to ADV on behalf

of $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DL,COMP}(s,s/2)}}$ for the statement $\{g_1, g_1^j\}_{j=1}^s$. Note that the simulator knows the discrete log for each pair of (g_1, g_1^j) in the statement.

SIM also emulates REC by sending the adversary the system parameters.

In the transfer phase the simulator invokes $\mathsf{TrapKeyGen}$ for all $j = 1, \ldots, s$ and computes $(\text{PK}_j, \text{SK}_j^0, \text{SK}_j^1) = ((g_j, h_j), r_j, r_j/t_j) \leftarrow \mathsf{TrapKeyGen}(\text{CRS}_j, t_j)$ for $j = 1, \ldots, s$, and sends the public keys to SEN. It further sends `Accept` to ADV on behalf of $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}(s)}}$. Upon receiving ADV's message, SIM extracts the sender's input (x_0^j, x_1^j) using $\text{SK}_j^0, \text{SK}_j^1$ for every $j = 1, \ldots, s$ and sends it to the ideal functionality $\mathcal{F}_{\text{CCOT}}$.

Note that the adversary's views differ only with respect to the ZK statements, since in a decryption mode the receiver's bit is perfectly hidden as well as the subset picked by the receiver. Now, since the proofs are run via ideal calls the simulated and hybrid views are statistically close.

The receiver is corrupted at the onset of the protocol. SIM begins by activating ADV and emulates the sender by receiving the witnesses on the behalf of the ZK PoK functionalities in the setup and transfer phases. It extracts $\mathcal{J}$ and σ and sends them to $\mathcal{F}_{\text{CCOT}}$, receiving back (x_0^j, x_1^j) for all $j \in \mathcal{J}$ and x_σ^j for all $j \notin \mathcal{J}$. SIM chooses an arbitrary $x_{1-\sigma}^j$ for all $j \notin \mathcal{J}$ and emulates the role of SEN using inputs (x_0^j, x_1^j) for all $j = 1, \ldots, s$.

The difference between the simulated and hybrid views is with respect to inputs $x_{1-\sigma}^j$ for all $j \notin \mathcal{J}$ for which the simulator uses arbitrary values. Indistinguishability is implied by the privacy of the dual-mode PKE when a left ciphertext is computed with a right key (or vice versa), just as in the static setting proof.

If none of the parties gets corrupted at the onset of the protocol execution SIM plays the role of the receiver in the setup phase using an arbitrary subset $\mathcal{J}'$ for the receiver. Note that SIM knows all the witnesses for the proof in the setup phase (i.e., the discrete logarithms of $\{g_1^j\}_{j=1}^s$ with respect to g_1 for all j values). It can thus equivocate the proof with respect to the real set $\mathcal{J}$. SIM further plays the role of the receiver in the transfer phase using an arbitrary σ' for the receiver. Specifically the simulator simulates the receiver by invoking $\mathsf{TrapKeyGen}$ for all $j = 1, \ldots, s$, computing $(\text{PK}_j, \text{SK}_j^0, \text{SK}_j^1) = ((g_j, h_j), r_j, r_j/t_j) \leftarrow \mathsf{TrapKeyGen}(\text{CRS}_j, t_j)$. It then sends the public keys to SEN and an `Accept` message on behalf of $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}(s)}}$. Note that SIM knows witnesses for both sub statements $\{(g_0, h_0^j, g_j, h_j)\}_{j=1}^s$ and $\{(g_1^j, h_1^j, g_j, h_j)\}_{j=1}^s$, which equal $\{r_j\}_{j=1}^s$ for the first set and $\{r_j/t_j\}_{j=1}^s$ for the second set.

The sender is corrupted between Steps 2 and 3. Upon corrupting SEN, SIM explains the internal state of the sender by honestly presenting the randomness used so far on the sender's behalf. Finally, SIM completes the execution in the transfer phase by playing the role of the receiver using an arbitrarily chosen σ'. Indistinguishability follows due to the same argument as in the previous corruption case since the simulator follows the same strategy.

The receiver is corrupted between Steps 2 and 3. Upon corrupting REC, SIM receives $\mathcal{J}$ and σ from $\mathcal{F}_{\text{CCOT}}$, as well as (x_0^j, x_1^j) for all $j \in \mathcal{J}$ and x_σ^j for all $j \notin \mathcal{J}$. It then explains the internal state of REC as follows. Namely, it first explains the witness for the ZK PoK functionality $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DL,COMP}(s,s/2)}}$ as the discrete logarithms of $\{g_1^j\}_{j \notin \mathcal{J}}^s$ with respect to g_1. It also explains the witness for $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}(s)}}$ as the witness for the σth set. Finally, it plays the role of the sender as in the previous corruption case. Indistinguishability follows similarity to the previous corruption case due to the security of the dual-mode PKE and the fact that the simulator follows the same strategy.

If none of the parties is corrupted until now, SIM plays the role of the sender in the transfer phase using arbitrary $(x_0'^j, x_1'^j)$ for all $j = 1, \ldots, s$.

The sender is corrupted after Step 3 is concluded. Upon corrupting SEN, SIM receives (x_0^j, x_1^j) for all $j = 1, \ldots, s$ from $\mathcal{F}_{\text{CCOT}}$. It then explains the internal state of SEN as in the previous corruption case, and further explains the inputs to $\mathcal{F}_{\text{SC}}$ as ciphertexts that encrypt the real inputs. Indistinguishability follows from the fact that the receiver's input is statistically hidden given the public keys.

The receiver is corrupted after Step 3 is concluded. Upon corrupting REC, SIM receives $\mathcal{J}, \sigma$, from $\mathcal{F}_{\text{CCOT}}$ as well as (x_0^j, x_1^j) for all $j \in \mathcal{J}$ and x_σ^j for all $j \notin \mathcal{J}$. It then explains the internal state of REC as in the previous corruption case, and further explains the messages received from $\mathcal{F}_{\text{SC}}$ as the encryptions of $\{x_0^j, x_1^j\}_{j \in \mathcal{J}}$ and $\{x_\sigma^j\}_{j \notin \mathcal{J}}$. Indistinguishability follows as above.

■

6.3.2 Malicious One-Sided Adaptively Secure Two-Party Computation

First, we remark that the single choice cut-and-choose protocol is executed for every input bit of P_1 in the main two-party computation protocol, but with respect to *the same* set $\mathcal{J}$. In order to ensure that the same $\mathcal{J}$ is indeed used the parties engage in a *batch single choice cut-and-choose OT* where a single setup phase is run first, followed by n parallel invocations of the transfer phase. We note that CRS and the set $\mathcal{J}$ are fixed in the setup phase and remain the same for all n parallel invocations of the transfer phase. We denote the batch functionality by $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ and the protocol by $\Pi_{\text{CCOT}}^{\text{BATCH}}$. We are now ready to formally describe our protocol Π_f^{MAL} that computes any functionality f on inputs x_0 and x_1.

Protocol 5 (One-sided adaptively secure malicious Yao (Π_f^{MAL}))

- **Inputs:** *P_0 has $x_0 \in \{0,1\}^n$ and P_1 has $x_1 \in \{0,1\}^n$. Let $x_0 = x_0^1, \ldots, x_0^n$ and $x_1 = x_1^1, \ldots, x_1^n$.*
- **Auxiliary Input:** *A boolean circuit C such that for every $x_0, x_1 \in \{0,1\}^n$, $C(x,y) = f(x,y)$ where $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. Furthermore, we assume that C is such that if a circuit-output wire leaves some gate, then the gate has no other wires leading from it into other gates (i.e. no circuit-output wire is also a gate-output wire). Likewise, a circuit-input wire that is also a circuit-output wire enters no gates.*

  Convention: *Unless specified differently, $i \in [n]$. We further assume that the gates of circuit C induce a topological sort.*
- **The Protocol:**

  1. **Garbled circuit computation.** *P_0 constructs s independent garbled circuits for C as follows:*
     (a) *P_0 picks n pairs of random values $((a_1^0, a_1^1), \ldots, (a_n^0, a_n^1)) \in \mathbb{Z}_q$ and $c_1, \ldots, c_s \in_R \mathbb{Z}_q$.*
     (b) *Let $w_1, \ldots, w_n$ be the input wires corresponding to P_0's input in C, and denote by $w_{i,j}$ the instance of wire w_i in the jth garbled circuit. Further let, $k_{i,j}^b$ denotes the key associated with bit b on wire $w_{i,j}$. Then P_0 sets the keys for its input wires to $(k_{i,j}^0 = H(g^{a_i^0 c_j}), k_{i,j}^1 = H(g^{a_i^1 c_j}))$, where H is a randomness extractor such as a universal hash function [JC79, HILL99, DGH+04].*
     (c) *P_0 constructs s independent garbled circuits for C, denoted as $GC_1, \ldots, GC_s$, using random keys except for the wires $w_1, \ldots, w_n$ for which the keys are as above.*
  2. **Oblivious transfers.** *The parties call $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ with their respective inputs and obtain outputs as follows:*
     (a) *P_0 defines vectors $\vec{z_1}, \ldots, \vec{z_n}$, where $\vec{z_i}$ contains the s pairs of random symmetric keys associated with P_1's ith input bit x_1^i in all garbled circuits $GC_1, \ldots, GC_s$.*
     (b) *P_1 inputs a random subset $\mathcal{J} \subset [s]$ of size $s/2$ and the bits $x_1^1, \ldots, x_1^n$.*

(c) *P_1 receives from $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ all the keys that are associated with its input wires for the circuits GC_i with $i \in \mathcal{J}$ (denoted as the check circuits). In addition, it receives the keys that correspond to its input for the remaining circuits (denoted the evaluation circuits).*

3. **Sending garbled circuits and commitments.** *P_0 sends P_1 s copies of the garbled circuit, encrypted using one-sided NCE and the values $((g^{a_1^0}, g^{a_1^1}), \ldots, (g^{a_n^0}, g^{a_n^1}), (g^{c_1}, \ldots, g^{c_s}))$ along with the "seed" of the hash function H which constitutes the commitments to the input keys on the wires associated with P_0's input.*[8]

4. **Revealing $\mathcal{J}$.** *P_1 reveals $\mathcal{J}$ and proves that it used this subset in the cut-and-choose OT protocol by sending the* pair *of keys associated with P_1's first input bit in each check circuit i.e. for every GC_i with $i \in \mathcal{J}$. Note that P_1 knows the key pair only for the check circuits. If the values received from P_1 are wrong, then P_0 aborts.*

5. **Decommitting $\mathbf{P_0}$'s input keys.** *In order to let P_1 know the keys for the input wires of P_0 within the check circuits, P_0 sends c_j for all $j \in \mathcal{J}$. P_1 computes the key pair $(H(g^{a_i^0 c_j}), H(g^{a_i^1 c_j}))$.*

6. **Verifying the check circuits.** *P_1 verifies the validity of the check circuits using all the keys associated with their input wires. This ensures that the evaluation circuits are correct with high probability.*

7. **Sending the garbled inputs for the evaluation circuits.** *In order to complete the evaluation phase P_1 is given the keys for the input wires of P_0. P_0 must be forced to give the keys that are associated with the same input for all circuits. Specifically, the following code is executed for all input bits of P_0:*

   (a) *For every evaluation circuit GC_j, P_0 sends $\tau_{i,j} = g^{a_i^{x_0^i} c_j}$ using an instance of $\mathcal{F}_{\text{SC}}$[9], where x_0^i is the ith input bit of P_0.*

   (b) *P_0 then proves that $a_i^{x_0^i}$ is in common for all keys associated with the ith input bit, which is reduced to showing that either the set $\{(g, g^{a_i^{x_0^i}}, g^{c_j}, \tau_{i,j})\}_{j=1}^s$ or the set $\{(g, g^{a_i^{1-x_0^i}}, g^{c_j}, \tau_{i,j})\}_{j=1}^s$ is comprised of DH tuples. Notably, it is sufficient to use a single UC ZK proof for the simpler relation $\mathcal{R}_{\text{DH,OR}}$ since the above statement can be compressed into a compound statement of two DH tuples as follows: P_0 first chooses s random values $\gamma_1, \ldots, \gamma_s \in \mathbb{Z}_p$ and sends them to P_1. Both parties compute $\tilde{g} = \prod_{j=1}^s (g^{c_j})^{\gamma_j}$, $\tilde{\tau} = \prod_{j=1}^s (\tau_{i,j})^{\gamma_j}$, of which P_0 proves that either $(g, g^{a_i^{x_0^i}}, \tilde{g}, \tilde{\tau})$ or $(g, g^{a_i^{1-x_0^i}}, \tilde{g}, \tilde{\tau})$ is a DH tuple. Thus, P_0 invokes $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$ with $\sum_{j=1}^s c_j \gamma_j$ as the witness.*

8. **Circuit evaluation.** *Upon receiving* `Accept` *from $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$, P_1 completes the evaluation of the circuits and sets the majority of these values as the output y.*

9. **Output communication.** *P_1 sends y using an instance of $\mathcal{F}_{\text{SC}}$.*

Informally, to ensure the one-sided security of Π_f^{MAL} we realize the functionalities used in the protocol as follows: **(1)** $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ is realized in **Step 2** using our one-sided batch single choice cut-and-choose OT. This implies the equivocation of P_1's input. **(2)** The statement of $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$ is transferred in **Step 7.(a)** via one sided NCE or a somewhat NCE with $\ell = 2$. Note that in order to obtain a witness equivocal proof for functionality $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$ (invoked in **Step 7.(b)**), it is sufficient to employ a standard static proof realizing this ZK functionality where the prover sends the third message of the proof using a one-sided NCE or a somewhat NCE with $\ell = 2$ (this is due to the fact that we anyway send the statement using a one-sided NCE or a somewhat NCE). Specifically, a statically secure proof is sufficient whenever both the statement and the third message of the (Σ-protocol) proof can be equivocated. This implies the equivocation of P_0's input. Next, we prove

Theorem 6.3 (One-sided malicious) *Let f be a deterministic same-output functionality and assume that the encryption scheme for garbling has indistinguishable encryptions under chosen plaintext attacks, an*

[8] At this point P_0 is committed to all the keys associated with the s circuits.

[9] The exact efficiency of the protocol can be further improved if somewhat NCE with $\ell = 2$ is used instead.

elusive and efficiently verifiable range, and that the DDH and DCR assumptions are hard in the respective groups. Then Protocol Π_f^{MAL} UC realizes $\mathcal{F}_f$ in the presence of one-sided malicious adversaries at a cost of $\mathcal{O}(s|C|)$ private key operations and $\mathcal{O}(s(|C| + (|\mathsf{input}| + |\mathsf{output}|))$ public key operations where s is a statistical parameter that determines the cut-and-choose soundness error.

The DDH+DCR assumptions imply a cut-and-choose OT with constant number of PKE operations for large sender's input spaces. In addition, the original [LP12] protocol is secure under the DDH assumption.

Proof: Our proof is shown in the $(\mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{CCOT}}^{\text{BATCH}}, \mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}})$-hybrid model. Let ADV be a probabilistic polynomial-time malicious adversary attacking Protocol 5 by adaptively corrupting one of the parties. We construct an adversary SIM for the ideal functionality $\mathcal{F}_f$ such that no environment ENV distinguishes with a non-negligible probability whether it is interacting with ADV in the real setting or with SIM in the ideal setting. We recall that SIM interacts with the ideal functionality $\mathcal{F}_f$ and the environment ENV. We refer to the interaction of SIM with $\mathcal{F}_f$ and ENV as the external interaction. The interaction of SIM with the simulated ADV is the internal interaction. We now explain the actions of the simulation for the following corruption cases: **(1)** No corruption takes place; **(2)** Corruption takes place at the outset; **(3)** Corruption takes place between Steps 2 and 3; **(4)** Corruption takes place between Steps 3 and 7 **(5)** Corruption takes place between Steps 7 and 9; **(6)** Corruption takes place at the end. We describe a simulator for all these cases considering the corruption of each party. These cases cover all potential cases of corruption.

No corruption. When no corruption takes place the simulator simulates both P_0 and P_1 as follows:

1. **Garbled circuit construction.** No communication is carried out in this step. SIM internally picks n pairs of random values $(a_1^0, a_1^1), \ldots, (a_n^0, a_n^1) \leftarrow \mathbb{Z}_q \times \mathbb{Z}_p$ and $c_1, \ldots, c_s \leftarrow \mathbb{Z}_q$ (which define the keys for P_0's input). It further picks a pair of random keys corresponding to each input bit of P_1.

2. **Oblivious transfers.** No communication is carried out in this step since an ideal call to $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ is made. SIM internally chooses a random subset $\mathcal{J}$ on the behalf of P_1.

3. **Sending garbled circuits and commitments.** No communication is carried out with respect to the garbled circuit since an ideal call to $\mathcal{F}_{\text{SC}}$ is made. SIM emulates the honest P_0 and sends the values $((g^{a_1^0}, g^{a_1^1}), \ldots, (g^{a_n^0}, g^{a_n^1}), (g^{c_1}, \ldots, g^{c_s}))$.

4. **Revealing $\mathcal{J}$.** SIM emulates the honest P_1 and sends $\mathcal{J}$ together with the pair of keys that are associated with P_1's first input bit in each check circuit, that is, for every GC_i for which $i \in \mathcal{J}$.

5. **Decommitting $\mathbf{P_0}$'s input keys.** SIM emulates the honest P_0 and sends c_j for all $j \in \mathcal{J}$.

6. **Verifying the check circuits.** No communication is carried out in this step.

7. **Sending the garbled inputs for the evaluation circuits.** No communication is carried out in this step as it involves calling the two ideal functionalities $\mathcal{F}_{\text{SC}}$ and $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$.

8. **Circuit evaluation.** No communication is carried out in this step.

9. **Output communication.** No communication is carried out in this step since an ideal call to $\mathcal{F}_{\text{SC}}$ is made.

Note that the simulated the hybrid executions are statistically close. Specifically, communication takes place only in Steps 3,4, and 5. This communication is independent of the parties' inputs and outputs. Therefore, security follows trivially for this case.

Corruption at the outset of the protocol execution.

- P_0 **is corrupted.** In Step 2 SIM emulates $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$, receiving the input of ADV for this functionality. Namely, the key pairs that correspond to all the input wires of the circuit. In Step 3, SIM receives s garbled circuits from ADV on the behalf of $\mathcal{F}_{\text{SC}}$ and the commitments to the key pairs that correspond to ADV's input. In Step 4, the simulator sends a random subset $\mathcal{J}$ and the pairs of the input keys that are associated with P_1's first input wire in each check circuit GC_i for which $i \in \mathcal{J}$ (recall that SIM knows all the input keys in Step 2). In Step 5 the simulator receives c_j for all $j \in \mathcal{J}$ from ADV. In Step 6 the simulator verifies the check circuits just as the honest P_1 would do, and aborts if a problem is detected. In Step 7 the simulator receives the keys $\{\tau_{i,j}\}_{i\in\{1,\ldots,n\},j\notin\mathcal{J}}$ as well as the statement and its corresponding witness for the ideal call of $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$. If the witness is verified correctly then SIM extracts ADV's input as follows. For some fixed $j \notin \mathcal{J}$, SIM extracts the ith bit of ADV's input x_0' by applying the hash function H on $\tau_{i,j}$ and then checking whether the result matches either $H(g^{a_i^0 c_j})$ which implies that $x_0'^i = 0$, or $H(g^{a_i^1 c_j})$ which implies that $x_0'^i = 1$. SIM sends x_0' to $\mathcal{F}_f$ and receives the output y'. Next it sends y' to ADV on behalf of $\mathcal{F}_{\text{SC}}$ in Step 9 and concludes the simulation.

- P_1 **is corrupted.** SIM mimics the [LP12] simulation that is designed the static case, building fake circuits that always compute $y' = f(x_0, x_1')$ for x_1' the input of ADV. Specifically, in Step 1 SIM emulates P_0 as in the no corruption case, i.e., it picks key pairs that correspond to the input wires of both parties. In Step 2, SIM emulates $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ and receives ADV's input x_1' and the subset $\mathcal{J}$ and returns the adversary's output. It then sends x_1' to $\mathcal{F}_f$ and receives back the output y'. Then in Step 3 SIM constructs $s/2$ correct circuits that correspond to indices from $\mathcal{J}$ and additional $s/2$ fake circuits that always output y' irrespective of the input. It then sends this set of correct and fake garbled circuits to ADV on behalf of $\mathcal{F}_{\text{SC}}$. It further sends the commitments to the correct key pairs that correspond to the inputs wires of P_0. In Step 4 the simulator receives $\mathcal{J}$ and the key pairs that correspond to the first input bit of ADV for all the circuits GC_i such that $i \in \mathcal{J}$. If ADV sends invalid keys, or the set $\mathcal{J}$ does not match the set SIM obtains in Step 2 during the simulation of the ideal cut-and-choose OT, then SIM aborts. In Step 5 SIM correctly decommits c_j such that $j \in \mathcal{J}$. In Step 7 SIM plays the role of the honest P_0 with input 0^n. Following that, SIM emulates the ideal functionality for $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$ and sends `Accept` to ADV. In Step 9 the simulator receives output y'' from ADV via $\mathcal{F}_{\text{SC}}$. If y'' is not equal y' then SIM aborts.

  We note that in the $(\mathcal{F}_{\text{SC}}, \mathcal{F}_{\text{CCOT}}^{\text{BATCH}}, \mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}})$-hybrid execution corruption at the onset of the protocol is proven similarly to the static proof provided in [LP12]. Specifically, the only difference between the proofs is that some parts of the communication of our protocol are transferred via $\mathcal{F}_{\text{SC}}$. However the information sent using $\mathcal{F}_{\text{SC}}$ is exactly what is sent by the simulator of [LP12] on clear. When corruption takes place later in the protocol, the functionalities allow to pretend that computation done at the corrupted parties end was carried out correctly and consistently with the input and communication done so far. The simulator simulates the honest party exactly the way it does so in the no corruption case. So the security for the cases where corruption occurs later in the protocol reduces to the security of the case where corruption happens at the outset.

Corruption between Steps 2 and 3.

- P_0 **is corrupted.** SIM simulates P_0 and P_1 in Steps 1 and 2 as in the no corruption case. Then, whenever P_0 is corrupted SIM constructs s garbled circuits correctly using the keys that were picked in Step 1. It then discloses the circuits and the input keys to the adversary. It further explains the internal state of P_0 so that its input to the OT functionality $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ is consistent with the above input keys. Finally, SIM emulates the honest P_1 exactly as in the case when P_0 is corrupted at the outset.

Note that the adversary's views in case P_0 is corrupted at the outset of the protocol execution and in case P_0 is corrupted between Steps 2 and 3 is identical since P_0 does not use its input yet at this point, and thus the simulation in both corruption cases is essentially the same.

- P_1 **is corrupted.** SIM simulates P_0 and P_1 in Steps 1 and 2 as in the no corruption case. Then, whenever P_1 is corrupted SIM receives its input and output (x_1, y). SIM explains the internal state of P_1 such that the input of P_1 to the $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ functionality is x_1 and a random subset $\mathcal{J}$. It then discloses the keys that P_1 should have received upon entering x_1 and $\mathcal{J}$ to $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ in Step 2. SIM completes the simulation by simulating P_0 exactly as in the case when P_1 is corrupted at the outset.

  Note that in the hybrid setting (where the OT executions are computed using an ideal call to the cut-and-choose OT), the adversary's views in case P_1 is corrupted at the outset of the protocol execution and in case P_1 is corrupted between Steps 2 and 3 is identical since the simulator can equivocate P_1's input to $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$. This implies that the adversary's internal state until corruption takes place is distributed as the internal state of the honest P_1, and the simulation can proceed exactly as in the static corruption case from [LP12].

Corruption between Steps 3 and 7.

- P_0 **is corrupted.** SIM simulates P_0 and P_1 in Step 1-3 as in the no corruption case. Then, whenever P_0 is corrupted SIM constructs s garbled circuits correctly using the keys that were picked in Step 1. It then discloses the circuits and the input keys to the adversary. It further explains the internal state of P_0 so that its inputs to $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ in Step 2 and to $\mathcal{F}_{\text{SC}}$ in Step 3 are consistent with garbled circuits constructed above. SIM completes the simulation by simulating P_1 exactly as in the case when P_0 is corrupted at the outset.

  Note that in this corruption case P_0 does not use its input yet and thus the indistinguishability argument is as in the prior corruption case

- P_1 **is corrupted.** SIM simulates P_0 and P_1 in Steps 1-3 as in the no corruption case. Then, whenever P_1 is corrupted SIM receives its input and output (x_1, y). SIM explains the internal state of P_1 such that the input of P_1 to the $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ functionality is x_1 and a random subset $\mathcal{J}$. It further discloses the input keys that P_1 should have received upon entering inputs x_1 and $\mathcal{J}$ to $\mathcal{F}_{\text{CCOT}}^{\text{BATCH}}$ in Step 2. Next, SIM constructs $s/2$ correct circuits that correspond to indices from $\mathcal{J}$ and $s/2$ additional fake circuits that always output y. It then explains the internal state of P_1 such that P_1 has received the garbled circuits constructed as above (so that the correct circuits are the check circuits). SIM concludes the simulation by simulating P_0 exactly as in the case when P_1 is corrupted at the outset.

  The adversary's view is distributed as in the static corruption case relying on security of $\mathcal{F}_{\text{SC}}$ that transfers the garbled circuits. In the static corruption case, SIM sends a set of garbled circuits containing $s/2$ fake circuits and $s/2$ good circuits on behalf of $\mathcal{F}_{\text{SC}}$. No equivocation is required since SIM knows P_1 is corrupted at the outset. In the current scenario, SIM equivocates and explains to ADV that such a set is delivered by $\mathcal{F}_{\text{SC}}$. In $\mathcal{F}_{\text{SC}}$-hybrid model, the security reduces to the security of static corruption case.

Corruption between Steps 7 and 9.

- P_0 **is corrupted.** SIM simulates P_0 and P_1 in Steps 1-7 as in the no corruption case. Then, whenever P_0 is corrupted SIM receives its input and output (x_0, y) and explains the internal state of P_0 until Step 7 as in the prior corruption case. Next, SIM explains the inputs of P_0 to the ideal call of $\mathcal{F}_{\text{SC}}$ in Step

7 so that they are consistent with the input x_0. Namely, the ith input wire of P_0 in the jth evaluation circuit is explained as $g^{a_i^{x_0^i} c_j}$. It further explains the witness to $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\text{DH,OR}}}$ correctly as $\sum_{j=1}^{s} c_j \gamma_j$ (recall that $\gamma_1, \ldots, \gamma_s$ are random elements that enable to combine s proofs into a single proof). The simulator concludes by simulating the honest P_1 exactly as in the case when P_0 is corrupted at the outset.

In the hybrid setting, indistinguishability follows easily as above since the privacy of P_1 follows from the privacy of the OT protocol. Moreover, all the messages sent by P_0 are emulated as being sent by the honest P_0 therefore security is reduced to the static corruption case.

- P_1 **is corrupted.** SIM simulates P_0 and P_1 in Steps 1-7 as in the no corruption case. Then, whenever P_1 is corrupted SIM receives its input and output (x_1, y) and explains the internal state of P_1 until Step 7 as in the prior corruption case. Next, SIM explains the internal state of P_1 in Step 7 using the garbled inputs keys of P_0 that are consistent with 0^n as the keys received from $\mathcal{F}_{\text{SC}}$. Namely, the ith input wire of P_0 in the jth evaluation circuit is explained as $g^{a_i^0 c_j}$. Finally, the simulator explains the message `Accept` as being received from the ZK functionality and concludes the simulation as above.

  The proof for this corruption case is identical to the prior case since P_1 already used its input in the OT phase, and the additional simulated messages only correspond to the garbled inputs for the check circuits transferred from P_0.

Post execution corruption.

- P_0 **is corrupted.** SIM simulates P_0 and P_1 during all the steps as in the no corruption case. The internal state of P_0 is explained as above except that in Step 9 SIM explains y as the output received via the ideal call of $\mathcal{F}_{\text{SC}}$.
- P_1 **is corrupted.** SIM simulates P_0 and P_1 during all the steps as in the no corruption case. The internal state of P_0 is explained as above except that in Step 9 SIM explains y as the input entered to the ideal call of $\mathcal{F}_{\text{SC}}$.

Security in the hybrid model is proven exactly as above since the non-committing channel can be equivocated to the correct output value. This concludes the proof. ■

7 Efficient Statically Secure and Witness Equivocal UC ZK PoKs

We present two results in this section. First, we show a technique for generating efficient statically secure UC ZK PoK for various Σ-protocols. Our protocols take a new approach where the prover commits to an additional transcript which, in turn, enables witness extraction without using rewinding. Our instantiations imply UC ZK PoK constructions that incur constant overhead with a negligible soundness error.

Next, we show how to generate efficient *witness equivocal* UC ZK PoK for compound statements. The additional feature that witness equivocal UC ZK PoK offers over static security is that it allows the simulator to equivocate the simulated proof upon corrupting the prover. In this work, we build witness equivocal UC ZK PoKs for a class of fundamental compound Σ-protocols without relying on NCE. Our approach yields proofs where the simulator knows the witnesses for all sub-statements (but not which one is the real witness). This notion is weaker than the notion of one-sided UC ZK PoK where the simulator is required to simulate the proof obliviously of the witness and later prove consistency with respect to the real witness. Our protocols are constant rounds and overhead, with a negligible soundness error.

We briefly describe our technique for generating efficient UC ZK PoK for Σ-protocols. Recall that in order to obtain a UC secure ZK PoK for a Σ-protocol it suffices to build a straight line simulator and witness extractor in the CRS model. A straight line simulator can be obtained by using standard techniques of committing the challenge of the verifier at the onset of the proof using UC commitments [DN02]. In what follows, we will focus on designing straight line extractors. We begin with a generalization of our UC ZK PoKs for Σ-protocols for relations of the form $\mathcal{R}_\Gamma = \left\{((\tilde{\mathbb{G}}, \tilde{\mathbb{H}}, y), x) |\ y = \Gamma(x)\right\}$ defined with respect to a one-way homomorphic mapping $\Gamma : \tilde{\mathbb{G}} \to \tilde{\mathbb{H}}$ from a source group $(\tilde{\mathbb{G}}, \oplus)$ to a target group $(\tilde{\mathbb{H}}, \odot)$. (Where Γ is homomorphic if $\Gamma(x_0 \oplus x_1) = \Gamma(x_0) \odot \Gamma(x_1)$).[10] Loosely speaking, given a Σ-protocol Π_Γ for $\mathcal{R}_\Gamma$ we define a new proof Π'_Γ by instructing the prover to send two responses z, z' to a pair of distinct challenges c, c' queried by the verifier. Specifically, the former response z is sent on clear and publicly verified as specified in Π_Γ, whereas the latter response z' is encrypted using a homomorphic PKE with plaintext space $\tilde{\mathbb{G}}$. Moreover, the validity of z' is carried out by a (Σ-protocol) UC ZK proof Π_Σ of consistency. Observe that an extractor can be easily constructed for Π'_Γ by placing a public key for the homomorphic PKE in the CRS, of which the extractor knows the corresponding secret key. Our technique also generalizes to proofs for compound statements [CDS94]. Clearly, the efficiency of our new proof depends heavily on the overhead of Π_Σ; we discuss two implementations below. For simplicity, we describe our protocols for a honest verifier. Standard techniques can be used to achieve full security.

7.1 Efficient Statically Secure UC ZK PoK for Σ-Protocols

DL-based UC secure ZK PoK. We continue with illustrating our technique on the Σ-protocol for proving the knowledge of a discrete logarithm in a prime order group $\mathbb{G}$. We instantiate $(\tilde{\mathbb{G}}, \oplus)$ with $(\mathbb{Z}_p, +)$ for operation $+$ denoting addition in $\mathbb{Z}_p$, and $(\tilde{\mathbb{H}}, \odot)$ with $(\mathbb{G}, \cdot)$ for operation $\cdot$ denoting multiplication in $\mathbb{G}$. Furthermore, the one-way group homomorphism is defined by $\Gamma(x) = \mathrm{EXP}(x) = g^x$ where g is a generator of $\mathbb{G}$ and induces the relation

$$\underline{\mathcal{R}_{\mathrm{DL}}} = \{((\mathbb{G}, g, h), x) |\ h = g^x\}\,.$$

We first apply our technique on the Σ-protocol due to [Sch89] and instantiate the additively homomorphic PKE within Π_Σ with Paillier [Pai99], that is defined by $\mathsf{Enc}_{\mathsf{PK}}(x; r) = (1+N)^x \cdot r^N \bmod N^2$ where N is an RSA composite. Formally,

Protocol 6 (UC ZK PoK of DL (Π_{DL}))

- **CRS:** *A public key* PK *for Paillier PKE.*
- **Joint statement:** *The description of a group $\mathbb{G}$ of prime order p and a generator g, and the public statement h.*
- **Auxiliary input for the prover:** $x \in \mathbb{Z}_p$ *such that* $h = g^x$.
- **The Protocol:**
  1. *Prover* P *picks a random $r \leftarrow \mathbb{Z}_p$ and sends the verifier $a = g^r$.*
  2. *Verifier* V *returns random challenges $c, c' \leftarrow \mathbb{Z}_p$.*
  3. P *sends $z \leftarrow r + cx \bmod p$ and encrypts $z' \leftarrow r + c'x \bmod p$ using* PK*, generating ciphertext e.* P *sends z and e to* V *and proves in UC ZK that the plaintext of e and the discrete log of $ah^{c'}$ are the same.*
  4. V *accepts if the ZK proof is verified correctly and $g^z = ah^c$.*

[10] This notation covers many basic relations such as discrete logarithm and quadratic residuosity.

The proof used in Step 3 is obtained from a Σ-protocol for the following relation

$$\mathcal{R}_1 = \left\{((N, \mathrm{PK}, e, \mathbb{G}, g, h), (x, \alpha))|\ e = (1+N)^x \alpha^N \bmod N^2 \wedge h = g^x\right\}.$$

Namely, the goal is to prove consistency of discrete logarithms with respect to two different group orders with generators $(1+N)$ and g, respectively. This can be achieved by combining the proof of knowledge of discrete logarithms over the integers [DJN10] and the proof of plaintext knowledge for Pailler (we note that [DJN10] shows a proof for consistent exponents, i.e., for DH tuples, but the same proof technique works here as well). Namely, the prover selects at random y and β, computes $e' = (1+N)^y \beta^N \bmod N^2$ and $h' = g^y$ and sends e', h' to the verifier, who returns a random challenge $c \in \mathbb{Z}_p$. The prover then replies with $z = y + cx$ (over integers) and $\gamma = \alpha\beta^c \bmod N$. However, to ensure the privacy of x within $y + cx$, y must be chosen so that its length is at least $|c| + |x| + \kappa$, where κ is a statistical parameter. The verifier then accepts if $(1+N)^z \gamma^N \bmod N^2 = e' e^c \bmod N^2$ and $g^z = h' h^c$. We further note that the above proof requires a special care since it must ensure that the exact same value x is encrypted under Paillier rather than $x + ip$, for p the order of $\mathbb{G}$ and i some integer. Nevertheless, an extractor that decrypts and learns $x + ip$ can still find x. Thus our extractor first learns z' by decrypting the Paillier ciphertext and then extracts x from z and z'. Finally, the above Σ-protocol and the PoK presented in Protocol 6 are proving in the UC framework using standard techniques of committing the verifier's challenge at the beginning of the proof using UC commitment scheme [GK96]. We denote the UC ZK PoK for $\mathcal{R}_{\mathrm{DL}}$ by Π_{DL}.

Proposition 7.1 *Assume that the DCR and DDH assumptions are hard in the respective groups. Then Protocol 6 UC realizes $\mathcal{F}_{\mathrm{ZKPoK}}^{R_{\mathrm{DL}}}$ with a negligible soundness error and constant overhead.*

Proof Sketch: Informally, the proof follows by having the extractor pick a pair of keys $(\mathrm{PK}, \mathrm{SK})$ and place PK in the CRS. Then, whenever receiving ciphertext e from the prover, the extractor decrypts it using SK and extracts the witness from z and z'. From the security of the ZK proof of discrete logarithms consistency, it holds that the prover must encrypt with overwhelming probability the correct value of z'. This implies that the extractor can extracts the witness correctly. Furthermore, the zero-knowledge property is implied by the zero-knowledge of the original proof for $\mathcal{R}_{\mathrm{DL}}$ and the ZK proof of consistency. Specifically, a simulator for Π_{DL} will compute the first message and z as in the original simulation of [Sch89]. It then obliviously samples a ciphertext e rather than encrypting the real message z', and employs the simulator for the ZK proof of consistency Π_{Σ} (which is a proof for relation $\mathcal{R}_1$ in Protocol 6). Note that the simulator can also encrypt an arbitrary value instead of obliviously sampling the ciphertext. Nevertheless, we stick to the former description since it simplifies the description of our protocol for the adaptive setting. It is simple to verify that the simulated view is computationally indistinguishable from a real view since the only difference is relative to ciphertext e and the simulated view of Π_{Σ}. Finally, the overhead of the protocol is constant since the overhead of the internal ZK proof is constant. ■

Consistency of discrete logarithms. Next, we consider a UC PoK for the following relation

$$\underline{\mathcal{R}_{\mathrm{DH}}} = \{((\mathbb{G}, g_0, g_1, h_0, h_1), x)|\ h_0 = g_0^x \wedge h_1 = g_1^x\}.$$

Here $(\tilde{\mathbb{G}}, \oplus)$ is instantiated with $(\mathbb{Z}_p, +)$ and $(\tilde{\mathbb{H}}, \odot)$ with $(\mathbb{G} \times \mathbb{G}, \cdot)$. We further define by $\Gamma(x) = (g_0^x, g_1^x)$ where g_0, g_1 are two generators in $\mathbb{G}$. As above, we use Paillier PKE to encrypt the second reply of the prover. The proof is an immediate extension of the Protocol 6 and the standard Σ-protocol for $\mathcal{R}_{\mathrm{DH}}$. The underlying ZK proof for proving the correctness of the plaintext encrypted by the prover is an extension of the proof for the relation used in Protocol 6. Specifically, the relation for the underlying ZK proof is:

$$\mathcal{R}_2 = \left\{((N, \mathrm{PK}, e, \mathbb{G}, g_0, g_1, h_0, h_1), (x, \alpha))|\ e = (1+N)^x \alpha^N \bmod N^2 \wedge h_0 = g_0^x \wedge h_1 = g_1^x\right\}.$$

UC PoKs for Nth root and quadratic residuosity. The proof we consider here is a proof of knowledge of an Nth root formally defined by,

$$\underline{\mathcal{R}_{\text{NR}}} = \left\{((u, N), v) | \; u = v^N \bmod N^2\right\}.$$

We instantiate $(\tilde{\mathbb{G}}, \oplus)$ with $(\mathbb{Z}_N^*, \cdot)$ and $(\tilde{\mathbb{H}}, \odot)$ with $(\mathbb{Z}_{N^2}^*, \cdot)$, where multiplication is computed in the respective group. Furthermore, $\Gamma(x) = x^N \bmod N^2$. Note that in order to encrypt the message of the prover we need to use a multiplicative PKE, and we therefore consider a variant of El Gamal PKE that operates in $\mathbb{Z}_N^*$ for a message space $\mathbb{QR}_N$ where $N = pq$ is an RSA composite such that $(p-1)/2$ and $(q-1)/2$ are relatively primes. Specifically, encrypting a message $m \in \mathbb{QR}_N$ is computed by $(e_1, e_2) = (g^r \bmod N, m \cdot h^r \bmod N)$ where g is a random element in $\mathbb{QR}_N$, $h = g^x \bmod N$ with a secret key $x \in \mathbb{Z}_{\phi(N)/4}$ and randomness $r \leftarrow \mathbb{Z}_{\phi(N)/4}$. The security of this scheme is based on the composite DDH assumption [DJ03] in $\mathbb{Z}_N^*$ (defined below). In the proof below, the verifier is required to ensure that $z'^N = au^{2c'}$. This is achieved by raising the ciphertext $e = (e_1, e_2)$ encrypting z' to the power of N component-wise modulo N^2, and then have the prover prove that $e_1^N, e_2^N / au^{2c'}$ is a Diffie-Hellman tuple in $\mathbb{Z}_{N^2}^*$. Such a ZK proof is provided in [DJN10]. Namely, we use $2c'$ instead of c' to ensure that z' is in $\mathbb{QR}_N$.

The Composite DDH Assumption. Let $N = pq$ be an RSA modulus and g is an element of $\mathbb{QR}_N$ the group of squares in $\mathbb{Z}_N^*$. Then values a and b are chosen uniformly at random in $\mathbb{Z}_{\phi(N)/4}$ and the value y is either random in $\mathbb{QR}_N$ or satisfies $y = g^{ab} \bmod N$. Finally, the assumption asserts that for any polynomial-time algorithm, the advantage in guessing which way y was sampled when given $(N, g, g^a \bmod N, g^b \bmod N, y)$ is negligibly close to $1/2$.

Protocol 7 (UC ZK PoK for $\mathcal{R}_{\text{NR}}$ (Π_{NR}))

- **Joint statement:** $u \in \mathbb{Z}_{N^2}^*$.
- **Auxiliary input for the prover:** $v \in \mathbb{Z}_N^*$ *such that* $u = v^N \bmod N^2$.
- **CRS:** *A composite N and a public key* $\mathsf{PK} = (\mathbb{G}, h = g^x)$ *for El Gamal PKE in* $\mathbb{Z}_N^*$.
- **The Protocol:**

  1. *Prover* P *picks a random* $r' \leftarrow \mathbb{Z}_N^*$ *and sends verifier* V *the value* a *where* $a = r^N \bmod N^2$ *where* $r \leftarrow r'^2 \bmod N$.
  2. V *returns random challenges* $c, c' \leftarrow \mathbb{Z}_N^*$.
  3. P *sets* $z \leftarrow rv^c \bmod N$ *and* $z' \leftarrow rv^{2c'} \bmod N$*, and encrypts z' using* PK *(note that* $z' \in \mathbb{QR}_N$*). Denote the generated ciphertext by* $e = (e_1, e_2)$. P *sends* V *values z and e and proves in UC ZK that the decryption of* $e^N \bmod N^2$ *corresponds to* $au^{2c'} \bmod N^2$*. That is,* P *proves that* $(\mathbb{Z}_{N^2}^*, g, h, e_1^N, e_2^N / au^{2c'})$ *is a Diffie-Hellman tuple in* $\mathbb{Z}_{N^2}^*$ *using the proof from [DJN10].*
  4. V *accepts if it accepts the ZK proof and if* $z^N = au^c \bmod N^2$.

Proposition 7.2 *Assume that the DCR and composite DDH assumptions are hard in the respective groups. Then Protocol 7 UC realizes $\mathcal{F}_{\text{ZKPoK}}^{R_{\text{NR}}}$ with a negligible soundness error and constant overhead.*

Finally, we consider a proof of knowledge of a square root that is formally defined by,

$$\underline{\mathcal{R}_{\text{QR}}} = \left\{((u, N), v) | \; u = v^2 \bmod N\right\}.$$

We instantiate $(\tilde{\mathbb{G}}, \oplus)$ with $(\mathbb{Z}_N^*, \cdot)$ and $(\tilde{\mathbb{H}}, \odot)$ with $(\mathbb{QR}_N, \cdot)$, where multiplication is computed in the respective groups. Furthermore, $\Gamma(x) = x^2 \bmod N$. Following a similar technique used for the ZK PoK of $\mathcal{R}_{\text{NR}}$ we design a proof for $\mathcal{F}_{\text{ZKPoK}}^{R_{\text{QR}}}$ based on the QR and composite DDH assumptions.

Proposition 7.3 *Assume that the QR and composite DDH assumptions are hard in the respective groups. Then there exists a protocol Π_{QR} that UC realizes $\mathcal{F}_{\text{ZKPoK}}^{R_{\text{QR}}}$ with a negligible soundness error and constant overhead.*

7.2 Efficient Witness Equivocal UC ZK PoK for Compound Statements

The proof technique discussed above cannot be used in the adaptive setting since it does not allow witness equivocation. Fortunately, in this work we only need to consider compound statements for which the simulator knows all witnesses but does not know which one to use during the simulation, since this choice depends on the input of the real prover. In compound statements for Σ-protocols the prover separates the challenge c that is given by the verifier into two values; c_1 and c_2 such that $c = c_1 \oplus c_2$. Assume w.l.o.g. that the prover does not have a witness for the first statement, then it always chooses c_1 in which it knows how to complete the proof (similarly to what the simulator does), and uses its witness for the other statement to complete the second proof on a given challenge c_2. Note that the verifier cannot distinguish whether the prover knows the first or the second witness (or both); see [CDS94] for more details.

This type of compound statements generalizes to s sub-statements for which the prover proves the knowledge of witnesses of some subset. For this general case, [CDS94] suggested to split the challenge using a perfect secret sharing scheme, e.g. Shamir's secret sharing scheme. First, the sub-statements are divided into two sets such that the prover knows the witnesses for the statements in set $\mathcal{J}$, but not the witnesses for the statements in $\bar{\mathcal{J}}$. The prover then creates challenges c_i for the sub-statements in $\bar{\mathcal{J}}$. Finally, the set $\{c_j\}_{j\in\bar{\mathcal{J}}}$ and the verifier's challenge c are respectively interpreted as $|\bar{\mathcal{J}}|$ shares and the secret relative to a secret sharing scheme, with s participants and a threshold $|\bar{\mathcal{J}}|+1$. Thus, the values c and $\{c_j\}_{j\in\bar{\mathcal{J}}}$ completely define all the s shares. In the proof, the prover runs the honest verifier zero-knowledge simulator for the sub-statements in $\bar{\mathcal{J}}$ using challenges $\{c_j\}_{j\in\bar{\mathcal{J}}}$. For the sub-statements in $\mathcal{J}$ it defines the challenges as defined by the shares $\{c_j\}_{j\in\bar{\mathcal{J}}}$ for the secret c and generates a response using its witnesses. The prover also sends the s shares of c to the verifier who checks that indeed the shares define c with threshold $|\bar{\mathcal{J}}|+1$.

Nevertheless, allowing the simulator to use all potential witnesses in the adaptive setting is not equivocal as well, since an adversary that corrupts the prover can detect a simulated execution by simply computing multiple witnesses when given the prover's internal state. In order to resolve this difficulty we instruct the prover to obliviously sample the ciphertexts for the statements it does not know the witness for within the internal proof of consistency Π_Σ, i.e., sampling a ciphertext without knowing the corresponding plaintext. This property holds with respect to both homomorphic PKEs used in our proofs above in order to encrypt the response for the second challenge. We formally describe our protocol for a compound statement that is defined relative to relations $\mathcal{R}_0$ and $\mathcal{R}_1$ (following the ideas from [CDS94]), a general description for any number of statements follows easily. We denote by Π_0 and Π_1 the respective UC ZK PoK Σ-protocols for $\mathcal{R}_0$ and $\mathcal{R}_1$.

Protocol 8 (UC ZK PoK for $\mathcal{R}_0$ and $\mathcal{R}_1$ (Π_{OR}))

- **Joint statement:** $x_0 \in L_0$ *and* $x_1 \in L_1$.
- **Auxiliary input for the prover:** ω_i *for* $i \in \{0,1\}$ *such that* $(x_i, \omega_i) \in \mathcal{R}_i$.
- **CRS:** *A CRS for* Π_Σ.
- **The Protocol:**

  1. *Prover* P *computes the first message as follows.*
     It first invokes the simulator SIM_{1-i} *for* Π_{1-i} *on* x_{1-i} *and arbitrary challenge* $\tilde{c}$*, and obtains message* m_{1-i}.[11]
     It then invokes the prover P_i *for* Π_i *on* (x_i, ω_i) *and obtains message* m_i.
     P *sends messages* (m_0, m_1) *to the verifier.*
  2. V *returns a random challenge* c *from the appropriate space.*

[11] The simulator actually returns the entire view for the proof but for simplicity we only consider the first message in this step.

3. P *computes the third message as follows.*
   It first invokes simulator SIM_{1-i} *on* x_{1-i} *and arbitrary challenge* $\tilde{c}$*, and receives a third message for* Π_{1-i} *which includes message* m'_{1-i}*, an obliviously sampled ciphertext* e_{1-i} *and a simulated view* $view_{1-i}$ *for the ZK proof* Π_Σ*.*
   It then invokes P_i *on* $(x_i, \omega_i), m_i, c \oplus \tilde{c}$ *and receives a third message for* Π_{1-i} *which includes message* m'_{1-i}*, a ciphertext* e_i *and a real view* $view_i$ *for the ZK proof* Π_Σ*.*
   P *sends the verifier* $((m'_{1-i}, e_{1-i}, view_{1-i}, \tilde{c}), (m'_i, e_i, view_i, c \oplus \tilde{c}))$*.*
4. V *invokes the verifiers for* Π_0 *and* Π_1 *and accepts if they both accept and if the two challenges received from the prover are shares of* c*.*

We conclude with the following theorem.

Theorem 7.1 *Assume the existence of homomorphic PKE with respect to a group* $\mathbb{H}$ *and operation* $\odot$ *that supports oblivious and invertible sampling of ciphertexts, and that* $\Gamma : \tilde{\mathbb{G}} \to \tilde{\mathbb{H}}$ *is a one-way group homomorphism. Then, Protocol 8 is a witness equivocal UC ZK* Σ*-protocol for relations* $\mathcal{R}_0$ *and* $\mathcal{R}_1$ *with a negligible soundness error and constant overhead.*

Proof Sketch: Proving PoK follows easily using the trapdoor from the CRS. We now prove that the protocol is ZK. Note that standard simulation follows from the ZK property of each sub-protocol and the [CDS94] proof. We recall next that our protocols only consider simulators that know both witnesses ω_0 and ω_1, but do not know which one is used by the real prover. Simulation in this case is trivial since the simulator simply uses its two witnesses. By the IND-CPA security of the homomorphic encryption scheme and the security of Π_{1-i}, the simulated view (when using two witnesses) and the real view (when using only one witness) are computationally indistinguishable. We now show that the protocol is witness equivocal. Recall that this property implies that the simulator must explain the internal state of the simulated prover with respect to the real prover's state. Say the real prover knows ω_i, then the view for Π_i can be easily explained as if the real prover generated it since the simulator used ω_i in its simulation. In addition, the simulated proof for x_{1-i} differs from the real view by (1) honestly encrypting message z' rather than obliviously sampling the ciphertext and (2) running the real prover for Π_i rather than the simulated one. Witness equivocal follows from the simulatability of the PKE and the fact that the real view of Π_i can be explained as simulated. Namely, the simulator for Protocol 8 can claim that the honestly generated ciphertext was obliviously sampled, and that the real view generated for Π_Σ is a simulated view (since Π_Σ is a Σ-protocol and view $view_i$ is identically distributed to the real view; see Definition 2.13). ■

We denote by $\Pi_{\Gamma,\text{OR}}$ a compound proof where both statements are in L_Γ, and by $\mathcal{F}_{\text{ZKPoK}}^{\mathcal{R}_{\Gamma,\text{OR}}}$ the ideal functionality for $\Pi_{\Gamma,\text{OR}}$. We denote by $\Pi_{\text{DH},\text{OR}(s)}$ a proof for which the statement is a combination of two sub-statements, each contains s tuples. Specifically, we consider a proof of knowledge of which one of the two sets comprised from DH tuples. We also consider a proof $\Pi_{\Gamma,\text{COMP}(s,t)}$ where the statement consists of s sub-statements of L_Γ and the prover proves the knowledge of t sub-statements out of s (for some $t < s$). We make use of these protocols in Section 6.3.

References

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

[Bea97] Donald Beaver. Plug and play encryption. In *CRYPTO*, pages 75–89, 1997.

[Bea98] Donald Beaver. Adaptively secure oblivious transfer. In *ASIACRYPT*, pages 300–314, 1998.

[BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In *EUROCRYPT*, pages 307–323, 1992.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT*, pages 1–35, 2009.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

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

[CDD+04] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *J. Cryptology*, 17(3):153–207, 2004.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[CDSMW09a] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In *ASIACRYPT*, pages 287–302, 2009.

[CDSMW09b] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In *TCC*, pages 387–402, 2009.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CHK05] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-secure, non-interactive public-key encryption. In *TCC*, pages 150–168, 2005.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, 2002.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.

[DGH+04] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the cbc, cascade and hmac modes. In *CRYPTO*, pages 494–510, 2004.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

[DJ03] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP*, pages 350–364, 2003.

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*, pages 432–450, 2000.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO*, pages 581–596, 2002.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, pages 247–264, 2003.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[FHKW10] Serge Fehr, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Encryption schemes secure against chosen-ciphertext selective opening attacks. In *EUROCRYPT*, pages 381–402, 2010.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *J. Cryptology*, 9(3):167–190, 1996.

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

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[GS12] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In *CRYPTO*, pages 105–123, 2012.

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In *CRYPTO*, pages 505–523, 2009.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[JC79] Mark N. Wegman J.Lawrence Carter. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[JL00] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *EUROCRYPT*, pages 221–242, 2000.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO*, pages 335–354, 2004.

[KTZ13] Jonathan Katz, Aishwarya Thiruvengadam, and Hong-Sheng Zhou. Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In *Public Key Cryptography*, pages 14–31, 2013.

[Lin09] Yehuda Lindell. Adaptively secure two-party computation with erasures. In *CT-RSA*, pages 117–132, 2009.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[LP09] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LP12] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, pages 111–126, 2002.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

[NR95] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of psuedo-random functions. In *FOCS*, pages 170–181, 1995.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[WW06] Stefan Wolf and Jürg Wullschleger. Oblivious transfer is symmetric. In *EUROCRYPT*, pages 222–232, 2006.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

A The DDH-Based OT from [PVW08]

In this section we recall the DDH-based PVW OT construction [PVW08]. We begin with their DDH-based dual-mode PKE Π_{DUAL} that is specified by algorithms $(\mathsf{SetupMessy}, \mathsf{SetupDecryption}, \mathsf{dGen}, \mathsf{dEnc}, \mathsf{dDec}, \mathsf{FindBranch}, \mathsf{TrapKeyGen})$ described below.

- $\mathsf{SetupMessy}$ and $\mathsf{SetupDecryption}$ are two algorithms that generate the system parameters in a messy and decryption mode, respectively. Both choose $\mathbb{G} = \mathbb{G}_{g,p}$ specified by a generator g and its prime order p. Specifically,

  $\mathsf{SetupMessy}(1^n)$: Choose (g_0, g_1, h_0, h_1) such that g_0, g_1 are random generators in $\mathbb{G}$ and $h_i = g_i^{y_i}$ for $y_0, y_1 \in \mathbb{Z}_p$. The CRS is (g_0, g_1, h_0, h_1) and the decryption trapdoor t is (y_0, y_1).

  $\mathsf{SetupDecryption}(1^n)$: Choose (g_0, g_1, h_0, h_1) such that g_0 is a random generator in $\mathbb{G}$ and $g_1 = g_0^x$ for $x \in \mathbb{Z}_p$ and $h_i = g_i^y$ for $y \in \mathbb{Z}_p$. The CRS is (g_0, g_1, h_0, h_1) and the decryption trapdoor t is x.

- dGen is the key generation algorithm that takes a bit α and the CRS as input. If $\alpha = 0$, then it generates left public and secret key pair. Otherwise, it creates right public and secret key pair. Specifically,

  $\mathsf{dGen}(\alpha)$: Choose $r \leftarrow \mathbb{Z}_p$ and let $g = g_\alpha^r$ and $h = h_\alpha^r$, then $\mathrm{PK} = (g, h)$ and $\mathrm{SK} = r$.

- dEnc is the encryption algorithm that takes a bit β, a public key $\mathrm{PK} = (g, h)$ and a message m as input. If $\beta = 0$, the it creates the left encryption of m, else it creates the right encryption. Given β, it chooses u, v and computes ciphertext $c \leftarrow ((g_\beta)^u (h_\beta)^v, g^u h^u m)$.

- dDec decrypts a message given a ciphertext and a secret key SK. Namely, given $c = (c_0, c_1)$ the algorithm outputs $c_1/(c_0)^{\mathrm{SK}}$.

- $\mathsf{FindBranch}(t, \mathrm{PK})$ finds whether a given public key (in a messy mode) is a left key or right key given the messy mode trapdoor $t = (y_0, y_1)$, such that $y_0 \neq y_1$. Namely, given a public key $\mathrm{PK} = (g, h)$ created when the CRS is in messy mode the algorithm decides that PK is a left key if $h = g^{y_0}$ and a right key otherwise.

- $\mathsf{TrapKeyGen}(t)$ generates a public key and two secret keys using the decryption mode trapdoor t such that both left and right encryptions under this public key can be decrypted using these secret keys. Namely, given a decryption mode trapdoor t it picks a random $t \in \mathbb{Z}_p$, computes $\mathrm{PK} = (g_0^r, h_0^r)$ and outputs $\mathrm{PK}, r, r/t$.

The complete security definition is found in [PVW08]. We continue with the OT protocol description.

Protocol 9 (UC static malicious OT)

- **Inputs:** *Sender* SEN *has* $x_0, x_1 \in \{0,1\}$ *and receiver* REC *has* $\sigma \in \{0,1\}$.
- **Auxiliary Input:** *A group description* $\mathbb{G} = \mathbb{G}_{g,p}$ *specified by a generator* g *and its prime order* p.
- **CRS:** (g_0, g_1, h_0, h_1) *generated either by* $\mathsf{SetupMessy}$ *or* $\mathsf{SetupDecryption}$.
- **The Protocol:**

1. **Message from the receiver.** REC *sends* SEN PK *where* $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{dGen}(\sigma)$.
2. **Message from the sender.** *Upon receiving two elements* $\mathrm{PK} = (g, h)$, SEN *generates* $c_0 \leftarrow \mathsf{dEnc}_{\mathrm{PK}}(x_0, 0)$ *and* $c_1 \leftarrow \mathsf{dEnc}_{\mathrm{PK}}(x_1, 1)$ *and sends* (c_0, c_1) *to* REC.
3. **Output.** *Upon receiving* $(c_0 = (c_{00}, c_{01}), c_1 = (c_{10}, c_{11}))$, REC *outputs* $\mathsf{dDec}_{\mathrm{SK}}(c_\sigma)$.

Theorem A.1 ([PVW08]) *Protocol 9 securely realizes $\mathcal{F}_{\mathrm{OT}}$ with UC security in the presence of static malicious adversaries.*