Lower Bounds in the Hardware Token Model

Shashank Agrawal* Prabhanjan Ananth† Vipul Goyal‡

Manoj Prabhakaran§ Alon Rosen¶

Abstract

We study the complexity of secure computation in the tamper-proof hardware token model. Our main focus is on non-interactive unconditional two-party computation using bit-OT tokens, but we also study computational security with stateless tokens that have more complex functionality. Our results can be summarized as follows:

- There exists a class of functions such that the number of bit-OT tokens required to securely implement them is at least the size of the sender's input. The same applies for receiver's input size (with a different class of functionalities).
- Non-adaptive protocols in the hardware token model imply efficient (decomposable) randomized encodings. This can be interpreted as evidence to the impossibility of non-adaptive protocols for a large class of functions.
- There exists a functionality for which there is no protocol in the stateless hardware token model accessing the tokens at most a constant number of times, even when the adversary is computationally bounded.

En route to proving our results, we make interesting connections between the hardware token model and well studied notions such as *OT hybrid model*, *randomized encodings* and *obfuscation*.

*University of Illinois Urbana-Champaign. Email: sagrawl2@illinois.edu. Research supported in part by NSF grants 1228856 and 0747027.

†University of California Los Angeles. Email: prabhanjan@cs.ucla.edu. Part of this work done while visiting IDC Herzliya, supported by the ERC under the EU's Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement n. 307952

‡Microsoft Research, India. Email: vipul@microsoft.com

§University of Illinois Urbana-Champaign. Email: mmp@illinois.edu. Research supported in part by NSF grants 1228856 and 0747027.

¶IDC Herzliya. Email: alon.rosen@idc.ac.il. Supported by ISF grant no. 1255/12 and by the ERC under the EU's Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement n. 307952.

1 Introduction

A protocol for secure two-party computation allows two mutually distrustful parties to jointly compute a function f of their respective inputs, x and y, in a way that does not reveal anything beyond the value $f(x, y)$ being computed. Soon after the introduction of this powerful notion [46, 22], it was realized that most functions $f(x, y)$ do not admit an unconditionally-secure protocol that satisfies it, in the sense that any such protocol implicitly implies the existence (and in some case requires extensive use [2]) of a protocol for Oblivious Transfer (OT) [10, 3, 33, 27, 39]. Moreover, even if one was willing to settle for computational security, secure two-party computation has been shown to suffer from severe limitations in the context of protocol composition [19, 7, 36, 37].

The above realizations have motivated the search for alternative models of computation and communication, with the hope that such models would enable bypassing the above limitations, and as a byproduct perhaps also give rise to more efficient protocols. One notable example is the so called *hardware token model*, introduced by Katz [32]. In this model, it is assumed that one party can generate hardware tokens that implement some efficient functionality in a way that allows the other party only black-box access to the functionality.

The literature on hardware tokens (sometimes referred to as *tamper proof* tokens[1]) discusses a variety of models, ranging from the use of *stateful* tokens (that are destroyed after being queried for some fixed number of times) to *stateless* ones (that can be queried for an arbitrary number of times), with either *non-adaptive* access (in which the queries to the tokens are fixed in advance) or *adaptive* access (in which queries can depend on answers to previous queries). Tokens with varying levels of complexity have also been considered, starting with simple functions such as bit-OT, and ranging all the way to extremely complex functionalities (ones that enable the construction of UC-secure protocols given only a single call to the token).

The use of hardware tokens opened up the possibility of realizing information-theoretically and/or composable secure two-party protocols even in cases where this was shown to be impossible in "plain" models of communication. Two early examples of such constructions are protocols for UC-secure computation [32], and one-time programs [24]. More recently, a line of research initiated by Goyal et al. [26] has focused on obtaining unconditionally-secure two-party computation using stateful tokens that implement the bit-OT functionality. In [25], Goyal et al. went on to show how to achieve UC-secure two party computation using stateless tokens under the condition that tokens can be encapsulated: namely, the receiver of a token A can construct a token B that can invoke A internally. Finally, Dottling et al. [16] have shown that it is possible to obtain information-theoretically secure UC two-party protocols using a single token, assuming it can compute some complex functionality.

Generally speaking, the bit-OT token model has many advantages over a model that allows more complex tokens. First of all, the OT functionality is simple thus facilitating hardware design and implementation. Secondly, in many cases [26], the bit-OT tokens do not depend on the functionality that is being computed. Hence, a large number of bit-OT tokens can be produced "offline" and subsequently used for any functionality. The main apparent shortcoming of bit-OT tokens in comparison to their complex counterparts is that in all previous works the number of tokens used is proportional to the size of the circuit being evaluated, rendering the resulting protocols impractical. This state of affairs calls for the investigation of the minimal number of bit-OT token invocations in a secure two-party computation protocol.

[1] There are papers which deal with the leakage of tokens' contents. We do not consider such a setting in this work.

In this work we aim to study the complexity of constructing secure protocols with respect to different measures in the hardware token model. Our main focus is on non-interactive information-theoretic two-party computation using bit-OT tokens, but we also study computational security with stateless tokens that compute more complex functionalities. En route to proving our results, we make interesting connections between protocols in the hardware token model and well studied notions such as *randomized encodings*, *obfuscation* and the *OT hybrid model*. Such connections have been explored before mainly in the context of obtaining feasibility results [26, 17].

The first question we address is concerned with the number of bit-OT tokens required to securely achieve information-theoretic secure two-party computation. The work on one-time programs makes use of bit-OT tokens in order to achieve secure two party computation in the computational setting, and the number of tokens required in that construction is proportional to the receiver's input size. On the other hand, the only known construction in the information-theoretic setting [26] uses a number of tokens that is proportional to the size of the circuit. This leads us to the following question: is it possible to construct information theoretic two party computation protocols in the token model, where the number of tokens is proportional to the size of the functionality's input? Problems of similar nature have been also studied in the (closely related) OT-hybrid model [15, 2, 45, 41, 42, 44].

The second question we address is concerned with the number of levels of adaptivity required to achieve unconditional two party computation. The known constructions [26] using bit-OT tokens are highly adaptive in nature: the number of adaptive calls required is proportional to the depth of the circuit being computed. The only existing protocols which are non-adaptive are either for specific complexity classes ([30] for NC1) or in the computational setting [24]. An interesting question, therefore, is whether there exist information-theoretic non adaptive protocols for all efficient functionalities.

The works of [25, 38] give negative results on the feasibility of using stateless tokens in the information-theoretic setting. Goyal et al. [26] have shown that it is feasible to construct protocols using stateless tokens under computational assumptions. So, a natural question would be to determine the minimum number of calls to the (stateless) token required in a computational setting.

1.1 Our Results

We exploit the relation between protocols in the hardware token model and cryptographic notions such as randomized encodings and obfuscation to obtain lower bounds in the hardware token model. We focus on non-interactive two-party protocols, where only one party (the sender) sends messages and tokens to the other party (the receiver). Our results are summarized below.

Number of bit-OT tokens in the information-theoretic setting. Our first set of results establishes lower bounds on the number of bit-OT tokens as a function of the parties' input sizes. Specifically:

- We show that there exists a class of functionalities such that the number of tokens required to securely implement them is at least the size of the sender's input. To obtain this result, we translate a similar result in the correlated distributed randomness model by Winkler et al. [44] to this setting.
- We provide another set of functionalities such that the number of tokens required to securely implement them is at least the size of the receiver's input.

While this still leaves a huge gap between the positive result (which uses number of tokens proportional to the size of the circuit) and our lower bound, we note that before this result, even such lower bounds were not known to exist. Even in the case of OT-hybrid model, which is very much related to the hardware token model (and more deeply studied), only lower bounds known are in terms of the sender's input size.

Non-adaptive protocols and randomized encodings. In our second main result we show that non-adaptive protocols in the hardware token model imply efficient randomized encodings. Even though currently known protocols [26] are highly adaptive, it was still not clear that non adaptive protocols for all functionalities were not possible. In fact, all functions in NC1 admit non adaptive protocols in the hardware token model [30]. To study this question, we relate the existence of non-adaptive protocols to the existence of a "weaker" notion of randomized encodings, called *decomposable randomized encodings*. Specifically, we show that if a function has a non adaptive protocol then correspondingly, the function has an efficient decomposable randomized encoding. The existence of efficient decomposable randomized encodings has far-reaching implications in MPC, providing strong evidence to the impossibility of non-adaptive protocols for a large class of functions.

Constant number of calls to stateless tokens. In our last result we show that there exists a functionality for which there does not exist any protocol in the *stateless* hardware token model making at most a constant number of calls. To this end, we introduce the notion of an *obfuscation complete oracle scheme*, a variant of obfuscation tailored to the setting of hardware tokens. Goyal et. al. [26] have shown such a scheme can be realized under computational assumptions (refer to Section 6.2.2 in the full version). We derive a lower bound stating that a constant number of calls to the obfuscation oracle does not suffice. This result can then be translated to a corresponding result in the hardware token model. This result holds even if the hardware is a complex stateless token (and hence still relevant even in light of our previous results) and (more importantly) against computational adversaries. Previous known lower bounds on complex tokens were either for the case of stateful hardware [23, 24, 18] or in the information theoretic setting [25, 38].

Our hope is that the above results will inspire future work on lower bounds in more general settings in the hardware token model and to further explore the connection with randomized encodings, obfuscation and the OT-hybrid model.

1.2 Related Work

The idea of using secure hardware for cryptographic applications goes back to the work of Goldreich and Ostrovsky [23] who studied software protection under the assumption that a 'shielded' CPU is available. Chaum, Pederson, Brands, and Cramer [9, 4, 12] proposed the use of smartcards in the context of electronic cash. More recently, the true power of tamper-proof hardware was demonstrated in the work of Katz [32]. He showed how to construct a UC-secure [6] protocol for any functionality with the help of stateful hardware tokens under the DDH assumption. Chandran et al. [8] improve upon Katz' work in several ways: they allow the adversary to transfer tokens as well as perform re-setting attacks on them. Moran and Segev [40] consider an asymmetric model of computation where only one party *Goliath* is capable of generating tokens. Damgard et al. [13, 14]

weaken the 'isolation' assumption of Katz; they allow a token to communicate a fixed number of bits to its creator.

Goyal et al. [26] construct the first unconditionally secure protocol for arbitrary functionalities using stateful tokens. Their protocol is non-interactive and only one party obtains the output of the computation. They also construct a general purpose obfuscation scheme from stateless tokens. In [25], Goyal et al. provide several interesting results about the power of stateless tokens. They show that such tokens can be used to construct statistically secure commitment protocols and zero-knowledge proofs for **NP**. They also show that if one token can be encapsulated into another, an unconditionally secure protocol exists for any functionality. Furthermore, if this encapsulation is not possible, then statistically secure oblivious transfer cannot be realized. Dottling et al. [16] show that information-theoretic UC-secure two party computation is achievable with the help of just one tamper-proof device. In [18], Dottling et al. study how general resettable computation can be realized using one (or two) stateless tokens and minimal number of rounds of interaction.

2 Preliminaries

2.1 Model of computation

Hardware tokens: Hardware tokens can be divided into two broad categories – stateful and stateless. As the name implies, stateful tokens can maintain some form of state, which might restrict the extent to which they can be used. On the other hand, stateless tokens cannot maintain any state, and could potentially be used an unbounded number of times. The first formal study of hardware tokens modeled them as stateful entities [32], so that they can engage in a two-round protocol with the receiver. Later on, starting with the work of Chandran et al. [8], stateless tokens were also widely studied.

The token functionality models the following sequence of events: (1) a player (the creator) 'seals' a piece of software inside a tamper-proof token; (2) it then sends the token to another player (the receiver), who can run the software in a black-box manner. Once the token has been sent, the creator cannot communicate with it, unlike the setting considered in [13, 14]. We also do not allow token *encapsulation* [25], a setting in which tokens can be *placed inside* other tokens.

Stateless tokens: The $\mathcal{F}_{wrap}^{stateless}$ functionality models the behavior of a stateless token. It is parameterized by a polynomial $p(.)$ and an implicit security parameter k. Its behavior is described as follows:

- **Create**: Upon receiving $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid}, M)$ from P_i, where M is a Turing machine, do the following: (a) Send $(\mathsf{create}, \mathsf{sid}, P_i, P_j, \mathsf{mid})$ to P_j, and (b) Store $(P_i, P_j, \mathsf{mid}, M)$.
- **Execute**: Upon receiving $(\mathsf{run}, \mathsf{sid}, P_i, \mathsf{mid}, \mathsf{msg})$ from P_j, find the unique stored tuple $(P_i, P_j, \mathsf{mid}, M)$. If no such tuple exist, do nothing. Run $M(msg)$ for at most $p(k)$ steps, and let out be the response ($\mathsf{out} = \perp$ if M does not halt in $p(k)$ steps). Send $(\mathsf{sid}, P_i, \mathsf{mid}, \mathsf{out})$ to P_j.

Here sid and mid denote the session and machine identifier respectively.

Stateful tokens: In the class of stateful tokens, our primary interest is the One Time Memory (OTM) token, studied first in [24]. This token implements a single Oblivious Transfer (OT) call, and hence is also referred to as OT token. Oblivious transfer, as we know, is one of the most widely studied primitives in secure multi-party computation. In the $\binom{n}{t}$-OT^k variant, sender has n strings

of k bits each, out of which a receiver can pick any t. The sender does not learn anything in this process, and the receiver does not know what the remaining $n-t$ strings were. The behavior of an OTM token is similar to $\binom{2}{1}$-OT^k.

The primary difference between the OT functionality and an OTM token is that while the functionality forwards an acknowledgment to the sender when the receiver obtains the strings of its choice, there is no such feedback provided by the token. Hence, one has to put extra checks in a protocol (in the token model) to ensure that the receiver opens the tokens when it is supposed to (see, for example, Section 3.1 in [26]). Formal definitions of $\mathcal{F}^{OT}$ and $\mathcal{F}^{OTM}$ are given below. We would be dealing with OTMs where both inputs are single bits. We will refer to them as bit-OT tokens.

Oblivious Transfer (OT): The functionality $\mathcal{F}^{OT}$ is parameterized by three positive integers n, t and k, and behaves as follows.

- On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \ldots, s_n))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \ldots, s_n))$. Here each s_i is a k-bit string.
- On receiving $(P_i, \mathsf{sid}, \mathsf{id}, l_1, l_2, \ldots, l_t)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \ldots, s_n))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_{l_1}, s_{l_2}, \ldots, s_{l_t})$ to P_j, **send an acknowledgment** $(P_j, \mathsf{sid}, \mathsf{id})$ to P_i, and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_1, s_2, \ldots, s_n))$. Else, do nothing. Here each l_j is an integer between 1 and n.

One Time Memory (OTM): The functionality $\mathcal{F}^{OTM}$ which captures the behavior an OTM is described as follows:

- On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$.
- On receiving $(P_i, \mathsf{sid}, \mathsf{id}, c)$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, s_c)$ to P_j and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (s_0, s_1))$. Else, do nothing.

Non-interactivity: In this paper, we are interested in non-interactive two-party protocols (i.e., where only one party sends messages and tokens to the other). Some of our results, however, hold for an interactive setting as well (whenever this is the case, we point it out). The usual setting is as follows: Alice and Bob have inputs $x \in \mathcal{X}_k$ and $y \in \mathcal{Y}_k$ respectively, and they wish to securely compute a function $f : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{Z}_k$, such that only Bob receives the output $f(x, y) \in \mathcal{Z}_k$ of the computation (here, k is the security parameter). Only Alice is allowed to send messages and tokens to Bob.

Circuit families. In this work, we assume that parties are represented by circuit families instead of Turing machines. A circuit is an acyclic directed graph, with the gates of the circuit representing the nodes of the graph, and the wires representing the edges in the graph. A circuit can either be boolean or arithmetic depending on whether the gates are AND-OR or ADD-MULTIPLY gates. We assume that the circuit can be broken down into layers such that the first layer takes the input of the circuit and outputs to the second layer and so on. The output of the last layer is the output of the circuit.

A circuit is typically characterized by its size and its depth. The size of the circuit is the sum of the number of gates and the number of wires in the circuit. We define the depth of a circuit C,

denoted by $\mathsf{Depth}(C)$ to be the number of layers in the circuit. There are several complexity classes defined in terms of depth and size of circuits. One important complexity class that we will refer in this work is the NC1 complexity class. This comprises of circuits which have depth $O(\log(n))$ and size $\mathsf{poly}(n)$, where n is the input size of the circuit. Languages in P can be represented by a circuit family whose size is polynomial in the size of the input.

2.2 Security

Definition 1 (Indistinguishability)**.** *A function $f : \mathbb{N} \to \mathbb{R}$ is negligible in n if for every polynomial $p(.)$ and all sufficiently large n's, it holds that $f(n) < \frac{1}{p(n)}$. Consider two probability ensembles $X := \{X_n\}_{n\in\mathbb{N}}$ and $Y := \{Y_n\}_{n\in\mathbb{N}}$. These ensembles are computationally indistinguishable if for every PPT algorithm $\mathcal{A}$, $|\mathsf{Pr}[\mathcal{A}(X_n, 1^n) = 1] - \mathsf{Pr}[\mathcal{A}(Y_n, 1^n) = 1]|$ is negligible in n. On the other hand, these ensembles are statistically indistinguishable if $\Delta(X_n, Y_n) = \frac{1}{2}\sum_{\alpha\in\mathcal{S}} |\mathsf{Pr}[X_n = \alpha] - \mathsf{Pr}[Y_n = \alpha]|$ is negligible in n, where $\mathcal{S}$ is the support of the ensembles. The quantity $\Delta(X_n, Y_n)$ is known as the statistical difference between X_n and Y_n.*

Statistical security: A protocol π for computing a two-input function $f : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{Z}_k$ in the hardware-token model involves Alice and Bob exchanging messages and tokens. In the (static) semi-honest model, an adversary could corrupt one of the parties at the beginning of an execution of π. Though the corrupted party does not deviate from the protocol, the adversary could use the information it obtains through this party to learn more about the input of the other party. At an intuitive level, a protocol is secure if any information the adversary could learn from the execution can also be obtained just from the input and output (if any) of the corrupted party. Defining security formally though requires that we introduce some notation, which we do below.

Let the random variables $\mathsf{view}_A^\pi(x, y) = (x, R_A, M, U)$ and $\mathsf{view}_B^\pi(x, y) = (y, R_B, M, V)$ denote the views of Alice and Bob respectively in the protocol π, when Alice has input $x \in \mathcal{X}_k$ and Bob has input $y \in \mathcal{Y}_k$. Here R_A (resp. R_B) denotes the coin tosses of Alice (resp. Bob), M denotes the messages exchanged between Alice and Bob, and U (resp. V) denotes the messages exchanged between Alice (resp. Bob) and the token functionality. Also, let $\mathsf{out}_B^\pi(x, y)$ denote the output produced by Bob. We can now formally define security as follows.

Definition 2 (ϵ-secure protocol [44])**.** *A two-party protocol π computes a function $f : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{Z}_k$ with $\epsilon - \mathsf{security}$ in the semi-honest model if there exists two randomized functions $\mathcal{S}_A$ and $\mathcal{S}_B$ such that for all sufficiently large values of k, the following two properties hold for all $x \in \mathcal{X}_k$ and $y \in \mathcal{Y}_k$:*

- $\Delta((\mathcal{S}_A(x), f(x, y)), (\mathsf{view}_A^\pi(x, y), \mathsf{out}_B^\pi(x, y))) \leq \epsilon(k)$,
- $\Delta(\mathcal{S}_B(y, f(x, y)), \mathsf{view}_B^\pi(x, y)) \leq \epsilon(k)$.

If π computes f with ϵ-security for a negligible function $\epsilon(k)$, then we simply say that π securely computes f. Further if $\epsilon(k) = 0$, π is a prefectly secure protocol for f.

Information Theory: We define some information-theoretic notions which will be useful in proving unconditional lower bounds. *Entropy* is a measure of the uncertainty in a random variable. The entropy of X given Y is defined as:

$$H(X|Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} \mathsf{Pr}[X = x \wedge Y = y] \log \mathsf{Pr}[X = x \mid Y = y].$$

For the sake of convenience, we sometimes use $h(p) = -p \log p - (1-p)\log(1-p)$ to denote the entropy of a binary random variable which takes value 1 with probability p ($0 \leq p \leq 1$).

Mutual information is a measure of the amount of information one random variable contains about another. The mutual information between X and Y given Z is defined as follows:

$$I(X;Y|Z) = H(X|Z) - H(X|YZ).$$

See [11] for a detailed discussion of the notions above.

3 Lower Bounds in Input Size for Unconditional Security

In this section, we show that the number of simple tokens required to be exchanged in a two-party unconditionally secure function evaluation protocol could depend on the input size of the parties. We obtain two bounds discussed in detail in the sections below. Our first bound relates the number of hardware tokens required to compute a function with the input size of the sender. (This bound holds even when the protocol is interactive.) In particular, we show that the number of bit-OT tokens required for oblivious transfer is at least the sender's input size (minus one). Our second result provides a class of functions where the number of bit-OT tokens required is at least the input size of the receiver.

3.1 Lower bound in Sender's input size

In this subsection we consider k to be fixed, and thus omit k from $\mathcal{X}_k$, $\mathcal{Y}_k$ and $\epsilon(k)$ for clarity. In [44], Winkler and Wullschleger study unconditionally secure two-party computation in the semi-honest model. They consider two parties Alice and Bob, with inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively, who wish to compute a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ such that only Bob obtains the output $f(x,y) \in \mathcal{Z}$ (but Alice and Bob can exchange messages back and forth). The parties have access to a functionality $\mathcal{G}$ which *does not take any input*, but outputs a sample (u,v) from a distribution p_{UV}. Winkler and Wullschleger obtain several lower bounds on the information-theoretic quantities relating U and V for a secure implementation of the function f.

Here, we would like to obtain the minimum number of bit-OT tokens required for a secure realization of a function. The functionality which models the token behavior $\mathcal{F}^{OTM}$ is an interactive functionality: not only does $\mathcal{F}^{OTM}$ give output to the parties, but also take inputs from them. Therefore, as such the results of [44] are not applicable to our setting. However, if we let U denote all the messages exchanged between Alice and $\mathcal{G}$, and similarly let V denote the entire message transcript between Bob and $\mathcal{G}$, we claim that the following lower bound (obtained for a non-interactive $\mathcal{G}$ in [44]) holds even when the functionality $\mathcal{G}$ is interactive. This will allow us to apply this bound on protocols where hardware tokens are exchanged.

Theorem 1. *Let $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a function such that*

$$\forall x \neq x' \in \mathcal{X}\ \exists y \in \mathcal{Y} : f(x,y) \neq f(x',y).$$

If there exists a protocol that implements f from a functionality $\mathcal{G}$ with ϵ security in the semi-honest model, then

$$H(U|V) \geq \mathsf{max}_{y \in \mathcal{Y}} H(X|f(X,y)) - (3|\mathcal{Y}| - 1)(\epsilon \log |\mathcal{Z}| + h(\epsilon)) - \epsilon \log |\mathcal{X}|,$$

where $H(U|V)$ is the entropy of U given V.

Proof. In order to prove that Theorem 1 holds with an interactive $\mathcal{G}$, we observe that the proof provided by Winkler and Wullschleger for a non-interactive $\mathcal{G}$ *almost* goes through for an interactive one. An important fact they use in their proof is that for any protocol π, with access to a non-interactive $\mathcal{G}$, the following mutual information relation holds: $I(X; VY|UM) = 0$, where M denotes the messages exchanged in the protocol. (In other words, $X - UM - VY$ is a Markov chain.) If one can show that the aforementioned relation holds even when $\mathcal{G}$ can take inputs from the parties (and U and V are redefined as discussed above), the rest of the proof goes through, as can be verified by inspection. Hence, all that is left to do is to prove that $I(X; VY|UM) = 0$ is true in the more general setting, where U and V stand for the *transcripts* of interactions with $\mathcal{G}$.

We will need the following simple chain rule about mutual information (for a proof see [11]):

$$I(AA'; B|C) = I(A; B|C) + I(A'; B|AC). \tag{1}$$

In particular, this implies the following:

$$\begin{aligned} I(A; B|CD) &= I(A; BC|D) - I(A; C|D) &(2)\\ &= I(AC; B|D) - I(B; C|D) &(3)\end{aligned}$$

Our goal is to show that the mutual information between X and VY conditioned on UM is 0. We prove this result with the help of induction on the steps in a protocol. At the beginning of the protocol when no messages have been exchanged (U, V and M are empty random variables), $I(X; VY|UM)$ simply becomes $I(X; Y)$. Since we can assume that the inputs of Alice and Bob are chosen independently of each other, $I(X; Y) = 0$. This proves the base case.

Suppose that the protocol has executed for i steps. Let the messages exchanged up to the first i steps between Alice and Bob be denoted by M^i, that between Alice and $\mathcal{G}$ by U^i, and that between Bob and $\mathcal{G}$ by V^i. By the induction hypothesis, we know that $I(X; V^iY|U^iM^i) = 0$. At the $i+1$th step one of the following things can happen: Alice sends a message to Bob, or vice versa; Alice sends a message to $\mathcal{G}$, or vice versa; Bob sends a message to $\mathcal{G}$, or vice versa. Accordingly, one of the random variables above gets updated. We prove that $I(X; V^{i+1}Y|U^{i+1}M^{i+1}) = 0$ for some of the updates, and the rest will follow in a similar manner:

- *Alice sends a message to* $\mathcal{G}$: In this case, $V^{i+1} = V^i$ and $M^{i+1} = M^i$. Let the message sent be denoted by U_{i+1}. Now,

$$\begin{aligned} I(X; V^iY|U^{i+1}M^i) &= I(X; V^iY|U^iU_{i+1}M^i)\\ &= I(XU_{i+1}; V^iY|U^iM^i) - I(V^iY; U_{i+1}|U^iM^i)\\ &= I(X; V^iY|U^iM^i) + I(U_{i+1}; V^iY|XU^iM^i)\\ &\qquad - I(V^iY; U_{i+1}|U^iM^i), \end{aligned}$$

  where the second equality follows from (3), and the third from (1). We know that $I(X; V^iY|U^iM^i) = 0$ by the induction hypothesis. Moreover, since U_{i+1} is a message sent by Alice, it is a function of X, U^i and M^i (and the local randomness of Alice). Therefore, $V^iY \to XU^iM^i \to U_{i+1}$, or $I(U_{i+1}; V^iY|XU^iM^i) = 0$. Now since mutual information is a positive quantity, $I(X; V^iY|U^{i+1}M^i)$ must be 0.

- *$\mathcal{G}$ sends a message to Bob*: In this case, $U^{i+1} = U^i$ and $M^{i+1} = M^i$. Let the message sent be denoted by V_{i+1}. Now,

$$\begin{aligned} I(X; V^{i+1}Y|U^iM^i) &= I(X; V^iV_{i+1}Y|U^iM^i) \\ &= I(X; V^iY|U^iM^i) + I(X; V_{i+1}|U^iM^iV^iY) \\ &= I(X; V^iY|U^iM^i) + I(XM^iY; V^{i+1}|U^iV^i) \\ &\qquad - I(V^{i+1}; M^iY|U^iV^i), \end{aligned}$$

  where the second equality follows from (1), and the third from (3). We know that $I(X; V^iY|U^iM^i) = 0$ by the induction hypothesis. Moreover, since V_{i+1} is a message sent by $\mathcal{G}$, it is a function of U^i and V^i (and the local randomness of $\mathcal{G}$ if any). Therefore, $XM^iY \to U^iV^i \to V_{i+1}$, or $I(XM^iY; V_{i+1}|U^iV^i) = 0$. Now since mutual information is a positive quantity, $I(X; V^{i+1}Y|U^iM^i)$ must be 0.

- *Bob sends a message to Alice*: In this case, $U^{i+1} = U^i$ and $V^{i+1} = V^i$. Let the message sent be denoted by M_{i+1}. Now,

$$\begin{aligned} I(X; V^iY|U^iM^{i+1}) &= I(X; V^iY|U^iM^iM_{i+1}) \\ &= I(X; V^iYM_{i+1}|U^iM^i) - I(X; M_{i+1}|U^iM^i) \\ &= I(X; V^iY|U^iM^i) + I(X; M^{i+1}|U^iM^iV^iY) \\ &\qquad - I(X; M_{i+1}|U^iM^i) \\ &= I(X; M^{i+1}|U^iM^iV^iY) - I(X; M_{i+1}|U^iM^i) \\ &= I(XU^i; M^{i+1}|M^iV^iY) - I(M^{i+1}; U^i|M^iV^iY) \\ &\qquad - I(X; M_{i+1}|U^iM^i), \end{aligned}$$

  where the second equality follows from (2), the third from (1), the fourth by the induction, and the fifth by (3). Since M_{i+1} is a message sent by $\mathcal{G}$, it is a function of M^i, V^i and Y (and the local randomness of Bob). Therefore, $XU^i \to M^iV^iY \to M_{i+1}$, or $I(XU^i; M_{i+1}|M^iV^iY) = 0$. Now since mutual information is a positive quantity, $I(X; V^iY|U^iM^{i+1})$ must be 0.

We can handle rest of the events in a similar fashion. This completes the proof of Theorem 1. □

Theorem 1 lets us bound the number of tokens required to securely evaluate a function, as follows. Suppose Alice and Bob exchange ℓ bit-OT tokens during a protocol. If Bob is the recipient of a token, there is at most one bit of information that is hidden from Bob after he has queried the token. On the other hand, if Bob sends a token, he does not know what Alice queried for. Therefore given V, entropy of U can be at most ℓ (or $H(U|V) \leq \ell$). We can use this observation along with Corollary 3 in [44] (full version) to obtain the following result.

Theorem 2. *If a protocol ϵ-securely realizes m independent instances of $\binom{n}{t}$-OT^k, then the number of bit-OT tokens ℓ exchanged between Alice and Bob must satisfy the following lower bound:*

$$\ell \geq ((1-\epsilon)n - t)km - (3\lceil n/t \rceil - 1)(\epsilon mtk + h(\epsilon)).$$

We conclude this section with a particular case of the above theorem which gives a better sense of the bound. Let us say that Alice has a string of n bits, and Bob wants to pick one of them. In other words, Alice and Bob wish to realize an instance of $\binom{n}{1}$-OT^1. Also, assume that they want to do this with perfect security, i.e., $\epsilon = 0$. In this case, the input size of Alice is n, but Bob's input size is only $\lceil \log n \rceil$. Now, we have the following corollary.

Corollary 3. *In order to realize the functionality $\binom{n}{1}$-OT^1 with perfect security, Alice and Bob must exchange at least $n-1$ tokens.*

Suppose Alice is the only party who can send tokens. Then, we can understand the above result intuitively in the following way. Alice has n bits, but she wants Bob to learn exactly one of them. However, since she does not know which bit Bob needs, she must send her entire input (encoded in some manner) to Bob. Suppose Alice sends ℓ bit-OT tokens to Bob. Since Bob accesses every token, the ℓ bits it obtains from the tokens should give only one bit of information about Alice's input. The remaining ℓ positions in the tokens, which remain hidden from Bob, must contain information about the remaining $n-1$ bits of Alice's input. Hence, ℓ must be at least $n-1$.

One can use Protocol 1.2 in [5] to show that the bound in Corollary 3 is tight.

3.2 Lower bound in Receiver's input size

In this section, we show that the number of bit-OT tokens required could depend on the receiver's input size. We begin by defining a non-replayable function family, for which we shall show that the number of tokens required is at least the input size of the receiver.

Definition 3. *Consider a function family $f : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{Z}_k$, $k \in \mathbb{I}^+$. We say that f is replayable if for every distribution $\mathcal{D}_k$ over $\mathcal{X}_k$, there exists a randomized algorithm $\mathcal{S}_B$ and a negligible function ν, such that on input $(k, y, f(x,y))$ where $(x,y) \leftarrow \mathcal{D}_k \times \mathcal{Y}_k$, $\mathcal{S}_B$ outputs $\perp$ with probability at most $3/4$, and otherwise outputs (y', z) such that (conditioned on not outputting $\perp$) with probability at least $1-\nu(k)$, $y' \neq y$ and $z = f(x,y')$.*

Theorem 4. *Let $f : \mathcal{X}_k \times \mathcal{Y}_k \to \mathcal{Z}_k$ be a function that is not replayable. Then, in any non-interactive protocol π that securely realizes f in the semi-honest model using bit-OT tokens, Alice must send at least $n(k) = \lfloor \log |\mathcal{Y}_k| \rfloor$ tokens to Bob.*

Proof. For simplicity, we omit the parameter k in the following. Suppose Alice sends only $n-1$ bit-OT tokens to Bob in the protocol π. We shall show that f is in fact replayable, by constructing an algorithm $\mathcal{S}_B$ as in Definition 3, from a semi-honest adversary $\mathcal{A}$ that corrupts Bob in an execution of π.

Let the input of Alice and Bob be denoted by x and y respectively, where x is chosen from $\mathcal{X}$ according to the distribution $\mathcal{D}$, and y is chosen uniformly at random over $\mathcal{Y}$. On input x, Alice sends tokens $(\Upsilon_1, \cdots, \Upsilon_{n-1})$ and a message m to Bob. Bob runs his part of the protocol with inputs y, m, a random tape r, and (one-time) oracle access to the tokens. Without loss of generality, we assume that Bob queries all the $n-1$ tokens. Bob's view consists of y, m, r, and the bits $b = (b_1, \ldots, b_{n-1})$ received from the $n-1$ tokens $(\Upsilon_1, \ldots, \Upsilon_{n-1})$. Let $q = (q_1, \ldots, q_{n-1})$ denote the query bits that Bob uses for the $n-1$ tokens. For convenience, we shall denote the view of Bob as (y, m, r, q, b) (even though q is fully determined by the rest of the view).

We define $\mathcal{S}_B$ as follows: on input (y_1, z_1), it samples a view (y, r, m, q, b) for Bob in an execution of π conditioned on $y = y_1$ and Bob's output being z_1. Next, it samples a second view

(y', r', m', q', b') conditioned on $(m', q', b') = (m, q, b)$. If $y' = y$, it outputs $\perp$. Else, it computes Bob's output z' in this execution and outputs (y', z').

To argue that $\mathcal{S}_B$ meets the requirements in Definition 3, it is enough to prove that when $x \in \mathcal{X}$ is sampled from any distribution $\mathcal{D}$, $y \leftarrow \mathcal{Y}$ is chosen uniformly, and $z = f(x, y)$: (1) (y', r', m', q', b') sampled by $\mathcal{S}_B(y, z)$ is distributed close (up to a negligible distance) to Bob's view in an actual execution with inputs (x, y'), and (2) with probability at least $\frac{1}{4}$, $y' \neq y$. Then, by the correctness of π, with overwhelming probability, whenever $\mathcal{S}_B$ outputs (y', z'), it will be the case that $z' = f(x, y')$, and this will happen with probability at least $1/4$.

The first claim follows by the security guarantee and the nature of a token-based protocol. Consider the experiment of sampling (x, y) and then sampling Bob's view (y, r, m, q, b) conditioned on input being y and output being $z = f(x, y)$. Firstly, this is only negligibly different from sampling Bob's view from an actual execution of π with inputs x and y, since by the correctness guarantee, the output of Bob will indeed be $f(x, y)$ with high probability. Now, sampling (x, y, r, m, q, b) in the actual execution can be reinterpreted as follows: first sample (m, q, b), and then conditioned on (m, q, b), sample x and (y, r) independent of each other. This is because, by the nature of the protocol, conditioned on (m, q, b), Bob's view in this experiment is independent of x. Now, (y', r') is also sampled conditioned on (m, q, b) in the same manner (without resampling x), and hence (x, y', r', m, q, b) is distributed as in an execution of π with inputs (x, y').

To show that $\mathcal{S}_B$ outputs $\perp$ with probability at most $\frac{3}{4}$, we rely on the fact that the number of distinct inputs y for Bob is 2^n, but the number of distinct queries the Bob can make to the tokens q is at most 2^{n-1}. Below, we fix an (m, b) pair sampled by $\mathcal{S}_B$, and argue that $\Pr[y = y'] \leq \frac{3}{4}$ (where the probabilities are all conditioned on (m, b)).

For each value of $q \in \{0,1\}^{n-1}$ that has a non-zero probability of being sampled by $\mathcal{S}_B$, we associate a value $Y(q) \in \{0,1\}^n$ as $Y(q) = \text{argmax}_y \Pr[y|q]$, where the probability is over the choice of $y \leftarrow \mathcal{Y}$ and the random tape r for Bob. If more than one value of y attains the maximum, $Y(q)$ is taken as the lexicographically smallest one. Let $\mathcal{Y}^* = \{y | \exists q \text{ s.t. } y = Y(q)\}$. Then, $|\mathcal{Y}^*| \leq |\mathcal{Y}|/2$, or equivalently (since the distribution over $\mathcal{Y}$ is uniform), $\Pr[y \notin \mathcal{Y}^*] \geq \frac{1}{2}$.

Let $\mathcal{Q}^* = \{q | \Pr[Y(q)|q] > \frac{1}{2}\}$. Further, let $\beta = \min\{\Pr[Y(q)|q] | q \in \mathcal{Q}^*\}$. Note that $\beta > \frac{1}{2}$. We claim that $\alpha := \Pr[q \in \mathcal{Q}^*] \leq \frac{1}{2}$. This is because

$$\begin{aligned}\frac{1}{2} \leq \Pr[y \notin \mathcal{Y}^*] &= \sum_{y \notin \mathcal{Y}^*, q \in \mathcal{Q}^*} \Pr[y, q] + \sum_{y \notin \mathcal{Y}^*, q \notin \mathcal{Q}^*} \Pr[y, q] \\ &\leq \sum_{q \in \mathcal{Q}^*} (1 - \beta) \Pr[q] + \sum_{q \notin \mathcal{Q}^*} \beta \Pr[q] = \alpha(1 - \beta) + \beta(1 - \alpha)\end{aligned}$$

Since $\beta > \frac{1}{2}$, if $\alpha > \frac{1}{2}$ then $\alpha(1 - \beta) + \beta(1 - \alpha) < \frac{1}{2}$, which is a contradiction. Hence $\alpha \leq \frac{1}{2}$. Now,

$$\begin{aligned}\Pr[y = y'] &\leq \alpha \Pr[y = y' | q \in \mathcal{Q}^*] + (1 - \alpha) \Pr[y = y' | q \notin \mathcal{Q}^*] \\ &\leq \alpha + (1 - \alpha)\frac{1}{2} \leq \frac{3}{4}.\end{aligned}$$

□

We give a concrete example of a function family that is not replayable. Let $\mathcal{X}_k = \{1, 2, \ldots, k\}$ be the set of first k positive integers. Let $\mathcal{Y}_k = \{S \subseteq \mathcal{X}_k : |S| = k/2 \wedge 1 \in S\}$. Define $f : \mathcal{X}_k \times \mathcal{Y}_k \to \{0, 1\}$ as follows: for all k, $x \in \mathcal{X}_k$ and $y \in \mathcal{Y}_k$, $f(x, S) = 1$ if $x \in S$, and 0 otherwise.

Fix a value of k. Suppose a simulator $\mathcal{S}_B$ is given S and $f(X,S)$ as inputs, where X, S denote random variables uniformly distributed over $\mathcal{X}_k$ and $\mathcal{Y}_k$ respectively. From this input, $\mathcal{S}_B$ knows that X could take one of $k/2$ possible values. Any $S' \neq S$ intersects S' or its complement in at most $k/2-1$ positions. Hence, $\mathcal{S}_B$ can guess the value of $f(X,S')$ with probability at most $1-2/k$. This implies that if $\mathcal{S}_B$ outputs (S', Z) with probability $1/4$, with a non-negligible probability $Z \neq f(X,S')$.

Note that the number of bits required to represent an element of $\mathcal{X}_k$ is only $\lceil \log k \rceil$, but that required to represent an element of $\mathcal{Y}_k$ is $n(k) = \lceil \log \frac{1}{2}\binom{k}{k/2} \rceil$, which is at least a polynomial in k. Since f is not replayable, it follows from Theorem 4 that in any protocol that realizes f, Alice must send at least $n(k)$ tokens to Bob.

4 Negative Result for Non-Adaptive Protocols

4.1 Setting

In this section, we explore the connection between the randomized encodings of functions and the protocols for the corresponding functionalities [2] in the bit-OT (oblivious transfer) token model. We deal with only protocols which are non-adaptive, non-interactive and are perfectly secure. The notions of non-interactivity (Section 2.1) and perfect security (Definition 2 in Section 2.2) have already been dealt with in the preliminaries. We will only explain the notion of non-adaptivity. A protocol in the bit-OT token model is said to be *non-adaptive* if the queries to the tokens are fixed in advance. This is in contrast with the adaptive case where the answers from one token can used to generate the query to the next token.

Such (non-adaptive and non-interactive) protocols have been considered in the literature and one-time programs [24, 26] is one such example, although one-time programs deal with malicious receivers. Henceforth, when the context is clear we will refer to "perfectly secure non-adaptive non-interactive protocols" as just "non-adaptive protocols".

We show that the existence of non-adaptive protocols for a function in the bit-OT token model implies an efficient (polynomial sized) decomposable randomized encoding for that function. This is done by establishing an equivalence relation between decomposable randomized encodings and a specific type of non-adaptive protocols in the bit-OT token model. Then, we show that a functionality having any non-adaptive protocol also has this specific type of protocol thereby showing the existence of a DRE for this functionality. Since decomposable randomized encodings are believed to not exist for all functions in P [21, 43, 20, 31], this gives a strong evidence to the fact that there cannot exist non-adaptive protocols in the bit-OT token model for all functions in P.

4.2 Randomized Encodings

We begin this section by describing the necessary background required to understand randomized encodings [29]. A randomized encoding for a function f consists of two procedures - encode and decode. The encode procedure takes an input circuit for f, x which is to be input to f along with randomness r and outputs $\hat{f}(x;r)$. The decode procedure takes as input $\hat{f}(x;r)$ and outputs $f(x)$. There are two properties that the encode and decode procedures need to satisfy for them to qualify

[2] Here, we abuse the notation and interchangeably use functions and functionalities.

to be a valid randomized encoding. The first property is (perfect) correctness which says that the decode algorithm always outputs $f(x)$ when input $\hat{f}(x;r)$. The second property, namely (perfect) privacy, says that there exists a simulator such that the output distribution of the encode algorithm on input x is identical to the output distribution of the simulator on input $f(x)$.

We deal with a specific type of randomized encodings termed as decomposable randomized encodings [34, 21, 28, 30, 35] which are defined as follows.

Definition 4. *An (efficient) Decomposable Randomized Encoding, denoted by DRE, consists of three PPT algorithms* (RE.Encode, RE.ChooseInpWires, RE.Decode)*:*

1. RE.Encode*: takes as input a circuit C and outputs $(\tilde{C}, \mathsf{state})$,where $\mathsf{state} = ((s_1^0, s_1^1), \ldots, (s_m^0, s_m^1))$ and m is the input size of the circuit.*
2. RE.ChooseInpWires*: takes as input (state, x) and outputs $\tilde{x}$, where x is of length m and $\tilde{x} = (s_1^{x_1}, \ldots, s_m^{x_m})$ and x_i is the i^{th} bit of x.*
3. RE.Decode*: takes as input $(\tilde{C}, \tilde{x})$ and outputs* out*.*

A decomposable randomized encoding needs to satisfy the following properties.

(Correctness):- Let RE.Encode on input C output $(\tilde{C}, \mathsf{state})$. Let RE.ChooseInpWires on input (state, x) output $\tilde{x}$. Then, RE.Decode$(\tilde{C}, \tilde{x})$ always outputs $C(x)$.
(Perfect privacy):- There exists a PPT simulator Sim such that the following two distributions are identical.

- $\left\{(\tilde{C}, \tilde{x})\right\}$, where $(\tilde{C}, \mathsf{state})$ is the output of RE.Encode on input C and $\tilde{x}$ is the output of ChooseInpWires on input (state, x).
- $\left\{(\tilde{C}_{\mathsf{Sim}}, \tilde{x}_{\mathsf{Sim}})\right\}$, where $(\tilde{C}_{\mathsf{Sim}}, \tilde{x}_{\mathsf{Sim}})$ is the output of the simulator Sim on input $C(x)$.

In the above definition, ϵ-privacy can also be considered instead of perfect privacy where the distributions are ϵ far from each other for some negligible ϵ. In this section, we only deal with DRE with perfect privacy. It can be verified that a decomposable randomized encoding is also a randomized encoding. There are efficient decomposable randomized encodings known for all functions in NC^1 [34, 30]. However, it is believed that there does not exist efficient decomposable randomized encodings for all functions in P. The existence of efficient decomposable randomized encodings for all efficiently computable functions has interesting implications, namely, multiparty computation protocols in the PSM (Private Simultaneous Message) model [21], constant-round two-party computation protocol in the OT-hybrid model [43, 20] and multiparty computation with correlated randomness [31].

We now proceed to relate the existence of non-adaptive protocols for a functionality to the existence of randomized encodings, and more specifically DRE, for the corresponding function. But first, we give an overview of our approach and then we describe the technical details.

4.3 Overview

We first make a simple observation which is the starting point to establish the connection between randomized encodings and non-adaptive protocols in the bit-OT token model. Consider the answer

obtained by the receiver of the non-adaptive protocol after querying the tokens. This answer can be viewed as a decomposable randomized encoding. The message contained in the bit-OT tokens along with the software sent by the sender corresponds to the output of the encode procedure. The choose-input-wires procedure corresponds to the algorithm the receiver executes before querying the bit-OT tokens. The decode procedure corresponds to the decoding of the answer from the tokens done by the receiver to obtain the output of the functionality. Further, these procedures satisfy the correctness and the privacy properties. The correctness of the decoding of the output follows directly from the correctness of the protocol. The privacy of the decomposable randomized encoding follows from the fact that the answer obtained from the tokens can be simulated which in turn follows from the privacy of the protocol. At this point it may seem that this observation directly gives us a decomposable randomized encoding from a non-adaptive protocol. However, there are two main issues. Firstly, the output of the encode procedure given by the protocol can depend on the input of the function while in the case of DRE, the encode procedure is independent of the input of the function. Secondly, the choose-inputs-procedure given by the protocol might involve a complex preprocessing on the receiver's input before it queries the tokens. This is in contrast to the choose-inputs-procedure of a DRE where no preprocessing is done on the input of the function.

We overcome these issues in a series of steps to obtain a DRE for a function from a non-adaptive protocol for that function. In the first step, we split the sender's algorithm into two parts - the first part does computation solely on the randomness and independent of the input while the second part does preprocessing on both its input as well as the randomness. We call protocols which have the sender defined this way to be $\mathsf{SplitState}$ protocols. We observe that every function that has a non-adaptive protocol also has a $\mathsf{SplitState}$ protocol. In the next step, we try to reduce the complexity of the preprocessing done on both the sender's as well as the receiver's inputs. The preprocessing refers to the computation done on the inputs before the hardware tokens are evaluated. We call protocols which have no preprocessing on its inputs to be *simplified* protocols. Our goal is then to show that if a protocol has a $\mathsf{SplitState}$ protocol then it also has a simplified protcol. At the heart of this result lies the observation that all NC^1 protocols have simplified protocols. We use the simplified protocols for NC^1 to recursively reduce the complexity of the preprocessing algorithm in a $\mathsf{SplitState}$ protocol to finally obtain a simplified protocol. Finally, by using an equivalence relation established between simplified protocols and efficient DRE, we establish the result that a function having a non-adaptive protocol also has an efficient DRE. We now proceed to the technical details.

4.4 Equivalence of RE and simplified protocols

We now show the equivalence of randomized encodings and simplified protocols in the bit-OT token model.

$\mathsf{SplitState}$ **protocols**. Consider the protocol Π in the bit-OT token model. We say that Π is a $\mathsf{SplitState}$ protocol if the sender and the receiver algorithms in $\mathsf{SplitState}$ protocol are defined as follows. The sender in Π consists of the tuple of algorithms ($\Pi.\mathsf{InpFreePP}$, $\Pi.\mathsf{Preproc}_{\mathsf{sen}}$, $\Pi.\mathsf{EvalHT}_{\mathsf{sen}}$). It takes as input x with randomness R_{sen} and executes the following steps.

- It first executes $\Pi.\mathsf{InpFreePP}$ on input R_{sen} to obtain the tokens ($\mathsf{htokens}_{\mathsf{sen}}$, $\mathsf{htokens}_{\mathsf{rec}}$) and $\mathsf{Software}$.
- It then executes $\Pi.\mathsf{Preproc}_{\mathsf{sen}}$ on input (x, R_{sen}) to obtain x'.

- It then executes $\Pi.\mathsf{EvalHT}_{\mathsf{sen}}$ on input $(x', \mathsf{htokens}_{\mathsf{sen}})$. The procedure $\Pi.\mathsf{EvalHT}_{\mathsf{sen}}$ evaluates the i^{th} token in $\mathsf{htokens}_{\mathsf{sen}}$ with the i^{th} bit of x' to obtain $\tilde{x}_i$. The value $\tilde{x}$ is basically the concatenation of all $\tilde{x}_i$.

- The sender then outputs $(\mathsf{htokens}_{\mathsf{rec}}, \mathsf{Software}, \tilde{x})$.

Notice that the third step in the above sender's procedure involves the sender evaluating the tokens $\mathsf{htokens}_{\mathsf{sen}}$. This seems to be an unnecessary step since the sender himself generates the tokens. Later we will see that modeling the sender this way simplifies our presentation of the proof significantly.

The receiver, on the other hand, consists of the algorithms ($\Pi.\mathsf{Preproc}_{\mathsf{rec}}$, $\Pi.\mathsf{EvalHT}_{\mathsf{rec}}$, $\Pi.\mathsf{Output}$). It takes as input y, randomness R_{rec} along with ($\mathsf{htokens}_{\mathsf{rec}}$, $\mathsf{Software}$, $\tilde{x}$) which it receives from the sender and does the following.

- It executes $\Pi.\mathsf{Preproc}_{\mathsf{rec}}$ on input $(y, R_{\mathsf{rec}}, \mathsf{Software}, \tilde{x})$ to obtain (q, state).

- It then executes $\Pi.\mathsf{EvalHT}_{\mathsf{rec}}$ by querying the tokens $\mathsf{htokens}_{\mathsf{rec}}$ on input q to obtain $\tilde{y}$. The i^{th} token in $\mathsf{htokens}_{\mathsf{rec}}$ is queried by the i^{th} bit of q to obtain the i^{th} bit of $\tilde{y}$.

- Finally, $\Pi.\mathsf{Output}$ is run on input $(\mathsf{state}, \tilde{y})$ to obtain z which is output by the receiver.

This completes the description of Π. The following lemma shows that there exists a $\mathsf{SplitState}$ protocol for a functionality if the functionality has a non-adaptive protocol.

Lemma 5. *Suppose a functionality f has a non-interactive and a non-adaptive protocol in the bit-OT token model. Then, there exists a* $\mathsf{SplitState}$ *protocol for the functionality f.*

Proof. Consider a non-adaptive protocol Π for a functionality f in the hardware token model. Without loss of generality, assume that it can be expressed as follows. The sender algorithm of Π, on input (x, R_{sen}), executes $\Pi.\mathsf{GenSwTok}$ to obtain $(\mathsf{sware}, \mathsf{htokens})$, where the length of sware is l_S and the number of tokens is l_T. It then sends $(\mathsf{sware}, \mathsf{htokens})$ across to the receiver. The receiver consists of $(\Pi.\mathsf{Preprocess}, \Pi.\mathsf{Evaluate}, \Pi.\mathsf{Output})$. It executes $\Pi.\mathsf{Preprocess}$ on input sware along with its input y and randomness R_{rec} to obtain query q. It then executes $\mathsf{Evaluate}$ on input q to obtain ans. Finally, $\Pi.\mathsf{Output}$ runs on input ans along with $(y, R_{\mathsf{rec}}, \mathsf{sware})$ to obtain the output of the functionality. For simplicity we modify $\Pi.\mathsf{GenSwTok}$ such that instead of outputting $\mathsf{htokens}$ it outputs the string contained in $\mathsf{htokens}$ which is defined to be the concatenation of all the bits contained in $\mathsf{htokens}$.

We construct a $\mathsf{SplitState}$ protocol Π' for f from Π. The sender of Π' consists of a tuple of algorithms ($\Pi'.\mathsf{InpFreePP}, \Pi'.\mathsf{Preproc}_{\mathsf{sen}}, \Pi'.\mathsf{EvalHT}_{\mathsf{sen}}$) which are defined as follows.

- $\Pi'.\mathsf{InpFreePP}$: On input R_{sen}, it executes as follows. It first constructs three types of tokens described below.

  - It constructs l_S tokens with each token containing 0 and 1 in the first and the second positions respectively. Recall that l_S is the size of the software output by $\Pi.\mathsf{GenSwTok}$. Denote this set of tokens by $\mathsf{htokens}^{(1)}_{\mathsf{sen}}$.
  - It then chooses $2l_T$ random bits $r_1, \ldots, r_{2l_T}$ from R_{sen}. Recall that l_T is the number of tokens output by $\Pi.\mathsf{GenSwTok}$. It generates $2l_T$ tokens are generated with the i^{th} token containing $(r_i \oplus 0, r_i \oplus 1)$. Denote this set of tokens by $\mathsf{htokens}^{(2)}_{\mathsf{sen}}$.

– It then constructs a set of l_T tokens with the first token containing (r_1, r_2), the second containing (r_3, r_4) and so on. Denote this set of tokens by $\mathsf{htokens}_{\mathsf{rec}}$.

The output of this algorithm is $(\mathsf{htokens}^{(1)}_{\mathsf{sen}}, \mathsf{htokens}^{(2)}_{\mathsf{sen}}, \mathsf{htokens}_{\mathsf{rec}})$. Note that this algorithm does not output any software.

- $\Pi'.\mathsf{Preproc}_{\mathsf{sen}}$: On input (x, R_{sen}) it executes as follows. The $\Pi'.\mathsf{Preproc}_{\mathsf{sen}}$ first executes $\Pi.\mathsf{GenSwTok}(x, R_{\mathsf{sen}})$ to obtain (sware, s), where s is the string contained in $\mathsf{htokens}$. The output of $\Pi'.\mathsf{Preproc}_{\mathsf{sen}}$ is (sware, s).

- $\Pi'.\mathsf{EvalHT}_{\mathsf{sen}}$: On input $((\mathsf{sware}, s), \mathsf{htokens}^{(1)}_{\mathsf{sen}}, \mathsf{htokens}^{(2)}_{\mathsf{sen}})$, it does the following. It queries $\mathsf{htokens}^{(1)}_{\mathsf{sen}}$ on input sware to obtain $\tilde{z}_1$. It then queries $\mathsf{htokens}^{(2)}_{\mathsf{sen}}$ on input s to obtain $\tilde{z}_2$.

The sender on input x and randomness R_{sen} first executes $\Pi'.\mathsf{InpFreePP}$ on input (x, R_{sen}) to obtain $(\mathsf{htokens}^{(1)}_{\mathsf{sen}}, \mathsf{htokens}^{(2)}_{\mathsf{sen}}, \mathsf{htokens}_{\mathsf{rec}})$. It then executes $\Pi'.\mathsf{Preproc}_{\mathsf{sen}}$ on input (x, R_{sen}) to obtain (sware, s). Finaly $\Pi'.\mathsf{EvalHT}$ is executed on input $((\mathsf{sware}, s), \mathsf{htokens}^{(1)}_{\mathsf{sen}}, \mathsf{htokens}^{(2)}_{\mathsf{sen}})$ to obtain $\tilde{z}_1$ and $\tilde{z}_2$. The sender sends to the receiver $(\tilde{z}_1, \tilde{z}_2)$ as the software and $\mathsf{htokens}_{\mathsf{rec}}$ as the hardware.

The receiver of Π' more or less behaves the same way as the receiver of Π. The receiver of Π' on input (y, R_{rec}) and upon receiving $(\mathsf{Software}, \mathsf{htokens})$ from the sender, does the following.

- Parse $\mathsf{Software}$ as $(\tilde{z}_1, \tilde{z}_2)$.

- It runs $\Pi.\mathsf{Preprocess}$ on input $(y, R_{\mathsf{rec}}, \tilde{z}_1)$ to obtain q.

- Query the tokens $\mathsf{htokens}$ on input q to obtain ans. Compute $\widetilde{ans}$ such that the i^{th} bit of $\widetilde{ans}$ is the XOR of the i^{th} bit of ans and the i^{th} bit of $\tilde{z}_2$.

- It then executes $\Pi.\mathsf{Output}$ on input $\widetilde{ans}$ to obtain the output of the functionality.

The proof of security of Π' can be more or less argued directly from the proof of security of Π. The main idea is that the simulator instead of giving the answers ans (as in the case of Π), first outputs a random string R as part of software. When the receiver submits its query it computes ans, using the simulator of Π, and then outputs $\widetilde{ans}$, where the i^{th} bit of $\widetilde{ans}$ is $\mathsf{ans}_i \oplus R_i$, where ans_i is the i^{th} bit of ans. The rest of the details of the simulator of Π' is the same as the simulator of Π. □

Whenever we say that a functionality has a protocol in the bit-OT token model we assume that it is a $\mathsf{SplitState}$ protocol. In the class of $\mathsf{SplitState}$ protocols, we further consider a special class of protocols which we term as *simplified protocols*.

Simplified protocols. These are $\mathsf{SplitState}$ protocols which have a trivial preprocessing algorithm on the sender's as well as receiver's inputs. In more detail, a protocol is said to be a *simplified protocol* if it is a $\mathsf{SplitState}$ protocol, and the sender's preprocessing algorithm $\mathsf{Preproc}_{\mathsf{sen}}$ as well as the receiver's preprocessing algorithm $\mathsf{Preproc}_{\mathsf{rec}}$ can be implemented by depth-0 circuits. Recall that depth-0 circuits which solely consists of wires and no gates. We now explore the relation between the simplified protocols and decomposable randomized encodings. We show, for every functionality, the equivalence of DRE and simplified protocols in the bit-OT token model.

Theorem 6. *There exists an efficient decomposable randomized encoding for a functionality f iff there exists a simplified protocol for f in the bit-OT token model.*

Proof. Consider a function f that takes as input of the form (x, y) whose total length is m, where x is of length m_x and y is of length m_y. Suppose there exists a decomposable randomized encoding for f, we construct a simplified protocol for f as follows. Let DRE for f consists of the following algorithms $(\mathsf{RE.Encode}, \mathsf{RE.ChooseInpWires}, \mathsf{RE.Decode})$. The sender of the simplified protocol, on input (x, R_{sen}), executes the following algorithms in order.

- $\mathsf{InpFreePP}$: On input randomness R_{sen} it first executes $\mathsf{RE.Encode}$ on input (x, R_{sen}) to obtain $\tilde{f}$ and state. Suppose that $\mathsf{state} = ((s_1^0, s_1^1), \ldots, (s_m^0, s_m^1))$. As a simplification, we assume that all s_i^b's are bits. The argument can be extended when s_i^b are strings. It composes the tokens as follows. The i^{th} token contains the pair of bits (s_i^0, s_i^1). Denote these tokens as $\mathsf{htokens}$. The tokens $\mathsf{htokens}$ can be further split into sender's tokens and receiver's tokens. The first m_x tokens of $\mathsf{htokens}$ is denoted by $\mathsf{htokens}_{\mathsf{sen}}$ (sender's tokens) and the rest of the m_y tokens is denoted by $\mathsf{htokens}_{\mathsf{rec}}$ (receiver's tokens). $\mathsf{InpFreePP}$ outputs $(\tilde{f}, \mathsf{htokens}_{\mathsf{sen}}, \mathsf{htokens}_{\mathsf{rec}})$.
- $\mathsf{Preproc}_{\mathsf{sen}}$: On input x and randomness R_{sen} it outputs x.
- $\mathsf{EvalHT}_{\mathsf{sen}}$: On input x from $\mathsf{Preproc}_{\mathsf{sen}}$ and $\mathsf{htokens}_{\mathsf{sen}}$ it does the following. It evaluates the i^{th} token in $\mathsf{htokens}_{\mathsf{sen}}$ using the i^{th} bit of x. Let $\tilde{x}$ be a concatenation of all the answers from the tokens. The output of this step is $\tilde{x}$.

The output of the sender is $(\mathsf{Software} = (\tilde{f}, \tilde{x}), \mathsf{htokens}_{\mathsf{rec}})$. We now proceed to describe the receiver. The receiver on input (y, R_{rec}) along with $(\mathsf{Software} = (\tilde{f}, \tilde{x}), \mathsf{htokens}_{\mathsf{rec}})$ which it receives from the sender, it does the following. It first executes $\mathsf{Preproc}_{\mathsf{rec}}$ which on input (y, R_{rec}) outputs y. It then executes $\mathsf{EvalHT}_{\mathsf{rec}}$ which evaluates the i^{th} token of $\mathsf{htokens}_{\mathsf{rec}}$ on input the i^{th} bit of y to obtain the i^{th} bit of $\tilde{y}$. The receiver then executes the Output algorithm which is described as follows. It takes as input $(\tilde{y}, R_{\mathsf{rec}}, \tilde{f}, \tilde{x})$ and executes $\mathsf{RE.Decode}(\tilde{f}, \tilde{x}, \tilde{y})$ to obtain z which is output by the receiver. Firstly, it can be seen that this is a simplified protocol. Further, the correctness and privacy properties of DRE respectively implies correctness and the security of the protocol.

We now prove the other direction. Suppose there exists a simplified protocol for f then we show that there exists a decomposable randomized encoding, denoted by $\mathsf{RE} = (\mathsf{Encode}, \mathsf{ChooseInpWires}, \mathsf{Decode})$, for f as follows. We first make modifications to the simplified protocol that makes the presentation of RE easier. The $\mathsf{InpFreePP}$ procedure is modified such that instead of outputting the tokens $\mathsf{htokens}_{\mathsf{sen}}$ and $\mathsf{htokens}_{\mathsf{rec}}$, outputs the string contained in them. We further use the fact that $\mathsf{Preproc}_{\mathsf{sen}}$ and $\mathsf{Preproc}_{\mathsf{rec}}$ are depth-0 circuits to modify them such that $\mathsf{Preproc}_{\mathsf{sen}}$, on input (x, R_{sen}), outputs x and $\mathsf{Preproc}_{\mathsf{rec}}$ on input (y, R_{rec}) outputs y. We now describe the RE procedure.

The $\mathsf{RE.Encode}$ procedure takes as input f (here we don't distinguish between a functionality and the circuit implementing it) and then executes $\mathsf{InpFreePP}(f, R_{\mathsf{sen}})$ to obtain $\tilde{f}$ and state (recall that $\mathsf{InpFreePP}$ is modified such that it outputs the string contained in the tokens instead of outputting the tokens.). That is, the $(2i-1)^{th}$ as well as the $2i^{th}$ bits in the string state are precisely the bits contained in the i^{th} token (Again, we are making a simplification here. This argument can be generalised if $(2i-1)^{th}$ as well as the $2i^{th}$ positions in state contains strings and not bits.). Further, the $\mathsf{RE.Decode}$ algorithm takes as input $\tilde{f}$ along with $\tilde{x}$ as well as $\tilde{y}$ and then it executes the Decode algorithm (of the receiver). The output of $\mathsf{RE.Decode}$ is essentially the output of the Decode algorithm of the receiver. It follows from the security of the simplified protocol that RE is a valid decomposable randomized encoding (and hence, a randomized encoding). □

Ishai et al. [30] show that there exists decomposable randomized encodings for all functions in NC^1. From this result and Theorem 6, the following corollary is immediate.

Corollary 7. *There exists a simplified protocol for all functions in* NC^1.

4.5 Main Theorem

We now state the following theorem that shows that every function that has a non-adaptive protocol in the bit-OT token model also has a simplified protocol. Essentially this theorem says the following. Let there be a non-adaptive protocol in the bit-OT token model for a function. Then, no matter how complex the preprocessing algorithm is in this protocol, we can transform this into another protocol which has a trivial preprocessing on its inputs. Since a function having a non-adaptive protocol also has a $\mathsf{SplitState}$ protocol from Lemma 5, we will instead consider $\mathsf{SplitState}$ protocols in the below theorem.

Theorem 8. *Suppose there exists a* $\mathsf{SplitState}$ *protocol for* f *in the bit-OT token model having* $p(k)$ *number of tokens, for some polynomial* p. *Then, there exists a simplified protocol for* f *in the bit-OT token model having* $O(p(k))$ *number of tokens.*

Proof. Consider the set S of all $\mathsf{SplitState}$ protocols for f each having $O(p(k))$ number of tokens. In this set S, consider the protocol Π'_{sen} having the least depth complexity of $\mathsf{Preproc}_{\mathsf{sen}}$. That is, protocol Π'_{sen} is such that the following quantity is satisfied.

$$\mathsf{Depth}(\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}) = \min_{\Pi \in S} \Big\{ \mathsf{Depth}(\Pi.\mathsf{Preproc}_{\mathsf{sen}}) \Big\}$$

We claim that the $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$ is a depth-0 circuit. If it is not a depth-0 circuit, then we arrive at a contradiction. We transform Π'_{sen} into Π''_{sen}, and show that $\mathsf{Depth}(\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}) < \mathsf{Depth}(\Pi''_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}})$. This would contradict the fact that the depth of $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$ is the least among all the protocols in S. To acheive the transformation, we first break $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$ into two circuits $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}$ and $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{sen}}$ such that, $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$ will first execute $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{sen}}$ and its output is fed into $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}$ whose output determines the output of $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$. Further, $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}$ consists of a single layer of the circuit and hence has depth 1 (If $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$ was just one layer to begin with then $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{sen}}$ would be a depth-0 circuit.). Then we define a functionality which executes the algorithms $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}$ and $\Pi'_{\mathsf{sen}}.\mathsf{EvalHT}_{\mathsf{sen}}$. We observe that this functionality can be realized by an NC^1 circuit. Then, we proceed to replace the procedures $\Pi'_{\mathsf{sen}}.\mathsf{EvalHT}_{\mathsf{sen}}$ and $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}$ by the sender algorithm of a simplified protocol defined for this functionality, the existence of which follows from Corollary 7. The $\mathsf{Preproc}_{\mathsf{sen}}$ of the resulting protocol just consists of $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{sen}}$ and this would contradict the choice of Π'_{sen}. We now proceed to the technical details.

The sender algorithm of Π'_{sen} can be written as ($\Pi'_{\mathsf{sen}}.\mathsf{InpFreePP}$, $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$, $\Pi'_{\mathsf{sen}}.\mathsf{EvalHT}_{\mathsf{sen}}$) and the receiver of Π'_{sen} can be written as ($\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{rec}}$, $\Pi'_{\mathsf{sen}}.\mathsf{EvalHT}_{\mathsf{rec}}$, $\Pi'_{\mathsf{sen}}.\mathsf{Output}$). The description of these algorithms are given in Section 4. Consider the following functionality, denoted by $f^{\mathsf{sen}}_{\mathsf{NC}^1}$.

$f^{\mathsf{sen}}_{\mathsf{NC}^1}(s, \mathsf{temp}_x; \perp)$:- On input (s, temp_x) from the sender, it first executes $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}(\mathsf{temp}_x)$ to obtain x'. It then parses s as $((s^0_1, s^1_1), \ldots, (s^0_m, s^1_m))$, where the size of x' is m. It then computes $\tilde{x} = (s^{x'_1}_1, \ldots, s^{x'_m}_m)$, where x'_i is the i^{th} bit of x'. Finally, output $\tilde{x}$. This functionality does not take any input from the receiver.

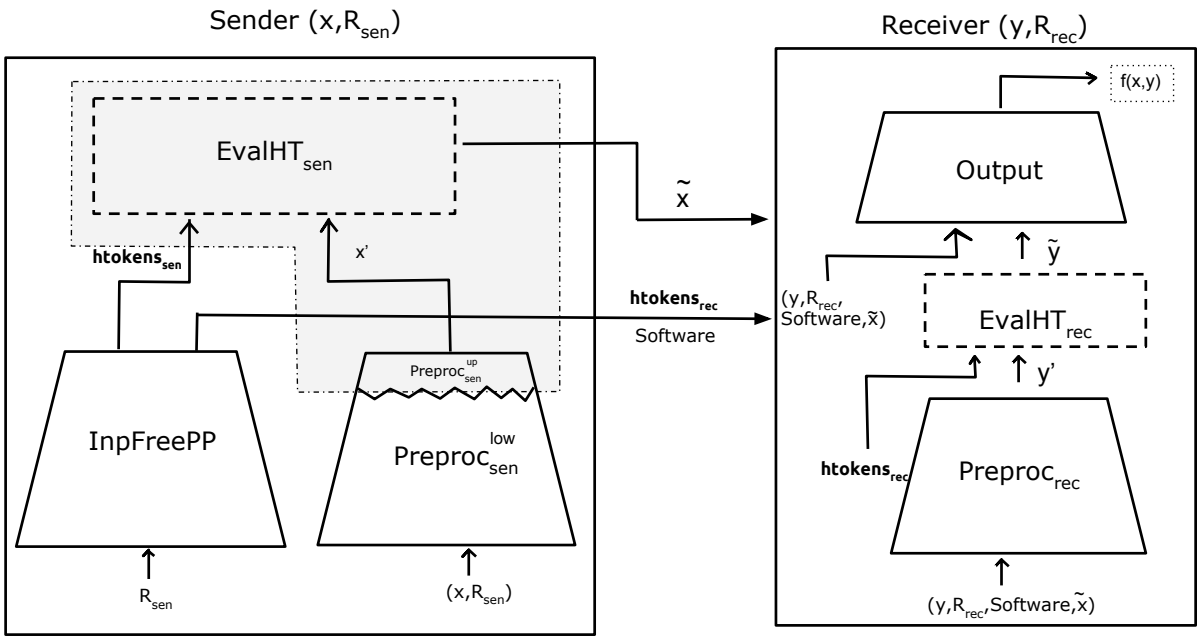

Figure 1: This represents the protocol Π'_{sen} and the shaded area denotes the functionality $f^{\mathsf{sen}}_{\mathsf{NC}^1}$.

Observe that $f^{\mathsf{sen}}_{\mathsf{NC}^1}$ is a NC^1 circuit and has a simplified protocol from Corollary 7. Let us call this protocol $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$. Since, the receiver's input is $\perp$, the sender algorithm in this protocol does not output any tokens [3]. We use Π'_{sen} and $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ to obtain Π''_{sen}. The protocol Π''_{sen} is described as follows.

Before we describe the sender algorithm of Π''_{sen}, we modify the sender of Π'_{sen} such that, the algorithm $\Pi'_{\mathsf{sen}}.\mathsf{InpFreePP}$ instead of outputting $\mathsf{htokens}_{\mathsf{rec}}$ just outputs s, which is nothing but the string contained in $\mathsf{htokens}_{\mathsf{sen}}$. The sender algorithm of Π''_{sen} on input (x, R_{sen}), does the following.

- It first executes $\Pi'_{\mathsf{sen}}.\mathsf{InpFreePP}(R_{\mathsf{sen}})$ to obtain $(\mathsf{Software}, s, \mathsf{htokens}_{\mathsf{rec}})$, where s, as described before is the string obtained by concatenating all the bits in $\mathsf{htokens}_{\mathsf{sen}}$.
- It then executes $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{sen}}$ on input (x, R_{sen}) to obtain temp_x.
- It then executes the sender algorithm of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ with input (s, temp_x). Let the output of this algorithm be $\mathsf{Software}^{\mathsf{NC}^1}$.
- Send $(\mathsf{Software}, \mathsf{Software}^{\mathsf{NC}^1}, \mathsf{htokens}_{\mathsf{rec}})$ across to the receiver (*recall that the sender of* $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ *does not output any tokens.*).

The receiver on input (y, R_{rec}) along with $(\mathsf{Software}, \mathsf{Software}^{\mathsf{NC}^1}, \mathsf{htokens}_{\mathsf{rec}})$ which it receives from the sender, does the following.

[3] From the Corollary 7 and Ishai et al. [30], the simplified protocols defined for NC^1 functionalities are such that the sender does not send any tokens to the receiver if the receiver does not have any input.

- It executes the receiver algorithm of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ on input $\mathsf{Software}^{\mathsf{NC}^1}$ as well as its internal randomness to obtain $\tilde{x}$. Note that the receiver of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ does not have its own input.
- It then executes the receiver algorithm of Π'_{sen} on input $(y, R_{\mathsf{rec}}, \mathsf{Software}, \tilde{x}, \mathsf{htokens}_{\mathsf{rec}})$. Let the output of this algorithm be out.
- Output out.

We show, in Figure 2, how we replace $\mathsf{EvalHT}_{\mathsf{sen}}$ and $\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{sen}}$ in Π'_{sen} by the protocol $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ to obtain the protocol Π''_{sen}. We give the final decription of Π''_{sen} in Figure 3 where we expand out the sender and the receiver algorithms of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$.

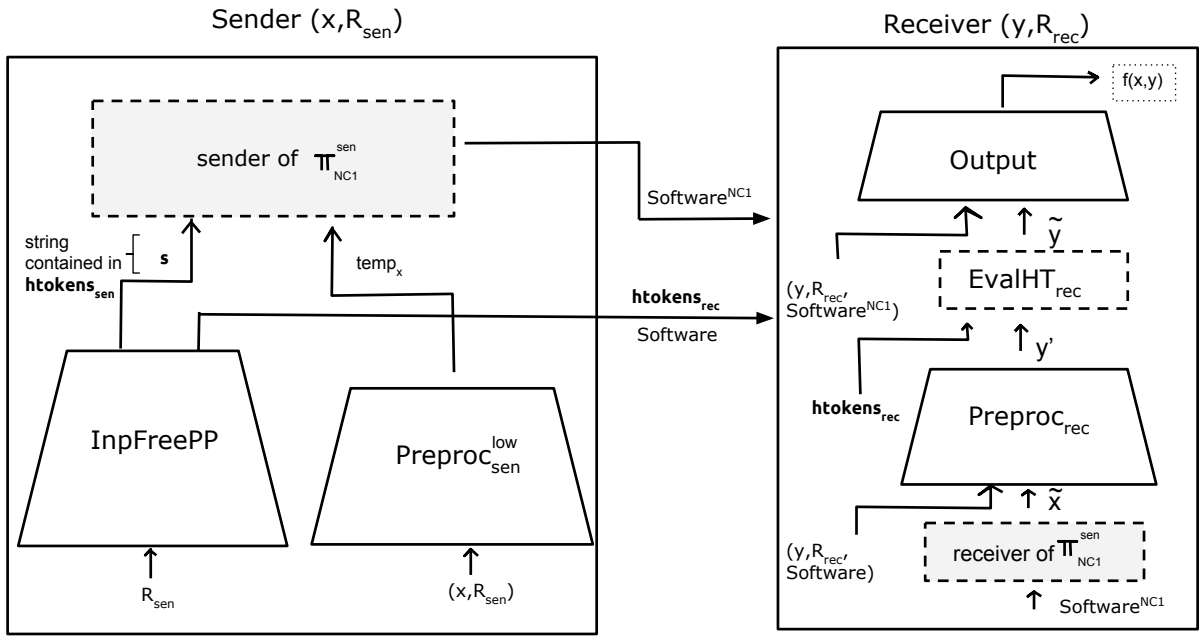

Figure 2: In this figure, the shaded area in Figure 1 is replaced by the protocol $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$.

We first claim that the protocol Π''_{sen} satisfies the correctness property. This follows directly from the correctness of the protocols Π'_{sen} and $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$. The security of the above protocol is proved in the following lemma.

Lemma 9. *Assuming that the protocol Π'_{sen} and $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ is secure, the protocol Π''_{sen} is secure.*

Proof Sketch. To prove this, we need to construct a simulator $\mathsf{Sim}_{\Pi''_{\mathsf{sen}}}$, such that the output of the simulator is indistinguishable from the output of the sender of Π''_{sen}. To do this we use the simulators of the protocols Π'_{sen} and $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ which are denoted by $\mathsf{Sim}_{\Pi'_{\mathsf{sen}}}$ and $\mathsf{Sim}_{\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}}$ respectively.

The simulator $\mathsf{Sim}_{\Pi''_{\mathsf{sen}}}$ on input out, which is the output of the functionality f, along with y' which is the query made by the receiver to the OT tokens does the following. It first executes $\mathsf{Sim}_{\Pi'_{\mathsf{sen}}}(\mathsf{out}, y')$ to obtain $(\mathsf{Software}, \tilde{x}, \tilde{y})$. Then, $\mathsf{Sim}_{\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}}$ on input $\tilde{x}$ is executed to obtain $\mathsf{Software}^{\mathsf{NC}^1}$. The output of $\mathsf{Sim}_{\Pi''_{\mathsf{sen}}}$ is $(\mathsf{Software}, \mathsf{Software}^{\mathsf{NC}^1}, \tilde{y})$. By standard hybrid arguments, it

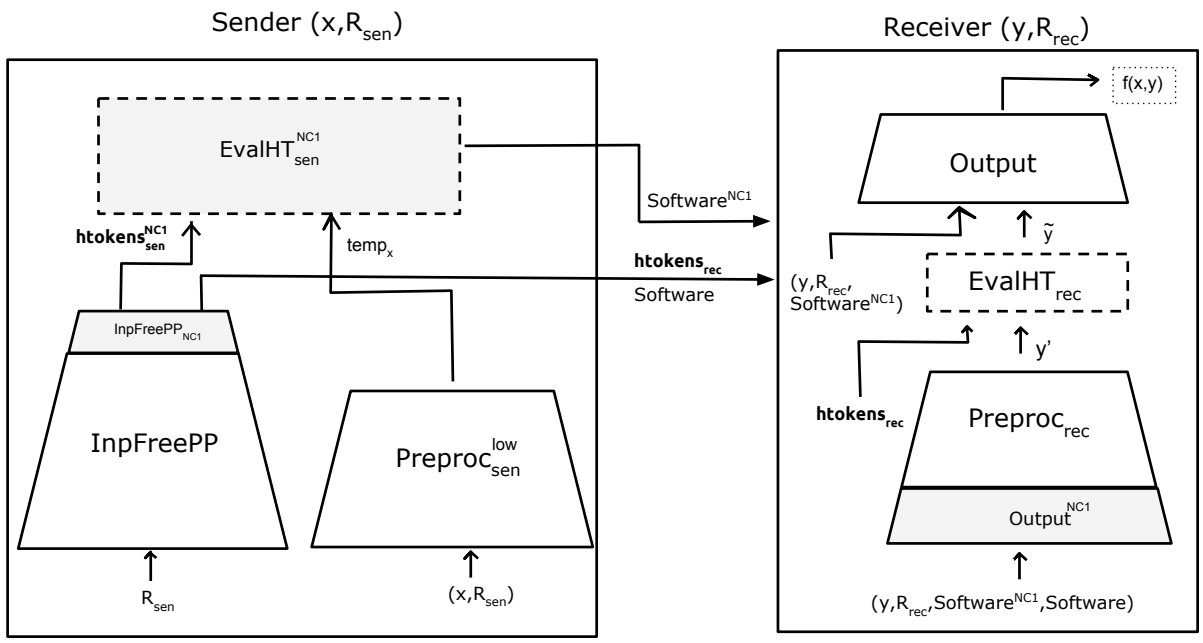

Figure 3: This figure depicts the protocol Π''_{sen} after expanding out the procedures in the sender and the receiver algorithms of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$. The algorithms $\mathsf{InpFreePP}_{\mathsf{NC}^1}$, $\mathsf{EvalHT}^{\mathsf{NC}^1}_{\mathsf{sen}}$ (which are shaded) are part of the sender of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$. Since, $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ is a simplified protocol, its $\mathsf{Preproc}_{\mathsf{sen}}$ of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ is a depth-0 circuit. Further, the procedure $\mathsf{Output}^{\mathsf{NC}^1}$ (shaded) is part of the receiver of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$. Since the sender of the protocol $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ does not output any tokens the receiver of $\Pi^{\mathsf{sen}}_{\mathsf{NC}^1}$ consists of just the algorithm $\mathsf{Output}_{\mathsf{NC}^1}$.

can be shown that the output of the simulator $\mathsf{Sim}_{\Pi''_{\mathsf{sen}}}$ is indistinguishable from the output of the sender of Π''_{sen}.

The above lemma proves that Π''_{sen} is a secure protocol for f. We claim that the number of tokens in Π''_{sen} is $O(p(k))$. This follows directly from the fact that the number of tokens output by the sender of Π'_{sen} is the same as the number of tokens output by Π''_{sen}. And hence, the number of tokens output by the sender of Π''_{sen} is $O(p(k))$. Further, the the depth of $\mathsf{Preproc}_{\mathsf{sen}}$ of Π''_{sen} is strictly smaller than the depth of $\Pi'_{\mathsf{sen}}.\mathsf{Preproc}_{\mathsf{sen}}$. This contradicts the choice of Π'_{sen} and so, the $\mathsf{Preproc}_{\mathsf{sen}}$ algorithm of Π'_{sen} is a depth-0 circuit.

Now, consider a set of protocols, $S' \subset S$ such that the $\mathsf{Preproc}_{\mathsf{sen}}$ algorithms of all the protocols in S' are implementable by depth-0 circuits. From the above arguments, we know that there is at least one such protocol in this set. We claim that there exists one such protocol in S whose $\mathsf{Preproc}_{\mathsf{rec}}$ algorithm is implementable by a depth-0 circuit. The argument for this is similar to the argument for the previous case. Hence, we will highlight only those places where the argument changes. Consider a protocol in S', denoted by Π'_{rec} such that the depth of $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}_{\mathsf{rec}}$ is at most the depth of $\mathsf{Preproc}_{\mathsf{rec}}$ of any protocol in S'. We then transform Π'_{rec} into another protocol Π''_{rec} such that the depth of $\Pi''_{\mathsf{rec}}.\mathsf{Preproc}_{\mathsf{rec}}$ is strictly less than the depth of $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}_{\mathsf{rec}}$

which contradicts the choice of Π'_{rec}. The transformation is as follows. We split $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}_{\mathsf{rec}}$ into $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{rec}}$ and $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{rec}}$. Also, $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{rec}}$ consists of a single layer of gates. Now consider the following functionality.

$f^{\mathsf{rec}}_{\mathsf{NC}^1}(\mathsf{str}; \mathsf{temp}_y)$: On input $(\mathsf{str}; \mathsf{temp}_y)$, where str is the sender's input and temp_y is the receiver's input, execute $\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{rec}}$ on input temp_y to obtain y'. Then, parse str as $((\mathsf{str}^0_1, \mathsf{str}^1_1), \ldots, (\mathsf{str}^0_l, \mathsf{str}^1_l))$, where l denotes the length of y'. Let $\tilde{y}$ be $(\mathsf{str}^{y'_1}_1, \ldots, \mathsf{str}^{y'_l}_l)$, where y'_i is the i^{th} bit of y'. Output $\tilde{y}$.

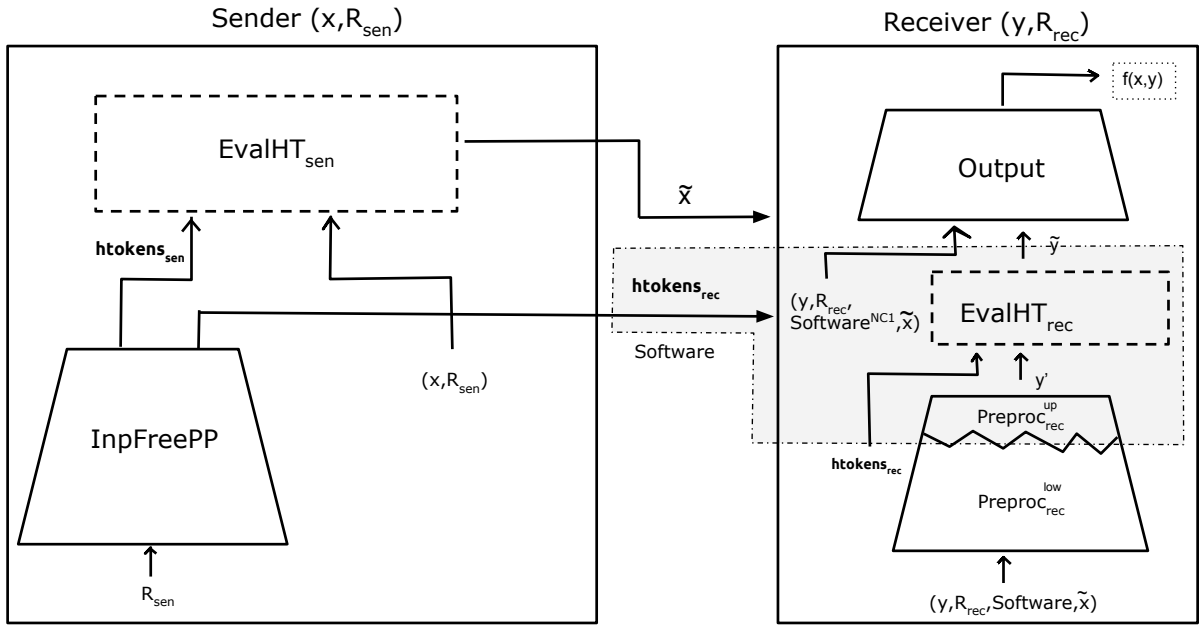

Figure 4: This represents the protocol Π'_{rec} and the shaded area denotes the functionality $f^{\mathsf{rec}}_{\mathsf{NC}^1}$.

Since, $f^{\mathsf{rec}}_{\mathsf{NC}^1}$ is a NC^1 circuit, from Corollary 7 there exists a simplified protocol for $f^{\mathsf{rec}}_{\mathsf{NC}^1}$, denoted by $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$. We use Π'_{rec} and $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ to obtain Π''_{rec}. The protocol Π''_{rec} is described as follows.

Before we describe the sender of Π''_{rec}, we modify the sender of Π'_{rec} such that the $\Pi'_{\mathsf{rec}}.\mathsf{InpFreePP}$, instead of outputting $\mathsf{htokens}_{\mathsf{rec}}$ outputs a string s which is the concatenation of the bits in $\mathsf{htokens}_{\mathsf{rec}}$. The sender algorithm of Π''_{rec} on input (x, R_{sen}), does the following.

- It first executes $\Pi'_{\mathsf{rec}}.\mathsf{InpFreePP}(R_{\mathsf{sen}})$ to obtain $(\mathsf{Software}, \mathsf{htokens}_{\mathsf{sen}}, s)$.
- It then executes $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}_{\mathsf{sen}}(R_{\mathsf{sen}})$ (which is a depth-0 circuit) on input (x, R_{sen}) to obtain x'.
- It then executes $\Pi'_{\mathsf{rec}}.\mathsf{EvalHT}_{\mathsf{sen}}$ on input $\mathsf{htokens}_{\mathsf{sen}}$ and x' to obtain $\tilde{x}$.
- It then executes the sender algorithm of $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ with input s. Let the output of this algorithm be $(\mathsf{Software}^{\mathsf{NC}^1}, \mathsf{htokens}^{\mathsf{NC}^1}_{\mathsf{rec}})$.

- Send $(\mathsf{Software}, \tilde{x}, \mathsf{Software}^{\mathsf{NC}^1}, \mathsf{htokens}^{\mathsf{NC}^1}_{\mathsf{rec}})$ across to the receiver.

The receiver on input $(\mathsf{Software}, \tilde{x}, \mathsf{Software}^{\mathsf{NC}^1}, \mathsf{htokens}^{\mathsf{NC}^1}_{\mathsf{rec}})$, does the following.

- It executes $\mathsf{Preproc}^{\mathsf{low}}_{\mathsf{rec}}$ of Π'_{rec} on input $(y, R_{\mathsf{rec}}, \mathsf{Software}, \tilde{x})$ to obtain temp_y.
- It then executes the receiver algorithm of $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ on input $(\mathsf{temp}_y,\ R_{\mathsf{rec}},\ \mathsf{Software}^{\mathsf{NC}^1},\ \tilde{x}^{\mathsf{NC}^1},\ \mathsf{htokens}^{\mathsf{NC}^1}_{\mathsf{rec}})$ to obtain $\tilde{y}$.
- Finally, the Output algorithm of Π'_{rec} is executed to obtain out, which is output by the receiver.

In Figure 5, we show how we replace $\mathsf{EvalHT}_{\mathsf{rec}}$ and $\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{rec}}$ in Π' by the protocol $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ to obtain the protocol Π''_{rec}. The final description of Π''_{rec} is given in Figure 6 where have expanded out the sender and the receiver algorithms of $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$.

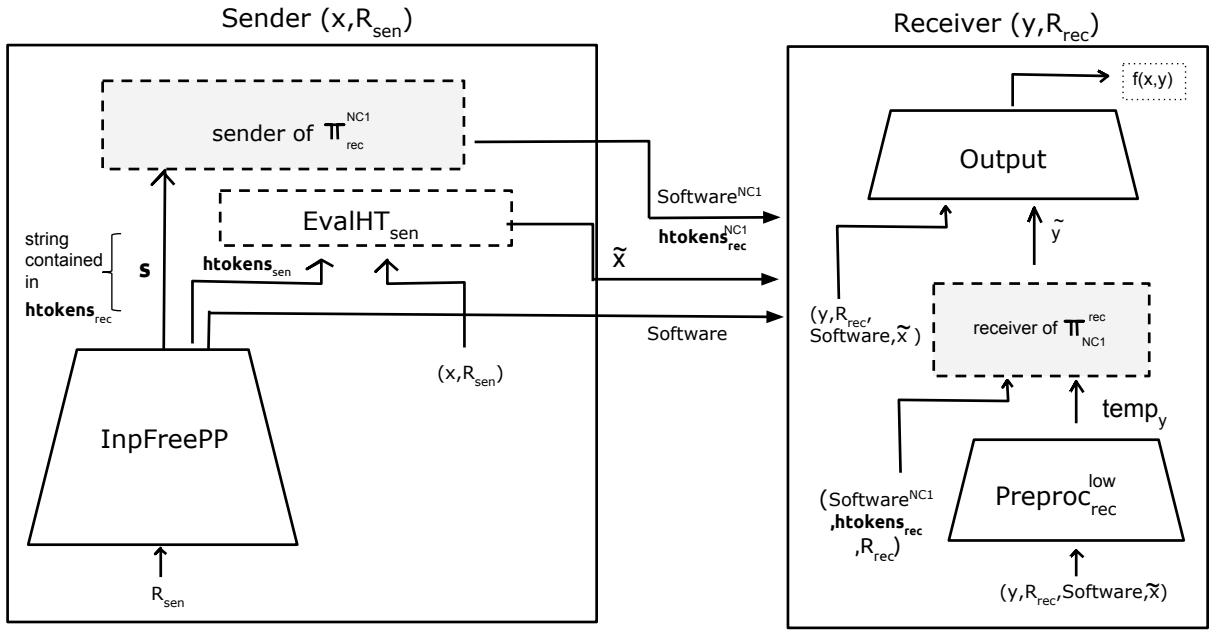

Figure 5: In this figure, the shaded area in Figure 4 is replaced by the protocol $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$.

As before, the correctness follows immediately. The following lemma proves the security of the above protocol.

Lemma 10. *Assuming that the protocol Π'_{rec} and $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ is secure, the protocol Π''_{rec} is secure.*

Proof Sketch. To prove this, we need to construct a simulator $\mathsf{Sim}_{\Pi''_{\mathsf{rec}}}$, such that the output of the simulator is indistinguishable from the output of the sender of Π''_{rec}. To do this we use the simulators of the protocols Π'_{rec} and $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ which are denoted by $\mathsf{Sim}_{\Pi'_{\mathsf{rec}}}$ and $\mathsf{Sim}_{\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}}$ respectively.

Before we construct the simulator, we first recall that the query to the tokens in Π''_{rec} is the output of $\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{rec}}$. The simulator $\mathsf{Sim}_{\Pi''_{\mathsf{rec}}}$ on input out, which is the output of the functionality f, along with temp_y which is the query made by the receiver of Π''_{rec} to the tokens does the following.

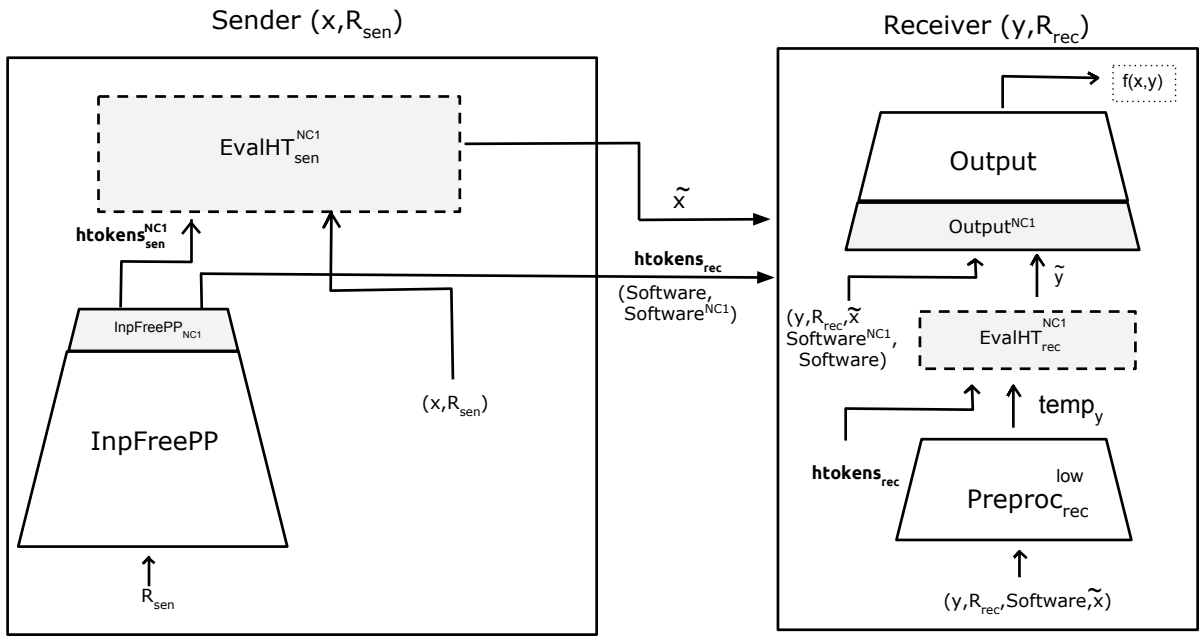

Figure 6: This figure depicts the protocol Π''_{rec}, after expanding out the procedures in the sender and the receiver algorithms of $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$. The algorithm $\mathsf{InpFreePP}_{\mathsf{NC}^1}$ (shaded) is part of the sender of $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$. Since, $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$ is a simplified protocol, its $\mathsf{Preproc}_{\mathsf{rec}}$ algorithm is a depth-0 circuit. Further, the procedures $\mathsf{EvalHT}^{\mathsf{NC}^1}_{\mathsf{rec}}$ and $\mathsf{Output}^{\mathsf{NC}^1}$ (shaded) are part of the receiver of $\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}$.

It first executes $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}^{\mathsf{up}}_{\mathsf{rec}}$ on input temp_y to obtain y'. It then executes $\mathsf{Sim}_{\Pi'_{\mathsf{rec}}}(\mathsf{out}, y')$ to obtain $(\mathsf{Software}, \tilde{x}, \tilde{y}^{\mathsf{NC}^1})$. Then, $\mathsf{Sim}_{\Pi^{\mathsf{rec}}_{\mathsf{NC}^1}}$ is run on input $(\mathsf{temp}_y, \tilde{y}^{\mathsf{NC}^1})$ to obtain $(\mathsf{Software}^{\mathsf{NC}^1}, \tilde{y})$. The output of $\mathsf{Sim}_{\Pi''_{\mathsf{rec}}}$ is $(\mathsf{Software}, \tilde{x}, \mathsf{Software}^{\mathsf{NC}^1}, \tilde{y}^{\mathsf{NC}^1})$. By standard hybrid arguments, it can be shown that the output of the simulator $\mathsf{Sim}_{\Pi''_{\mathsf{rec}}}$ is indistinguishable from the output of the sender of Π''_{rec}.

The above lemma proves that Π''_{rec} is a secure protocol for f. Further, the number of tokens in Π''_{rec} is $O(p(k))$. To see this, we observe that the number of the tokens in the simplified protocol for a NC^1 functionality F obtained from Corollary 7 [30] is $2^d|F|$, where $|F|$ is the size of the circuit implementing F and d denotes the depth of the circuit. Since, $f^{\mathsf{rec}}_{\mathsf{NC}^1}$ can be implemented by a constant depth circuit with size $O(p(k))$, the number of tokens in $\Pi^{\mathsf{NC}^1}_{\mathsf{rec}}$, and hence Π''_{rec}, is $O(p(k))$. Further, note that the depth of $\mathsf{Preproc}_{\mathsf{rec}}$ of Π''_{rec} is strictly smaller than the depth of $\Pi'_{\mathsf{rec}}.\mathsf{Preproc}_{\mathsf{rec}}$. This contradicts the choice of Π'_{rec}, thus showing that $\mathsf{Preproc}_{\mathsf{rec}}$ algorithm of Π'_{rec} is a depth-0 circuit. Moreover, the $\mathsf{Preproc}_{\mathsf{sen}}$ algorithm of Π'_{rec} is also a depth-0 circuit, which proves that Π'_{rec} is a simplified protocol. □

We now show that the existence of a non-adaptive protocol for a function implies the existence of a decomposable randomized encoding for that function. Suppose there exists a non-interactive and a non-adaptive protocol for f in the bit-OT token model. Then, from Theorem 8 it follows

that there exists a simplified protocol for f. Further, from Theorem 6, it follows that there exists a DRE, and hence an efficient randomized encoding for f. Summarising, we have the following.

Theorem 11. *If there exists a non-interactive and a non-adaptive protocol in the bit-OT token model for a function f then there exists an efficient decomposable randomized encoding for f.*

5 Lower Bound for Obfuscation Complete Oracle Schemes

In this section, we study the notion of an obfuscation complete oracle scheme. Roughly speaking, an obfuscation complete oracle scheme consists of an oracle generation algorithm whose execution results in: (a) a secret obfuscation complete circuit (whose size is only dependent on the security parameter), and, (b) a public obfuscation function. We call an oracle implementing the functionality of the secret obfuscation complete circuit an obfuscation complete (OC) oracle. The public obfuscation function can be applied on any desired (polynomial size) circuit to produce an *obfuscated oracle circuit*. This oracle circuit would make calls to the OC oracle during its execution. The OC oracle implements a fixed functionality and cannot keep any state specific to the execution of any obfuscated program. Informally, our security requirement is that for every polynomial size circuit C, whatever can be computed given access to the obfuscated oracle circuit for C and the OC oracle, can also be computed just given access to an oracle implementing the functionality of C. An obfuscation complete oracle scheme is formally defined as follows.

Definition 5. *A secure obfuscation complete oracle scheme consists of a randomized algorithm OracleGen called the oracle generation algorithm such that an execution $\mathsf{OracleGen}(1^\kappa)$ (where κ denotes the security parameter) results in a tuple $(T, \mathcal{O}^{\mathcal{T}})$. The string T is the description of the circuit called the secret obfuscation complete circuit while $\mathcal{O}^{\mathcal{T}}$ is a function (or the description of a Turing machine) called the public obfuscation function.*[4] *The tuple $(T, \mathcal{O}^{\mathcal{T}})$ has the following properties:*

1. ***Preserve Functionality.*** *The application of the function $\mathcal{O}^{\mathcal{T}}(\cdot)$ to a circuit C results in an* obfuscated oracle circuit *$\mathcal{O}^{\mathcal{T}}(C)$ (which during execution might make calls to the oracle $\mathcal{T}$ implementing the functionality T). We require the obfuscated oracle circuit $\mathcal{O}^{\mathcal{T}}(C)$ to have the same functionality as the circuit C. In other words, $\forall C, \forall x$, we must have:*

$$\mathcal{O}^{\mathcal{T}}(C) = C(x)$$

2. ***Polynomial Slowdown.*** *There exist polynomials $p(\cdot,\cdot)$ and $q(\cdot)$ such that for sufficiently large κ and $|C|$, we have:*

$$|\mathcal{O}^{\mathcal{T}}(C)| \leq p(|C|, \kappa), \quad \text{and}, \quad |T| \leq q(\kappa)$$

   Observe that the size of the circuit T is dependent only on the security parameter.

3. ***Virtual Black Box.*** *For every PPT adversary A, there exists a PPT simulator Sim and a negligible function $\mathsf{negl}(\cdot)$ such that for every PPT distinguisher D, for every circuit C and for every polynomial size auxiliary input z:*

[4] The modeling of T as a circuit rather than a Turing machine is to reflect the fact that given the security parameter, the size and the running time of T is fixed and it handles inputs of fixed size (so that T can, for example, be implemented in a small tamper proof hardware token).

$$Pr[D(A^{\mathcal{T}}(\mathcal{O}^{\mathcal{T}}(C), z), z) = 1] - Pr[D(Sim^{\mathcal{C}}(1^{|C|}, T, z), z) = 1] \leq negl(\kappa)$$

In other words, we require the output distribution of the adversary A and that of the simulator Sim to be computationally indistinguishable.

By replacing the above virtual black box definition by the "predicate" virtual black box definition used by Barak et al. (see [1] for more details), we obtain a relaxed security notion for obfuscation complete oracles schemes. This relaxed version will be used for our lower bounds.

5.1 Lower Bounds

In Section 6.2.2 [26] (full version), Goyal et al. construct an obfuscation complete oracle scheme in the $\mathcal{F}_{wrap}^{stateless}$-hybrid model[5]. In their scheme, if the size of original circuit is $|C|$, then the obfuscated oracle circuit makes $O(|C| \cdot \log(|C|))$ calls to the OC oracle, which is embedded inside a stateless token. Thus, a natural question is: "Do there exist obfuscation complete oracles schemes for which the above query complexity is lower?" Towards that end, we show a lower bound which rules out obfuscation complete oracles schemes where this query complexity is a constant.

5.1.1 Turing machines

We start by proving the lower bound result for the case of Turing machines. While this case is significantly simpler, it would already illustrate the fundamental limitations of OC Oracle schemes with low query complexity. For an OC scheme, denote by $Q(|M|)$ the number of queries the obfuscated Oracle Turing machine $\mathcal{O}^{\mathcal{T}}(M)$ makes to the Oracle $\mathcal{T}$. We now have the following theorem.

Theorem 12. *For every constant q, there does not exist any obfuscation complete oracle scheme such that for every Turing machine M, query complexity $Q(|M|) \leq q$.*

Proof. We prove the above theorem by contradiction. Assume that there exists such an OC Oracle scheme would query complexity $Q(|M|) \leq q$. Let the size of response to a query to the Oracle $\mathcal{T}$ be bounded by $p(k)$. Hence, observe that the information "flowing" from the Oracle $\mathcal{T}$ to the obfuscated Oracle TM $\mathcal{O}^{\mathcal{T}}(M)$ is bounded by $q \cdot p(k)$. We will show that this communication between the Oracle and the obfuscated TM is not sufficient for successful simulation. Let $f_1 : \{0,1\}^{\leq \text{poly(k)}} \rightarrow \{0,1\}^{q \cdot p(k)+k}$ and $f_2 : \{0,1\}^{\leq \text{poly(k)}} \rightarrow \{0,1\}^{\text{poly(k)}}$ denote functions drawn from a pseudorandom function ensemble. Now define a functionality $F_{f_1,f_2,s}(.,.)$ as follows. For $b \in \{1,2\}$, we have $F_{f_1,f_2,s}(b,x) = f_b(x)$. For $b = 3$ (referred to as mode 3), we interpret the input x as the description of an *Oracle* TM M and a sequence of q strings $a_1, \ldots, a_q$. The function outputs $\perp$ if there exists an i s.t. $|a_i| > p(k)$. Otherwise, run the machine $M(1, f_2(M))$. When the machine makes the ith Oracle query, supply a_i as the response (irrespective of what the query is). Now, if $M(1, f_2(M)) = f_1(f_2(M))$, output s, else output $\perp$. To summarize, check if the Oracle TM behaves like the PRF f_1 on a random point (determined by applying PRF f_2 on the description of the machine) and if so, output the secret s. (The function $F_{f_1,f_2,s}(.,.)$ is actually uncomputable.

[5] They actually construct a secure protocol for stateless oblivious reactive functionalities. However, it is easy to see that the same protocol gives us an obfuscation complete oracle scheme.

However, similar to [1], we can truncate the execution after poly(k) steps and output 0 if M does not halt.) Denote the obfuscated Oracle TM for this function as $\mathcal{O}^{\mathcal{T}}(F_{f_1,f_2,s})$.

Consider the real world when the adversary is given access to description of the Oracle TM $M' = \mathcal{O}^{\mathcal{T}}(F_{f_1,f_2,s})$ and is allowed to query the Oracle $\mathcal{T}$. In this case, the adversary can recover s as follows. First recover $d = M'(2, M')$ (by simply running the obfuscated Oracle TM M' on its own description string with the help of the Oracle $\mathcal{T}$). Now the adversary executes $M'(1, d)$ and stores responses of $\mathcal{T}$ to all the queries made by $M'(1, d)$. Call the responses $a_1, \ldots, a_q$. Finally, prepare a string x containing the description of M' along with the strings $a_1, \ldots, a_q$ and execute $M'(3, x)$. M' will in turn execute $M'(1, d)$ using $a_1, \ldots, a_q$ and, by construction, will get $f_1(f_2(M'))$. Thus, the adversary will receive s as output. Hence, we have constructed a real word adversary A such that:

$$\Pr[A^{\mathcal{T}}(\mathcal{O}^{\mathcal{T}}(F_{f_1,f_2,0})) = 1] - \Pr[A^{\mathcal{T}}(\mathcal{O}^{\mathcal{T}}(F_{f_1,f_2,1})) = 1] = 1 \tag{4}$$

Now consider the ideal world where the adversary S only has Oracle access to the functionality $F_{f_1,f_2,s}$. For simplicity, we first consider the hybrid ideal world where the functions f_1 and f_2 are truly random (that is, for each input, there exists a truly random string which is given as the output). Without loss of generality, we assume that S does not query $F_{f_1,f_2,s}$ multiple times with the same input. Consider a query $(2, M)$ to the functionality $F_{f_1,f_2,s}$. Then it is easy to see that, except with negligible probability, S has not issued the query $(1, f_2(M))$ so far (where the probability is taken over the choice of truly random function f_2). Now when $M(1, f_2(M))$ is executed, depending upon how the Oracle queries are answered, the total number of possible outputs is $2^{q \cdot p(k)}$. Lets call this output set S_o. The probability (taken over the choice of f_1) that $f_1(f_2(M)) \in S_o$ can be bounded by $\frac{1}{2^k}$ ($= \frac{|S_o|}{2^{|f_1(f_2(M))|}}$) which is negligible. Thus, when S queries with $(3, M||a_1|| \ldots ||a_q)$, except with negligible probability, it will get $\perp$ as the output no matter what $a_1, \ldots, a_q$ are. By a straightforward union bound, it can be seen that except with negligible probability, all the queries of S in mode 3 will result in $\perp$ as the output (as opposed to s). By relying on the pseudorandomness of f_1 and f_2, this will also be true not only in the hybrid ideal world but also in the actual ideal world. Hence we have shown that for all ideal world adversaries S,

$$\Pr[S^{F_{f_1,f_2,0}}(1^k) = 1] - \Pr[S^{F_{f_1,f_2,1}}(1^k) = 1] \leq \text{negl}(k) \tag{5}$$

Combining equations 4 and 5, we get a contradiction with the relaxed virtual black box property (see the predicate based virtual black box property in [1]) of the OC Oracle scheme. □

5.1.2 Circuits

In extending the impossibility result to the case of circuits, the basic problem is that since the input length of the circuit is fixed, it may not be possible to execute a circuit on its own description. To overcome this problem, [1] suggested a functionality "implementing homomorphic encryption". This allowed the functionality to let the user (or adversary) evaluate a circuit "gate by gate" (as opposed to feeding the entire circuit at once) and still test certain properties of the user circuit. These techniques do not directly generalize to our setting. This is because in our setting, the Oracle queries made by the adversary's circuit will have to be seen and answered by the adversary. This might leak the input on which the circuit is being "tested" by the functionality. Thus, once the adversary knows the input and hence the "right output", he might, for example, try to change the

circuit or tamper with intermediate encrypted wire values to convince the functionality that the circuit is giving the right output. We use the techniques developed in Section 6.2.2 [26] to overcome these problems. Note that these problems do not arise in the setting of Barak et al [1] since there the adversary never gets to see the input on which his circuit is being tested (and hence cannot pass the test even if he can freely force the circuit to give any output of his choice at any time). We now state and prove our impossibility results for circuits.

Theorem 13. *For every constant q, there does not exist any obfuscation complete oracle scheme such that for every circuit C, query complexity $Q(|C|) \leq q$.*

Proof. We only provide a sketch of the proof (details follow from the ideas in Theorem 12, and Sections 6.2.2 and 6.2.3 in the full version of [26]). Again, assume that there exists such an OC Oracle scheme with query complexity $Q(|C|) \leq q$. Let the size of response to a query to the Oracle $\mathcal{T}$ be bounded by $p(k)$ s.t. the information flowing from the Oracle $\mathcal{T}$ to the obfuscated Oracle circuit $\mathcal{O}^{\mathcal{T}}(C)$ is bounded by $q \cdot p(k)$. Let $f_1 : \{0,1\}^{\leq \text{poly(k)}} \rightarrow \{0,1\}^{q \cdot p(k)+k}$, $f_2 : \{0,1\}^{\leq \text{poly(k)}} \rightarrow \{0,1\}^{\text{poly(k)}}$ and $f_3 : \{0,1\}^{\leq \text{poly(k)}} \rightarrow \{0,1\}^{\text{poly(k)}}$ denote functions drawn from a pseudorandom function ensemble. Now define a functionality $F_{f_1,f_2,f_3,k,s}(.,.)$ as follows. For $b \in \{1,2\}$, we have $F_{f_1,f_2,f_3,k,s}(b,x) = f_b(x)$. In mode 3 (i.e., $b = 3$), as before, the function enables a user to feed in an Oracle circuit, get it tested on a random input and if the output is correct, get the secret s. This is done by enabling various types of queries in this mode. Below, we describe how a user can get an Oracle circuit C of his choice tested by the functionality (and describe queries handled by the functionality in mode 3 as we go along). Without loss of generality, we assume that C consists only of fan-in two NAND gates and oracle gates.

- User starts with an Oracle circuit $C : \{0,1\}^m \rightarrow \{0,1\}^n$. Uniquely number each wire in C such that the m input wires are assigned numbers from 1 to m and n output wires are assigned numbers from $m+1$ to $m+n$. Each non-Oracle gate in C is uniquely identified by a tuple (a,b,c) where a and b are the index numbers of the input wires and c is the index number of the output wire. Sort these tuples in the increasing order of c and create a hash chain as:
$$H_i = H(H_{i-1}, a, b, c),$$
where H is a collision resistant hash function (CRHF) and H_0 is an all zero string of appropriate length. Assuming there are g non-Oracle gates in C, the result will be H_g. Note that w.l.g., each Oracle gate in C has fan out $p(k)$. Denote the index numbers of the output wires of the ith Oracle gate by $w_i^1, \ldots, w_i^{p(k)}$. Query the functionality $F_{f_1,f_2,f_3,k,s}$ with $(2, H_g||m||n||(|C|)\ ||w_1^1|| \ldots ||w_q^{p(k)})$ to get $d = f_2(H_g||m||n||(|C|)||w_1^1|| \ldots ||w_q^{p(k)})$. The user now executes the Oracle circuit C on input $(1,d)$ and stores responses of $\mathcal{T}$ to all the queries made by $C(1,d)$. Call the responses $a_1, \ldots, a_q$.

- The user now queries to functionality in mode 3 with $H_g, m, n, |C|, \{w_i^1, \ldots, w_i^{p(k)}\}_{i \in [q]}, a_1, \ldots, a_q$. The basic idea is that given these values, the user is committed to the value on each wire in the Oracle circuit C. $F_{f_1,f_2,f_3,k,s}$ sets $E_{ID} = f_3(H_g||m||n||(|C|)||w_1^1|| \ldots ||w_i^{p(k)}\ ||a_1|| \ldots ||a_q)$. Let $EM_k(m)$ denote the authenticated encryption of the message m with the secret key k (using, e.g., encrypting then including a MAC of the ciphertext). Similar in spirit to the construction in Section 6.2.2 in [26], the functionality $F_{f_1,f_2,f_3,k,s}$ gives out an authenticated encryption for each of the input wires. That is, for $i \in [m]$, output $EM_k(w_i^j, E_{ID}, a_i[j])$ where $a_i[j]$ is the jth bit of the string a_i.

- Very similar to the construction in Section 6.2.2 in [26], the user now queries the functionality to "unwind" the hash chain H_g to obtain $EM_k(a, b, c, E_{ID})$. For a gate represented by the tuple (a, b, c). For each query, the functionality ensures that the index number of the output wire c is strictly lower than that of the one in the previous link of the hash chain (this can be done in the natural way by including index numbers of the previous output wires in the intermediate authenticated encryptions). Also, the functionality ensures that for all $i \in [q], j \in [p(k)], c \neq w_i^j$. These checks are to avoid assigning multiple values to the same wire in the circuit C.

- User now evaluates the circuit gate by gate for all non-Oracle gates by issuing a query of the form $EM_k(a, E_{ID}, v_a)$, $EM_k(b, E_{ID}, v_b)$ and $EM_k(a, b, c, E_{ID})$. The functionality checks the consistency of the execution identity E_{ID} and outputs $EM_k(c, E_{ID}, v_c)$ where $v_c = v_a$ **NAND** v_b.

- User can finally get the secret s by querying with the "header information" and the encrypted values on the output wires. More precisely, user queries with $H_g, m, n, |C|, \{w_i^1, \ldots, w_i^{p(k)}\}_{i\in[q]}$, $a_1, \ldots, a_q$, $EM_k(m+1, E_{ID}, v_{m+1}), \ldots, EM_k(m+n, E_{ID}, v_{m+n})$. The functionality first verifies that the execution identity in all of the encrypted output wires is correct and then recovers $v = v_{m+1}||\ldots||v_{m+n}$. The functionality then checks if $v = f_1(f_2(H_g, m, n, |C|, \{w_i^1, \ldots, w_i^{p(k)}\}_{i\in[q]}))$ and outputs s if so. Output $\perp$ if any of these checks fail.

Indeed, as for the case of Turing machines, it is easy to see that an adversary A in the real world can recover s. Hence,

$$\Pr[A^{\mathcal{T}}(\mathcal{O}^{\mathcal{T}}(F_{f_1,f_2,f_3,k,0})) = 1] - \Pr[A^{\mathcal{T}}(\mathcal{O}^{\mathcal{T}}(F_{f_1,f_2,f_3,k,1})) = 1] = 1 \tag{6}$$

In the the ideal world, consider the query $H_g, m, n, |C|, \{w_i^1, \ldots, w_i^{p(k)}\}_{i\in[q]}$, $a_1, \ldots, a_q$. Given this information, it can be shown that the adversary S is essentially "committed" to the values on all the wires in the circuit C (more precisely, an analog of Lemma 23 in [26] is true in this setting). Similar to the case of Turing machines, let S_o be the set of all possible output of the Oracle circuit C (recall that the output is determined by how the Oracle queries are answered). Thus, again the probability that $f_1(f_2(H_g, m, n, |C|, \{w_i^1, \ldots, w_i^{p(k)}\}_{i\in[q]})) \in S_o$ is bounded by $\frac{1}{2^k}$. This shows that for all S,

$$\Pr[S^{F_{f_1,f_2,f_3,k,0}}(1^k) = 1] - \Pr[S^{F_{f_1,f_2,f_3,k,1}}(1^k) = 1] \leq \text{negl}(k) \tag{7}$$

Combining equations 6 and 7, we get a contradiction with the relaxed virtual black box property of the OC Oracle scheme. □

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, May 2012. 26, 27, 28

[2] A. Beimel and T. Malkin. A quantitative approach to reductions in secure computation. In *TCC*, pages 238–257, 2004. 1, 2

[3] A. Beimel, T. Malkin, and S. Micali. The all-or-nothing nature of two-party secure computation. In *CRYPTO*, pages 80–97, 1999. 1

[4] S. Brands. Untraceable off-line cash in wallet with observers. In D. R. Stinson, editor, *Advances in Cryptology CRYPTO 93*, number 773 in Lecture Notes in Computer Science, pages 302–318. Springer Berlin Heidelberg, Jan. 1994. 3

[5] G. Brassard, C. Crepeau, and M. Santha. Oblivious transfers and intersecting codes. *IEEE Transactions on Information Theory*, 42(6):1769–1780, 1996. 10

[6] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, IEEE Annual Symposium on*, volume 0, page 136, Los Alamitos, CA, USA, 2001. IEEE Computer Society. Full version available at `http://eprint.iacr.org/2000/067`. 3

[7] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. *SIAM J. Comput.*, 32(1):1–47, 2002. 1

[8] N. Chandran, V. Goyal, and A. Sahai. New constructions for UC secure computation using tamper-proof hardware. In N. Smart, editor, *Advances in Cryptology EUROCRYPT 2008*, number 4965 in Lecture Notes in Computer Science, pages 545–562. Springer Berlin Heidelberg, Jan. 2008. 3, 4

[9] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology CRYPTO 92*, number 740 in Lecture Notes in Computer Science, pages 89–105. Springer Berlin Heidelberg, Jan. 1993. 3

[10] B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991. 1

[11] T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006. 7, 8

[12] R. J. F. Cramer and T. P. Pedersen. Improved privacy in wallets with observers. In T. Helleseth, editor, *Advances in Cryptology EUROCRYPT 93*, number 765 in Lecture Notes in Computer Science, pages 329–343. Springer Berlin Heidelberg, Jan. 1994. 3

[13] I. Damgård, J. B. Nielsen, and D. Wichs. Isolated proofs of knowledge and isolated zero knowledge. In N. Smart, editor, *Advances in Cryptology EUROCRYPT 2008*, number 4965 in Lecture Notes in Computer Science, pages 509–526. Springer Berlin Heidelberg, Jan. 2008. 3, 4

[14] I. Damgård, J. B. Nielsen, and D. Wichs. Universally composable multiparty computation with partially isolated parties. In O. Reingold, editor, *Theory of Cryptography*, number 5444 in Lecture Notes in Computer Science, pages 315–331. Springer Berlin Heidelberg, Jan. 2009. 3, 4

[15] Y. Dodis and S. Micali. Lower bounds for oblivious transfer reductions. In J. Stern, editor, *Advances in Cryptology EUROCRYPT 99*, number 1592 in Lecture Notes in Computer Science, pages 42–55. Springer Berlin Heidelberg, Jan. 1999. 2

[16] N. Döttling, D. Kraschewski, and J. Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Y. Ishai, editor, *Theory of Cryptography*, number 6597 in Lecture Notes in Computer Science, pages 164–181. Springer Berlin Heidelberg, Jan. 2011. 1, 4

[17] N. Döttling, T. Mie, J. Müller-Quade, and T. Nilges. Basing obfuscation on simple tamper-proof hardware assumptions. Technical Report 675, 2011. 2

[18] N. Döttling, T. Mie, J. Müller-Quade, and T. Nilges. Implementing resettable UC-Functionalities with untrusted tamper-proof hardware-tokens. In A. Sahai, editor, *Theory of Cryptography*, number 7785 in Lecture Notes in Computer Science, pages 642–661. Springer Berlin Heidelberg, Jan. 2013. 3, 4

[19] C. Dwork, M. Naor, and A. Sahai. Concurrent zero knowledge. pages 409–418, 1998. 1

[20] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985. 12, 13

[21] U. Feige, J. Killian, and M. Naor. A minimal model for secure computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 554–563. ACM, 1994. 12, 13

[22] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. pages 218–229, 1987. 1

[23] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431473, May 1996. 3

[24] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In D. Wagner, editor, *Advances in Cryptology CRYPTO 2008*, number 5157 in Lecture Notes in Computer Science, pages 39–56. Springer Berlin Heidelberg, Jan. 2008. 1, 2, 3, 4, 12

[25] V. Goyal, Y. Ishai, M. Mahmoody, and A. Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In T. Rabin, editor, *Advances in Cryptology CRYPTO 2010*, number 6223 in Lecture Notes in Computer Science, pages 173–190. Springer Berlin Heidelberg, Jan. 2010. 1, 2, 3, 4

[26] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In D. Micciancio, editor, *Theory of Cryptography*, number 5978 in Lecture Notes in Computer Science, pages 308–326. Springer Berlin Heidelberg, Jan. 2010. Full version available at `http://eprint.iacr.org/2010/153`. 1, 2, 3, 4, 5, 12, 26, 28, 29

[27] D. Harnik, M. Naor, O. Reingold, and A. Rosen. Completeness in two-party secure computation: A computational view. *J. Cryptology*, 19(4):521–552, 2006. 1

[28] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *Theory of Computing and Systems, 1997., Proceedings of the Fifth Israeli Symposium on*, pages 174–183. IEEE, 1997. 13

[29] Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 294–304. IEEE, 2000. 12

[30] Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In P. Widmayer, S. Eidenbenz, F. Triguero, R. Morales, R. Conejo, and M. Hennessy, editors, *Automata, Languages and Programming*, number 2380 in Lecture Notes in Computer Science, pages 244–256. Springer Berlin Heidelberg, Jan. 2002. 2, 3, 13, 17, 19, 24

[31] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Extracting correlations. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 261–270. IEEE, 2009. 12, 13

[32] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, number 4515 in Lecture Notes in Computer Science, pages 115–128. Springer Berlin Heidelberg, Jan. 2007. 1, 3, 4

[33] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988. 1

[34] J. Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31. ACM, 1988. 13

[35] V. Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *Advances in Cryptology-ASIACRYPT 2005*, pages 136–155. Springer, 2005. 13

[36] Y. Lindell. General composition and universal composability in secure multi-party computation. 2003. 1

[37] Y. Lindell. Lower bounds for concurrent self composition. In *Theory of Cryptography Conference (TCC)*, volume 1, 2004. 1

[38] M. Mahmoody and D. Xiao. Languages with efficient zero-knowledge pcps are in szk. In *Theory of Cryptography*, pages 297–314. Springer, 2013. 2, 3

[39] H. K. Maji, M. Prabhakaran, and M. Rosulek. Complexity of multi-party computation problems: The case of 2-party symmetric secure function evaluation. In *TCC*, pages 256–273, 2009. 1

[40] T. Moran and G. Segev. David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In N. Smart, editor, *Advances in Cryptology EUROCRYPT 2008*, number 4965 in Lecture Notes in Computer Science, pages 527–544. Springer Berlin Heidelberg, Jan. 2008. 3

[41] V. Prabhakaran and M. Prabhakaran. Assisted common information. In *2010 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 2602–2606, 2010. 2

[42] V. Prabhakaran and M. Prabhakaran. Assisted common information: Further results. In *2011 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 2861–2865, 2011. 2

[43] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005. 12, 13

[44] S. Winkler and J. Wullschleger. On the efficiency of classical and quantum oblivious transfer reductions. In T. Rabin, editor, *Advances in Cryptology CRYPTO 2010*, number 6223 in Lecture Notes in Computer Science, pages 707–723. Springer Berlin Heidelberg, Jan. 2010. Full version available at http://arxiv.org/abs/1205.5136. 2, 6, 7, 9

[45] S. Wolf and J. Wullschleger. New monotones and lower bounds in unconditional two-party computation. *IEEE Transactions on Information Theory*, 54(6):2792–2797, 2008. 2

[46] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982. 1