On Succinct Non-Interactive Secure Computation with Malicious Security

Maya Farber Brodsky[1], Arka Rai Choudhuri[2], Abhishek Jain[3], and Omer Paneth[1]

[1]Tel Aviv University
[2]zkBricks
[3]NTT Research and JHU

Abstract

A *non-interactive secure computation (NISC)* protocol allows a client with input x and a server with input y to compute $f(x, y)$ using a single message from the client and a single response from the server. The protocol is called *succinct* if the size of the server's message depends only on the output length and is independent of the size of y and the complexity of f. In the semi-honest setting, succinct NISC is known from fully homomorphic encryption (FHE). In contrast, malicious security is currently known only from non-standard assumptions, such as SNARKs for NP.

In this work, we construct maliciously secure succinct NISC protocols for natural and widely studied functionalities from *standard assumptions*, namely, FHE and batch arguments (BARGs). Our first result is a protocol for *private set membership (PSM)*: the client holds an element x, the server holds a large set S, and the function outputs 1 if and only if $x \in S$. We then give several generalizations:

- *Dictionary lookup:* The server holds a dictionary D of key–value pairs, the client's input is a key k, and the output is $D[k]$.
- *Verifiable dictionary lookup:* The server's dictionary must additionally satisfy a predicate P, computable by a read-once machine with small state.
- *UP search:* The client input is an instance x, and the output is $D[w]$, where w is the unique witness for x under some UP relation.

Our protocols achieve *split-simulation* security against a malicious server and standard security against a malicious client. Split-simulation is a relaxation of the standard real–ideal paradigm, where correctness of the client's output and indistinguishability of the server's view are guaranteed separately.

At the heart of our results lies a *new simulation technique* in which the server's large input is extracted piece by piece and reconstructed into a coherent input. This reconstruction is enabled by a new monotone coupling argument based on Strassen's theorem.

Contents

1 Introduction **3**
- 1.1 Our Results . . . 4
- 1.2 Related Work . . . 5

2 Technical Overview **7**

3 Preliminaries **13**
- 3.1 Hash Family with Local Opening . . . 13
- 3.2 Batch Arguments . . . 14
- 3.3 Non-interactive Zero Knowledge Arguments of Knowledge . . . 16
- 3.4 Fully Homomorphic Encryption . . . 17
- 3.5 Somewhere Extractable Hash . . . 17

4 One-Round Two-Party Secure Computation **19**
- 4.1 Weak Client Security . . . 20
- 4.2 Input-Dependent Simulation-Security . . . 21

5 Technical Tool: Monotone Coupling **22**
- 5.1 Efficient simulation . . . 23

6 General Theorem: Input-Dependent Simulation-Security **25**
- 6.1 Sorted bounded-state functions . . . 27
  - 6.1.1 Protocol. . . . 27
  - 6.1.2 Client-output Indistinguishability. . . . 30
  - 6.1.3 Server-input Indistinguishability with respect to G'. . . . 38
  - 6.1.4 Server Security. . . . 41
  - 6.1.5 Client Input Privacy. . . . 43
- 6.2 Sorted lookup functions . . . 44
  - 6.2.1 Protocol. . . . 44
  - 6.2.2 Analysis. . . . 46

7 Main Result: Private Set Membership and VDL with UP Search **50**
- 7.1 Input-Dependent Simulation to Weak Simulation . . . 51

1 Introduction

Ishai et al. [IKO+11] introduced the notion of non-interactive secure computation (NISC): a secure two-party computation protocol that is carried out in one round of interaction. In NISC for a function f, a client, holding an input x sends a single message to a server. The server, holding an input y, responds with a single message from which the client can obtain the output $f(x, y)$. The protocol guarantees that the server does not learn x while the client does not learn anything about y beyond the function output.

The main appeal of NISC is that it does not require any extra rounds beyond the necessary query-response round. In addition to round complexity, previous work has sought to minimize the overhead of the protocol's communication complexity. Ideally, we would like to minimize the overhead compared to the naive insecure protocol where the server evaluates f: The client's message should grow with the size of its input x, and the server's message should grow only with the size of the function output, and not with the size of the input, or with the complexity of computing f.[1] We refer to NISC protocols with such communication as *succinct*.[2]

In the semi-honest setting, succinct NISC for all functions follows from fully homomorphic encryption (FHE) [Gen09]. In contrast, succinct NISC with malicious security in the common reference string (CRS) model[3] is only known from much stronger assumptions: Assuming succinct-non-interactive arguments of knowledge (SNARKs) for NP [Mic94, BCC+17], one can compile a semi-honest succinct NISC into a protocol with malicious security. SNARKs are also known to be *necessary* for general purpose succinct NISC with malicious security. The existence of SNARKs, however, is subject to strong barriers [BCPR14] and constructions are only known in idealized models or based on non-standard, non-falsifiable assumptions [Mic94, BCCT13].

In light of these barriers towards succinct NISC for all functions, we ask:

For which functions can we realize succinct NISC with malicious security?

Focusing on solutions based on *standard* assumptions, we do not know of positive results for any non-trivial function whose output length is shorter than the server's input.

The extraction barrier. In order to understand the challenge in achieving malicious security, we focus on the case of malicious server. Recall that simply hiding the client's input x is not sufficient. We must also guarantee that the server chooses its input *independently* of x (this is referred to as input-independence). This requirement is captured by the standard definition of simulation security: For any malicious server, there should exist an efficient simulator that can produce indistinguishable views for the client and server in the ideal execution of the protocol. In the ideal execution, both the client and the simulator submit inputs to a trusted party that provides the client with the function output. The input submitted by the simulator is typically extracted from the malicious server. However, when the server's message is shorter than its input, we have no techniques for extracting the server's input based on standard assumptions.

[1] We also allow for multiplicative factors that depend polynomially on the security parameter.

[2] Note that this is a stronger requirement than *circuit succinctness*, where the server's message may grow with its input size and the output size, but is otherwise independent of the size of the circuit. See Section 1.2 for further discussion.

[3] It is well known that (even non-succinct) NISC with malicious security is impossible in the plain model.

Split-Simulation security. Towards overcoming this barrier, we investigate a weaker notion of security that we refer to as *split-simulation*: The simulator produces views for the client and server that are *individually* indistinguishable from their views in the real execution, however, their joint distribution may not be indistinguishable as required by the standard definition of secure computation. More specifically, split-simulation provides two separate guarantees: (1) Indistinguishability of the server's view in real-world executions with different client inputs, and (2) Indistinguishability of the client's output distribution in the real and ideal world.

As discussed in [Gol04], split-simulation security is weaker than standard simulation. Nevertheless, it provides strong guarantees, including *input independence*.

For general functions, succinct NISC with split-simulation is still known to imply SNARKs. However, Ben-David et al. [BDKP22] demonstrated the feasibility of this notion for a simple function related to private information retrieval (PIR): the client holds an index i, the server holds a database D, and the function output is the i-th entry of D if D satisfies a fixed predicate P, and $\perp$ otherwise. In this work, we show that split-simulation is meaningful and achievable for a much broader class of functions.

1.1 Our Results

In this work, we demonstrate the feasibility of succinct NISC with malicious security for a non-trivial class of functions. All of our constructions satisfy split-simulation security for the clients and standard security for the servers. While our primary focus is on showing feasibility of malicious succinct NISC for a large class of functions and not on applications, our results are for natural and useful functions for which secure computation protocols have been studied and applied extensively. We rely on two cryptographic tools: fully-homomorphic encryption (FHE) and non-interactive batch arguments (BARGs) for NP [BHK17, KPY19]. Both of these tools can be instantiated from the learning with errors (LWE) assumption [BV11, CJJ22].

Private-Set Membership. Our first result is a succinct NISC for the private set-membership (PSM) function. In PSM, the client holds a value x, the server holds a set S, and the function outputs 1 iff $x \in S$. PSM has many real-world applications such as password-breach alert systems [Goo19, KLM21, App21] and private contact discovery [Mar14].

Theorem 1 (Informal)**.** *Assuming FHE and BARGs for NP, there exists a succinct NISC for PSM with split-simulation security for clients and standard security for servers.*

Split-simulation for PSM provides important security guarantees that semi-honest security does not. For example, it guarantees that a malicious server cannot make a client output 1 regardless of the client's input. More generally, split-simulation means that the server must "know" some (distribution over) polynomial size sets which is independent of the client's input, and the client's output is consistent with this set.

In the following, we discuss several important extensions of the PSM function. We state our result at the end for the most general setting.

Dictionary Lookup. The dictionary lookup function (aka, key-value search) generalizes PSM from sets to dictionaries. In dictionary lookup, the client holds a key k, and the server holds a dictionary D consisting of key-value pairs where all keys are distinct. The function outputs a value

v if D contains an entry of the form (k, v) and $\perp$ otherwise. One can think of dictionary lookup as implementing an arbitrary function whose output is not $\perp$ on at most a polynomial number of inputs.

Verifiability. Next, we extend the above function to *verifiable* dictionary lookup, where we can enforce that the server's dictionary satisfies some public predicate P. That is, the function outputs a value v if $P(D) = 1$ *and* D contains an entry of the form (k, v). Otherwise, the output is $\perp$.

In this work, we consider *bounded-state* predicates P [BDKP22]. We say that P is bounded state if it can be computed by reading the entries of D once, maintaining only a logarithmic size state between one entry to the next. For example, a bounded-state predicate can verify that the number of pairs satisfying some efficiently computable property is above/below some threshold. Additionally, we can ensure that D satisfies a bounded-state predicate P when the entries are sorted according to some order.[4]

UP search. Finally, we further extend the above function to support more general key matching procedures. Instead of directly searching for the key k, the client holds an instance x and we search for a witness for x with respect to some UP relation[5] R_{UP}. That is, the function outputs a value v if $P(D) = 1$ *and* D contains an entry of the form (w, v) such that $R_{\mathsf{UP}}(x, w) = 1$. Otherwise, the output is $\perp$.

We now state our final result:

Theorem 2 (Informal)**.** *Assuming FHE and BARGs for NP, there exists a succinct NISC for verifiable dictionary lookup with UP search for the class of (sorted) bounded-state predicates with split-simulation security for clients and standard security for servers.*

We can extend the theorem to predicates with state of super-logarithmic size s at the cost of allowing the server's message to grow with s, assuming FHE and BARGs that are 2^s-secure, and letting the simulator run in time 2^s.

Private set intersection. In Theorem 1 and Theorem 2 we consider functions where the client's input matches at most a single element in the server's set/dictionary. Therefore, a natural goal is to extend these theorems to the setting of private set intersection (PSI), where the client's input is also a set. However, we observe that succinct NISC with split-simulation for PSI implies (designated-verifier) SNARKs. In fact, this limitation holds even in the case where the client's input contains only 3 elements. See Section 2 for more details.

1.2 Related Work

Laconic Cryptography. The notion of laconic function evaluation (LFE) [QWW18, CDG+17] can be viewed as the dual of semi-honest succinct NISC. In LFE, a server (holding a *large* input) sends the first message to a client (holding a *small* input). The client responds to the server who then computes the function output. Similar to succinct NISC, LFE requires that the size of the server's message is independent of its input size. For this reason, achieving malicious-secure LFE involves

[4] If the size of the entries is super-logarithmic, a bounded-state predicate cannot check if the entries are ordered.

[5] A UP relation is an NP relation with the restriction that each instance x has at most one witness.

similar barriers as malicious-secure succinct NISC. However, it is possible to achieve distinguisher-dependent simulation security [JKKR17] based on standard cryptographic assumptions [DGGM19]. In distinguisher-dependent simulation, instead of requiring a universal simulator that works for every distinguisher, we allow the simulator to depend on the distinguisher. Due to the reverse communication pattern in LFE (i.e., server speaks before the client), distinguisher-dependent simulation is enabled by rewinding techniques. However, this is not known to be possible in the setting of succinct NISC.

Recently, Bartusek et al. [BGJP25] gave a construction of laconic private-set membership in the generic group model. They consider a notion of authenticated-set security which requires the server's database to contain entries signed by a trusted-third party. It is easy to see that our result in Theorem 2 can also enforce such predicates on the server's database. Unlike their work, we support a broader class of predicates and achieve security from *standard* assumptions. Most significantly, while their protocol requires the server to speak first (and compute the function output), our protocol uses the reverse communication pattern where the client speaks first and learns the output.

Credibility in PSM. In a recent work, Garg et al. [GHJ+23] investigated malicious-secure succinct NISC for private set membership and present constructions from standard assumptions. Their security definition, however, is significantly weaker than the notion of split-simulation. In particular, their definition only guarantees that for high-entropy client inputs, a malicious server cannot force an accepting output. Split-simulation, in contrast, does not place such restrictions, and is therefore closer to standard simulation security. For this reason, the techniques used in our work are very different from theirs.

Verifiable PIR. Ben-David, Kalai and Paneth [BDKP22] consider succinct NISC for the private information retrieval (PIR) functionality with the added guarantee that the database D used by the server must satisfy some bounded-state predicate P. They refer to this notion as verifiable PIR (vPIR) and achieve it with split-simulation security for the client based on BARGs. vPIR can be seen as a special case of verifiable dictionary lookup where the key space is of polynomial size.

Circuit-Succinct Secure Computation. In this work, we consider a strong notion of succinctness where the size of the server's message is independent of both the size of the server's input and the complexity of computing f. Prior works have considered a relaxed notion of *circuit-succinct* secure computation, where the size of the server message may grow with the input length and output length, but is otherwise independent of the circuit complexity of f. Morgan, Pass and Polychroniadou [MPP20] construct circuit-succinct NISC achieving simulation-security, albeit with a simulator whose running time is super-polynomial in the security parameter. Further, they require the underlying primitives used in their construction to also satisfy super-polynomial security.

We also mention recent works [ABJ+19, CMOS25] that study circuit-succinct secure computation in the *interactive* setting. These works construct protocols that achieve standard simulation security (with a polynomial-time simulator) based on standard assumptions.

2 Technical Overview

We first provide an overview of our PSM protocol and then discuss its extensions. Before describing our final protocol, we will present a protocol for general bounded-state functions. While we will not be able to prove that this protocol satisfies split-simulation security, we will show that it satisfies a weaker notion of security that will be useful in our final construction.

Bounded-state functions. A bounded-state function $f(x, y)$ reads the input $y = (y_1, \ldots, y_k)$, one element at a time, maintaining a logarithmic-size state. More formally, f is bounded state if there exist $s = O(\log \lambda)$, an efficiently computable state update function $U : \{0,1\}^* \times \{0,1\}^s \to \{0,1\}^s$ and an initial state $z_0 \in \{0,1\}^s$ such that:

$$f(x, \epsilon) = z_0 \ ,$$
$$f(x, (y_1, \ldots, y_k)) = U((x, y_k), f(x, (y_1, \ldots, y_{k-1}))) \ .$$

BARGs. The succinct NISC protocol for bounded state f is based on FHE and BARGs, as well as a hash tree. Recall that a BARG is a non-interactive argument system in the CRS model for the conjunction of NP statements where the proof size is polynomial in the size of a single witness and independent of the number of statements. More specifically, we will use BARGs for the index relation [CJJ22]: the BARG verifier is given an verification machine M and the number of statements k. The BARG proof asserts that for every $i \in [k]$ there exists a witness w_i such that $M(i, w_i) = 1$.

The security property satisfied by the BARG is called somewhere extraction [CJJ22]: For every $i \in [k]$, it is possible to sample a CRS that is "programmed" on i together with a trapdoor τ such that (i) the programmed CRS is indistinguishable from an honestly generated CRS; and (ii) for every poly-size adversary that is given a CRS programmed on i and outputs a verification machine M and a proof π, if π is an accepting proof for (M, k) then using τ we can extract from π a witness w such that $M(i, w) = 1$ with all but negligible probability.

A protocol for bounded-state functions. Next we describe a succinct NISC protocol for any bounded-state function f. In what follows, we focus on split simulation security for the client and ignore server security. To achieve server security, one can simply compose our protocol with a non-succinct NISC protocol with full simulation security for both parties. Such a protocol can be obtained from oblivious transfer and non-interactive zero-knowledge proofs [IKO+11]. In this work, we present a simpler transformation tailored to our protocol, using non-interactive zero-knowledge proofs.

The CRS contains two independent BARG CRSs. Given input x, the client sends an FHE encryption $\hat{x}$ of x. Given input $y = (y_1, \ldots, y_k)$ and message $\hat{x}$, the server:

- Homomorphically evaluates $f(\cdot, (y_1, \ldots, y_i))$ on $\hat{x}$ for every $i \in [0, k]$. Let $\hat{z}_0, \ldots, \hat{z}_k$ be the resulting ciphertexts for all intermediate states.
- Computes a hash tree over the strings $y_1, \ldots, y_k$ and $\hat{z}_0, \ldots, \hat{z}_k$. Let h be the root of the hash tree.
- Computes two BARG proofs (one under every CRS) that for every $i \in [k]$ there exist local openings of h to $y_i, \hat{z}_{i-1}$ and $\hat{z}_i$ such that $\hat{z}_i$ is obtained by homomorphically evaluating $U((\cdot, y_i), \cdot)$ on $(\hat{x}, \hat{z}_{i-1})$. Let π_1, π_2 be the BARG proofs.

Finally, the server sends h, π_1, π_2 and local openings of h to $\hat{z}_0$ and $\hat{z}_k$. The verifier checks π_1, π_2, the local openings and that $\hat{z}_0$ is indeed the homomorphic evaluation of $f(\cdot, \epsilon)$ on $\hat{x}$. If all checks pass, it decrypts and outputs z_k.

A flawed simulation. Next we describe a flawed split-simulation strategy for the above protocol. The server's view can be simulated by encrypting some dummy input $\perp$ instead of x. This is indistinguishable from the client's message by the security of the FHE. We proceed to describe how to simulate the client's output.[6] For every x and $i \in [k]$, let $(y_i^x, \tilde{z}_{i-1}^x, z_i^x)$ be as follows:

- Sample two CRSs programmed on i and a client message $\hat{x}$ and give these to the adversary.
- Obtain the adversary's answer that contains h, π_1, π_2 and a local openings of h to $\hat{z}_0$ and $\hat{z}_k$. If the answer does not verify, go back to the first step.
- Using the trapdoor for the first CRS, extract from π_1 a witness that contains local openings of h to $y_i, \hat{z}_{i-1}$ and $\hat{z}_i$.
- Decrypt z_{i-1} and z_i and set $(y_i^x, \tilde{z}_{i-1}^x, z_i^x) = (y_i, z_{i-1}, z_i)$.

Set $\tilde{z}_k^x = z_k^x$ .Using the CRS indistinguishability and extraction properties of the BARG, we can show by a standard argument that for every x and $i \in [k]$:

- $z_i^x = U((x, y_i^x), \tilde{z}_{i-1}^x)$ except with negligible probability.
- z_i^x and $\tilde{z}_i^x$ are statistically close.

Moreover, $\tilde{z}_0^x = z_0$ except with negligible probability and z_k^x is statistically close to the output of the client in the real experiment on input x.

Since z_i^x and $\tilde{z}_i^x$ are statistically close, we can couple[7] together the k distributions above into a single correlated distribution $(y_i^x, z_i^x)_{i \in [k]}$ such that for every $i \in [k]$, $z_i^x = U((x, y_i^x), z_{i-1}^x)$ except with negligible probability. Denoting $y^x = (y_1^x, \ldots, y_k^x)$ this condition is equivalent to $z_k^x = f(x, y^x)$.

Consider the (flawed) simulator that samples y^x and submits it to the trusted party in the ideal experiment. Indeed, the output of the ideal experiment will be $z_k^x = f(x, y^x)$ which is statistically close to the output of the client in the real experiment on input x. The problem with this simulation strategy is, of course, that it depends on the client's input x. Observe that to sample the variable y_i^x, we only need $\hat{x}$ and not the decryption key or x in the clear. Therefore, for every x, x', we have that y_i^x and $y_i^{x'}$ are computationally indistinguishable. However, this is not the case for the joint distributions y^x and $y^{x'}$ since the coupling depends on the z_i^x variables which we cannot sample without the decryption key. Nonetheless, we observe that the protocol above does satisfy a useful notion of security that we will refer to as *input-dependent simulation*.

Before describing the notion of input-dependent simulation and how to use it, we discuss a couple of subtle technical issues with the proposed simulation strategy (this may be skipped on a first reading). First, the argument above implicitly assumes that the client never outputs $\perp$ in the real experiment. Moreover, if the client outputs $\perp$ with probability $1 - \epsilon$, the expected time

[6] For readers that are familiar with the simulator of [BDKP22], here we use the same strategy, except that it is under homomorphic encryption.

[7] A coupling of random variables A, B is a joint distribution (A, B) whose marginals match A and B. If A and B are statistically close, there exists a coupling (A, B) such that $A = B$ except with negligible probability.

to sample from y^x becomes $\mathsf{poly}(1/\epsilon)$ which may not be polynomially bounded. To resolve both issues, we first estimate ϵ by executing the adversary with a random CRS and client message and checking its answer multiple times. Then, with probability $1-\epsilon$, we simulate the output $\perp$ and with probability ϵ, we sample y^x and proceed as before. In the remainder of this overview, we continue to assume, for simplicity, that the client never outputs $\perp$ (i.e. that $\epsilon = 1$).

The second issue is that, while we can efficiently sample from $(y_i^x, \tilde{z}_{i-1}^x, z_i^x)$ for each i, we may not be able to couple these distributions efficiently. To resolve this, we take $\ell = 2^{O(s)} = \mathsf{poly}(\lambda)$ samples from $(y_i^x, \tilde{z}_{i-1}^x, z_i^x)$ and consider the empirical distribution given by taking one of these ℓ samples at random. Since $|z_i^x| = s$ we get that with high probability over the samples, the value of z_i^x in the empirical distribution and the actual distribution are $1/\mathsf{poly}(\lambda)$-close in statistical distance. Therefore, we can still couple the empirical distributions such that the simulation error is $1/\mathsf{poly}(\lambda)$. Since the empirical distributions have support size $\mathsf{poly}(\lambda)$, we can couple them efficiently by computing a matching on the support of each pair of distributions.

Finally, we note that we can extend the protocol and its simulation to support functions with states of super-logarithmic size s; however, the simulation time will grow with $2^{O(s)}$ since we take $\ell = 2^{O(s)}$ samples, and we need to rely on FHE and BARGs that are 2^s-secure. While super-polynomial time simulation is still meaningful in many settings, it may be problematic when using the NISC protocol for cryptographic functions or when using the protocol in a larger system with other cryptographic primitives.

Input-dependent simulation. The strategy above is able to simulate the client's output in the real experiment on input x by using the input x. This task on its own is clearly trivial: we can simply emulate the real experiment on input x and obtain the client's output. What makes the strategy above interesting is that it efficiently extracts a distribution y_x that satisfies the following properties:

***Client-output indistinguishability*:** For every x, the client's output in the real experiment on input x is computationally indistinguishable from $f(x, y^x)$. That is, the server's input distribution y^x "explains" the client's output in the real experiment on input x.

***Server-input indistinguishability*:** For every x, x' and $i \in [k]$, y_i^x and $y_i^{x'}$ are computationally indistinguishable. That is, the distribution of each individual entry y_i^x looks independent of x.

Our idea is to use the above protocol that provides input-dependent simulation security for general bounded-state functions, to achieve split-simulation security for specific bounded-state functions such as PSM.

A flawed PSM protocol. To implement a PSM protocol we consider the bounded-state function $f^\#(x, y)$ that outputs the number of times x appears in the input $y = (y_1, \ldots, y_k)$. Indeed, $f^\#$ can be implemented with a state of size $\log(k)$. Given a protocol for $f^\#$, we construct a PSM protocol as follows: the client and server execute the protocol for $f^\#(x, y)$ where x is the client's element and y contains the elements of the server's set S in some arbitrary order. The client obtains the outputs $f^\#(x, y)$ and outputs 1 iff $f^\#(x, y) > 0$.

We propose a (flawed) split-simulator for $f^\#$: Simply follow the input-dependent simulation strategy using a dummy input $\perp$. That is, the simulator samples $y^\perp$ and submits it to the trusted

party in the ideal experiment. We argue that the output of the simulator is "similar" to the output of the client in the real experiment on input x, in the sense that both variables have almost the same expectation. Indeed, by the server-input indistinguishability property of the input-dependent simulation we have that:

$$\mathbb{E}\left[f^{\#}(x, y^{\perp})\right] = \mathbb{E}\left[\sum_{i\in[k]} \mathbb{1}_{y_i^{\perp}=x}\right] = \sum_{i\in[k]} \mathbb{E}\left[\mathbb{1}_{y_i^{\perp}=x}\right]$$

$$\approx \sum_{i\in[k]} \mathbb{E}\left[\mathbb{1}_{y_i^{x}=x}\right] = \mathbb{E}\left[\sum_{i\in[k]} \mathbb{1}_{y_i^{x}=x}\right] = \mathbb{E}\left[f^{\#}(x, y^{x})\right] \quad .$$

By the client-output indistinguishability property, we know that $f^{\#}(x, y^x)$ is indistinguishable from the client's output in the real experiment on input x. Since the output of f is in a polynomial size set, this means that $\mathbb{E}\left[f^{\#}(x, y^x)\right]$ is negligibly close to the expected client's output in the real experiment. Therefore, the same holds also for $\mathbb{E}\left[f^{\#}(x, y^{\perp})\right]$.

The problem, of course, is that while the client's output distribution has almost the same expectation in the real and in the ideal experiment, the two distributions may be very different. For example, consider an adversary that follows the protocol, but sets its input y as follows: with probability $1/k$ it sets each y_i to x and with probability $1-1/k$ its sets each y_i to some $x' \notin \{x, \perp\}$. In both the real and in the ideal experiments, the client output 1 in expectation. However, in the real world on input x the server's input entries are either all x or all x' while in the simulated server input $y^{\perp}$ the distributions of entries are independent. Therefore, in the real experiment the client's output will be > 0 with probability $1/k$ while in the ideal world the client's output is 1 with probability $> 1/2$.

'To fix our PSM protocol, we go back and design a protocol with input-dependent simulation for a richer class of functions.

Sorted bounded-state functions. Going beyond bounded-state functions, we give an succinct NISC protocol with input-dependent simulation for the class of *sorted* bounded-state functions. We say that a function f is sorted bounded-state if there exists a bounded-state function g and an efficiently computable total order R (either strict or non-strict) such that:

$$f(x, y = (y_1, \ldots, y_k)) = \begin{cases} g(x, y), & \text{if } \forall i \in [k-1] : \ (y_i, y_{i+1}) \in R \\ \perp, & \text{otherwise} \end{cases} \quad .$$

We note that if each entry y_i is of size at most s, then any sorted bounded-state function can be implemented by a function with state of size $O(s)$ which also holds in its state the entry y_i previously read, and compares it to the current entry y_{i+1}. Therefore, if the server's input has bounded-size entries, we can use the protocol for bounded-state functions, however, if the entries are of super-logarithmic size, this will require super-polynomial time simulation. In contrast, our protocol for sorted bounded-state functions has an efficient input-dependent simulator, and it does not require the entries to be bounded in size.

Before describing our protocol with input-dependent simulation for sorted bounded-state functions, we explain how to use it to get a protocol for PSM with split-simulation. The idea is that by forcing the entries $y_1, \ldots, y_k$ to be strictly increasing (and, in particular, distinct), we restrict the

output of $f^{\#}(x, y)$ (which counts the number of times x appears in y) to be in $\{0, 1\}$ instead of $[k]$. Since our simulator guarantees that client's output distribution has almost the same expectation in the real and in the ideal experiment, and since these distributions are now supported on $\{0, 1\}$, it follows that the two distributions must also be close in statistical distance.

A protocol for sorted bounded-state functions. Next we describe our succinct NISC protocol for a sorted bounded-state function f given by a bounded-state function g and a total order R. The protocol is identical to the protocol for bounded-state functions above except that we add another BARG to the server message to prove that y is sorted. That is, the new BARG proves that for every $i \in [k-1]$ there exist local openings of h to y_i and y_{i+1} such that $R(y_i, y_{i+1}) = 1$.

We now describe an input-dependent simulation for this protocol. For simplicity, in this overview we ignore the function g and focus on simulating a sorted server input. Extending this to general sorted bounded-state functions is done by a straightforward combination of the simulation below with the simulation for bounded-state functions above. Similarly to the input-dependent simulator above, for every x and $i \in [k-1]$, we sample $(y_i^x, \tilde{y}_{i+1}^x)$ and set $y_k^x = \tilde{y}_k^x$. From the CRS indistinguishability and extraction properties of the BARG, we can show that for every $i \in [k-1]$:

- $R(y_i^x, \tilde{y}_{i+1}^x)$ except with negligible probability.
- y_{i+1}^x and $\tilde{y}_{i+1}^x$ are computationally indistinguishable.

If y_{i+1}^x and $\tilde{y}_{i-1}^x$ were statistically close, then we could couple together these $k-1$ distributions into a single correlated distribution $y = (y_1^x, \ldots, y_k^x)$ such that for every $i \in [k-1]$, $R(y_i^x, y_{i+1}^x)$ (i.e. y is sorted) except with negligible probability. However, in general y_{i-1}^x and $\tilde{y}_{i-1}^x$ may not be statistically close.

Monotone coupling. To handle the computational case, we prove the following monotone coupling lemma: If a pair of random variables A and B are computationally indistinguishable, there exists a coupling (A, B) such that $R^*(A, B) = 1$ except with negligible probability, where R^* is the non-strict version of R. This lemma follows from a variant of Strassen's theorem that gives necessary and sufficient conditions for the existence of coupling satisfying a relation R. For intuition, we also sketch a direct proof of the lemma: for $r \in [0, 1]$ we denote by $a_r \in A$ and $b_r \in B$ the elements such that:

$$\Pr_{a \leftarrow A}[R^*(a, a_r)] = \Pr_{b \leftarrow B}[R^*(b, b_r)] = r \quad .$$

This defines a natural coupling: (a_r, b_r) for a uniform $r \leftarrow [0, 1]$. Note that even if A and B are computationally indistinguishable, it is possible that $R^*(a_r, b_r) = 0$ for most r's. Intuitively, however, a_r and b_r must be "close" to each other with respect to R^*. Therefore, we define a new coupling that is "shifted" by some ϵ: $(a_r, b_{r+\epsilon \bmod 1})$ for a uniform $r \leftarrow [0, 1]$. Since A and B are computationally indistinguishable, there exists a negligible ϵ such that $R^*(a_r, b_{r+\epsilon}) = 1$ for every $r \in [0, 1-\epsilon]$. Otherwise, we can distinguish between A and B since:

$$\Pr_{x \leftarrow B}[R^*(x, a_r) = 1] - \Pr_{x \leftarrow A}[R^*(x, a_r) = 1] \geq$$
$$\Pr_{x \leftarrow B}[R^*(x, b_{r+\epsilon}) = 1] - \Pr_{x \leftarrow A}[R^*(x, a_r) = 1] = \epsilon \quad .$$

Since r falls outside $[0, 1-\epsilon]$ with probability ϵ the lemma follows.

Using this lemma we can couple the distributions $(y^x_{i-1}, \tilde{y}^x_i)$ and $(y^x_i, \tilde{y}^x_{i+1})$ into a joint distribution $(y^x_{i-1}, \tilde{y}^x_i, y^x_i, \tilde{y}^x_{i+1})$ such that except with negligible probability:

$$R(y^x_{i-1}, \tilde{y}^x_i) \quad \wedge \quad R^*(\tilde{y}^x_i, y^x_i) \quad \wedge \quad R(y^x_i, \tilde{y}^x_{i+1}) \quad \Rightarrow \quad R(y^x_{i-1}, \tilde{y}^x_{i+1}) \ .$$

By repeating this process, we get a single joint distribution $y = (y^x_1, \ldots, y^x_k)$ such that y is sorted except with negligible probability. However, we may not be able to sample y efficiently.

To get efficient sampling, we prove a version of the monotone coupling lemma for empirical distributions: Let A and B be computationally indistinguishable random variables, and let L_A and L_B be the empirical distributions obtained by taking ℓ independent samples from A and from B respectively and considering the uniform distribution over these ℓ samples. Then there exists a coupling (L_A, L_B) such that $R^*(L_A, L_B) = 1$ except with probability $1/\mathsf{poly}(\ell)$. Since the empirical distributions have support size ℓ, we can find this coupling in time $\mathsf{poly}(\ell)$ by computing a matching on the support of the distributions. By taking $\ell = \mathsf{poly}(\lambda)$ to be sufficiently large, we can efficiently sample $y = (y^x_1, \ldots, y^x_k)$ such that y is sorted except with probability $1/\mathsf{poly}(\lambda)$.

Extending to dictionary lookup. Next we describe how to extend our PSM protocol to dictionary lookup. Recall that here, a client holds a key k, the server holds a dictionary $D = \{(k_i, v_i)\}_{i\in[m]}$, and the client obtains the value v_i for $i \in [m]$ such that $k_i = k$. If the size of each v_i is logarithmic, then we can use the same protocol for sorted bounded-state functions, replacing $f^\#(k, D)$ with a function that outputs the sum of v_i for each i such that $k_i = k$. We can then use the same argument using expectation to argue that input-dependent simulation of this protocol suffices to achieve split-simulation.

To support values v_i that cannot be stored in bounded state, we extend our protocol from sorted bounded-state functions to sorted *lookup* functions. Given a bounded-state function Search and a total order R, we define the lookup function

$$f(x, y = (y_1, \ldots, y_m)) = \begin{cases} \mathsf{Index}(\mathsf{Search}(x, y), y), & \text{if } \forall i \in [m-1] : \ (y_i, y_{i+1}) \in R \\ \perp, & \text{otherwise} \end{cases} ,$$

where the indexing function $\mathsf{Index}(i, y)$ returns y_i if $i \in [m]$, and otherwise $\perp$.

This class of functions captures dictionary lookup: the Search function will return the index of the matching key (which unlike the value, can be stored in bounded state), and the total order is on the keys.

To support this class, we need to modify our protocol. Intuitively, we compose our protocol with a (specific) protocol for private information retrieval. The server, after computing the encrypted output $\hat{z}$ of Search, uses this ciphertext as a PIR query to its input, obtains a PIR answer, and sends it to the client (along with the previous hash, ciphertexts and proofs).

The only remaining issue is to ensure consistency: the server should use the same input for the PIR and for the protocol computing Search. To achieve this, we use a specific PIR scheme, itself given by a BARG. Here, the PIR query is a BARG CRS (which we set to be programmed on $\hat{z}$), and the PIR answer is a BARG proof. The BARG proves that for each i, there exists a local opening of h (the hash value in our original protocol) to y_i. The client then obtains its final output by extracting from this proof, a witness which contains y_z.

The input-dependent simulator for the resulting protocol is almost identical, except that we additionally couple based on the output $\hat{z}$.

Extending to verifiability and UP search. Using sorted lookup functions, we can support checking bounded-state predicates P on the server's input, and more general key matching procedures given by a UP relation R_{UP}, by defining appropriate search functions. Namely, we can define $\mathsf{Search}(x, y)$ that returns the index i of the first entry $y_i = (w, v)$ such that $R_{\mathsf{UP}}(x, w) = 1$, or $\perp$ if there is no match or if the predicate $P(y)$ is not satisfied.

The unique witness property is needed to ensure that the sum of matching values (which is what we can argue about in expectation) is a single value. To hide the matching witness w (so that the client only receives the value v), we define a more general Index that additionally applies a projection function $\mathsf{Project}(y_i)$, and the witnesses for the additional BARG only contain the projections.

On succinct NISC for private set intersection. Finally, we argue that succinct NISC for PSI with split-simulation implies (designated-verifier) SNARKs. We can use succinct NISC protocol for PSI to construct a designated verifier SNARK with constant soundness, as follows. The prover, given a witness w, first computes a PCP proof π over the binary alphabet with query complexity 3 and knowledge soundness. The prover then constructs the set $\{i : \pi_i = 1\}$ as its PSI server input. The SNARK CRS contains the CRS for the PSI together with PSI client's first message, where the client's input is 3 indices corresponding to a random PCP query. To compute the proof, the SNARK prover (acting as a server in the PSI protocol) outputs the server's message. To verify the proof, the SNARK verifier (acting as a client in the PSI protocol) obtains the PCP answers via the PSI protocol, verifies the PCP, and accepts/rejects.

The SNARK's knowledge extractor uses the simulator of the PSI protocol: The simulator outputs a simulated server's input, which in turn defines a PCP string. Split simulation guarantees that for every client input (i.e. the PCP queries), the client output (i.e. the PCP answers) is indistinguishable in the real and ideal world. In particular, if the real-world client accepts with some probability over a random query, the same must hold in the ideal world, for the extracted PCP proof. Knowledge soundness of the PCP guarantees that we can extract an accepting witness from the PCP proof.

3 Preliminaries

Notations. We use PPT to denote probabilistic polynomial-time, and denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from the distribution $\mathcal{D}$. By $\{X_\lambda\}_\lambda \approx_c \{Y_\lambda\}_\lambda$ we denote that the two distributions $\{X_\lambda\}_\lambda$ and $\{Y_\lambda\}_\lambda$ are computationally indistinguishable.

We define the primitives used in the work below.

3.1 Hash Family with Local Opening

In this section we recall the definition of a hash family with local opening [Mer88].[8]

[8] In what follows we use the notation HT to denote a hash family with local opening, where HT symbolizes a Hash Tree construction. We emphasize that we are not restricted to such a construction, and use this notation only to give the reader an example to have in mind.

Syntax. A hash family (HT) with succinct local opening consists of the following algorithms:

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$**.** This is a PPT algorithm that takes as input the security parameter 1^λ in unary and outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{rt}$**.** This is a deterministic poly-time algorithm that takes as input a hash key hk and an input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, and outputs a hash value rt.

$\mathsf{Open}(\mathsf{hk}, x, j) \to \rho$**.** This is a deterministic poly-time algorithm that takes as input a hash key hk, an input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, and an index $j \in [N]$, and outputs an opening ρ.

$\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, b, \rho) \to 0/1$**.** This is a deterministic poly-time algorithm that takes as input a hash key hk, a hash value rt, an index $j \in [N]$, a bit $b \in \{0,1\}$ and an opening ρ. It outputs a bit (1 to accept, 0 to reject).

Definition 1. *(Properties of* HT*) A* HT *family* $(\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ *is required to satisfy the following properties.*

Opening completeness. *For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any $x \in \{0,1\}^N$, and any index $j \in [N]$,*

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, x_j, \rho) = 1 \;:\; \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathsf{rt} = \mathsf{Hash}(\mathsf{hk}, x), \\ \rho = \mathsf{Open}(\mathsf{hk}, x, j) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Succinctness. *In the completeness experiment above, we have that* $|\mathsf{hk}| + |\mathsf{rt}| + |\rho| = \mathsf{poly}(\lambda)$.

Collision resistance w.r.t. opening. *For any poly-size adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, 0, \rho_0) = 1 \\ \wedge\, \mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, 1, \rho_1) = 1 \end{array} \;:\; \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ (\mathsf{rt}, j, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Remark 1. *We say that a hash family with local opening is Λ-secure, for $\Lambda = \Lambda(\lambda)$, if the collision resistance w.r.t. opening property holds against any* $\mathsf{poly}(\Lambda)$*-size adversary (as opposed to* $\mathsf{poly}(\lambda)$*-size) and the probability that the adversary finds a collision is* $\mathsf{negl}(\Lambda)$ *(as opposed to* $\mathsf{negl}(\lambda)$*). We refer to this property as Λ-collision-resistance w.r.t. opening.*

Remark 2. *We note that while the above definition specifies opening to a single bit, we shall use hash trees to open to a fixed number of bits, say ℓ. The above definition and construction easily extends to this setting as well.*

Theorem 3 ([Mer88])**.** *Assuming the existence of a collision resistant hash family there exists a hash family with local opening (according to 1).*

It was recently shown [BKP+24], that collision resistant hash functions are implied by batch arguments (BARGs Definition 2), and thus we do not explicitly assume the existence of collision resistant hash functions in our theorems if we additionally assume the existence of BARGs.

3.2 Batch Arguments

We recall the definition of batch arguments [CJJ22].

Syntax. A (publicly verifiable and non-interactive) somewhere extractable batch argument for NP (BARG) is given by the following polynomial-time algorithms:

- $\mathsf{Gen}(1^\lambda, i)$: On input the security parameter and index i the generation algorithm returns a CRS crs and a trapdoor τ.
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{M}, 1^t, (w_1, \ldots, w_m))$: On input the CRS crs, a Turing machine M, time bound 1^t, and a sequence of m witnesses $w_1, \ldots, w_m$ the prover algorithm return a proof π.
- $\mathsf{Verify}(\mathsf{crs}, \mathsf{M}, t, m, \pi)$: On input the CRS crs, a Turing machine M, time bound t, a number m and a proof π the verification algorithm returns a bit.
- $\mathsf{Extract}(\tau, \pi)$: On input the trapdoor τ and proof π the extraction algorithm outputs a witness w.

Definition 2 (somewhere extractable BARG)**.** *A somewhere extractable BARG for NP satisfies the following requirements.*

- *(Completeness) For every $\lambda \in \mathbb{N}$, $m, t \leq 2^\lambda$, $i \in [m]$, Turing machine M and witnesses $w_1, \ldots, w_m \in \{0,1\}^n$ such that for every $j \in [m]$, $\mathsf{M}(j, w_j)$ accepts in t steps:*

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{crs}, \mathsf{M}, t, m, \pi) = 1 & : \quad (\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda, i) \\ \mathsf{Extract}(\tau, \pi) = w_i & \quad\;\; \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{M}, 1^t, (w_1, \ldots, w_m)) \end{array}\right] = 1.$$

- *(Efficiency) In the completeness experiment above $|\pi| = \mathsf{poly}(\lambda, n)$. The argument is rate-1 if $|\pi| = n + \mathsf{poly}(\lambda)$.*
- *(Index Hiding) For every polynomial-size adversary $\mathcal{A}$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and indexes $i_0, i_1 \leq 2^\lambda$:*

$$\Pr\left[\mathcal{A}(\mathsf{crs}) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\} \\ (\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda, i_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

- *(Somewhere Extraction) For every polynomial-size adversary $\mathcal{A}$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $i \leq 2^\lambda$:*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \mathsf{M}, t, m, \pi) = 1 \\ \mathsf{M}(i, w) \text{ does not accept within } t \text{ steps} \end{array} \quad : \quad \begin{array}{l} (\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda, i) \\ (\mathsf{M}, 1^t, m, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w \leftarrow \mathsf{Extract}(\tau, \pi) \end{array}\right] \leq \mu(\lambda).$$

BARGs satisfying the above definition can be constructed based on the hardness of LWE, the k-Lin assumption for $k > 1$ or sub-exponential DDH [CJJ22, HJKS22, WW22, CGJ+23, KLVW23].

Remark 3. *Given a somewhere extractable BARG, one can naturally extend the definition of the key generation algorithm* Gen *to take as input an index set $I \subset [k]$, as opposed to a single index.* $\mathsf{Gen}(1^\lambda, I)$ *will simply run* $\mathsf{Gen}(1^\lambda, i)$ *for every $i \in I$. The prover algorithm* Prove*, given a* crs *that encodes the $|I|$ indices, will simply generate $|I|$ proofs (one for each* crs*), and the verifier will check these $|I|$ proofs independently. The size of the proof π now grows linearly in $|I|$. The index hiding property holds for any I and I' as long as $|I| = |I'|$. Lastly, extraction will output the witnesses at all positions indexed by I, i.e.* $\mathsf{Extract}$ *outputs $\{w_i\}_{i \in I}$.*

3.3 Non-interactive Zero Knowledge Arguments of Knowledge

Syntax. Let $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of NP languages, and $\mathsf{Rel} = \{\mathsf{Rel}_\lambda\}_{\lambda\in\mathbb{N}}$ the corresponding family of NP relations. A non-interactive zero-knowledge (NIZK) argument of knowledge given by the following polynomial-time algorithms:

- $\mathsf{Gen}(1^\lambda)$: On input the security parameter, the generation algorithm returns a CRS crs and a trapdoor τ.
- $\mathsf{Prove}(\mathsf{crs}, x, w)$: On input the CRS crs, a statement x, and a witness w, the prover algorithm returns a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi)$: On input the CRS crs, a statement x and a proof π the verification algorithm returns a bit.
- $\mathsf{Extract}(\tau, \pi)$: On input the trapdoor τ and proof π the extraction algorithm returns a witness w.

Definition 3. *A non-interactive zero-knowledge (NIZK) argument* $\Pi = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract})$ *for an NP language* $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ *relation* $\mathsf{Rel} = \{\mathsf{Rel}_\lambda\}_{\lambda\in\mathbb{N}}$ *is required to satisfy the following properties:*

Completeness. *For every* $\lambda \in \mathbb{N}$, $x \in \mathcal{L}_\lambda$ *and witness* w *such that* $(x, w) \in \mathsf{Rel}_\lambda$*:*

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l}(\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda)\\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array}\ \right] = 1.$$

Zero-Knowledge. *For any polynomial-size verifier* $\mathcal{V}^*$*, there exists a polynomial-size simulator* Sim *and a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $x \in \mathcal{L}_\lambda$ *and witnesses* $w \in \{0,1\}^n$ *such that* $(x, w) \in \mathsf{Rel}_\lambda$*:*

$$\left|\Pr\left[\ \mathcal{V}^*(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l}(\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda)\\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array}\ \right]\right.$$
$$\left. - \Pr\left[\ \mathcal{V}^*(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l}(\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda)\\ \pi \leftarrow \mathsf{Sim}(x, \tau)\end{array}\ \right]\right| \leq \mathsf{negl}(\lambda)$$

Knowledge-extractability. *For every polynomial-size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\ \begin{array}{l}\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1\\ (x, w) \notin \mathsf{Rel}_\lambda\end{array} \quad : \quad \begin{array}{l}(\mathsf{crs}, \tau) \leftarrow \mathsf{Gen}(1^\lambda, i)\\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})\\ w \leftarrow \mathsf{Extract}(\tau, \pi)\end{array}\ \right] \leq \mathsf{negl}(\lambda).$$

Efficiency. *The size of the proof is* $|w| \cdot \mathsf{poly}(\lambda)$.

Remark 4. *Given any NIZK for all of NP, one can add a knowledge extractor with the use of any public-key encryption scheme in a standard manner. The* crs *now also includes a randomly sampled public key, and the trapdoor* τ *is the corresponding secret key. A prover simply encrypts their witness under* pk *and proves using the NIZK that the value encrypted in the ciphertext is a witness to the language, the proof now also consisting of the ciphertext. Extraction is done by decrypting the ciphertext to obtain the witness. For simulation, one simply encrypts the* 0 *string, and uses the simulator of the underlying NIZK. Using a rate-1 NIZK [BDS25] and rate-1 PKE scheme, gives us the necessary efficiency in the scheme.*

3.4 Fully Homomorphic Encryption

Syntax. A fully homomorphic encryption scheme FHE consists of a fixed ciphertext size $\ell_{\mathsf{ctxt}} = \ell_{\mathsf{ctxt}}(\lambda)$ and the following polynomial time algorithms.

$\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ This is a probabilistic algorithm that takes as input a security parameter 1^λ. It outputs a public key pk and a secret key sk.

$\mathsf{Enc}(\mathsf{pk}, m) \to c$. This is a probabilistic algorithm that takes as input a public key pk and a message $m \in \{0,1\}^n$. It outputs a ciphertext $c \in \{0,1\}^{n \cdot \ell_{\mathsf{ctxt}}}$.

$\mathsf{Dec}(\mathsf{sk}, c) \to m$. This is a deterministic algorithm that takes as input a secret key pk and a ciphertext $c \in \{0,1\}^{n \cdot \ell_{\mathsf{ctxt}}}$. It outputs a bit $m \in \{0,1\}^n$.

$\mathsf{Eval}(\mathsf{pk}, f, c) \to c^*$. This is a deterministic algorithm that takes as input a public key pk, a circuit representing a function $f : \{0,1\}^n \mapsto \{0,1\}^u$ and ciphertext $c \in \{0,1\}^{n \cdot \ell_{\mathsf{ctxt}}}$. It outputs a ciphertext $c^* \in \{0,1\}^{u \cdot \ell_{\mathsf{ctxt}}}$.

Definition 4 (FHE)**.** *A fully homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is required to satisfy the following properties:*

Encryption Correctness. *For any choice of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*, and any* $m \in \{0,1\}^n$ *and any* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ *we have* $\mathsf{Dec}(\mathsf{sk}, c) = m$*.*

Evaluation Correctness. *For any choice of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*, and any ciphertext* $c \in \{0,1\}^{n \cdot \ell_{\mathsf{ctxt}}}$ *such that* $\mathsf{Dec}(\mathsf{sk}, c) = m \in \{0,1\}^n$ *and any circuit* $f : \{0,1\}^n \mapsto \{0,1\}^u$*, if we set* $c^* = \mathsf{Eval}(f, c)$ *then* $\mathsf{Dec}(\mathsf{sk}, c^*) = f(m)$*.*

Security. *The encryption scheme must satisfy semantic security, i.e. for every* $\lambda \in \mathbb{N}$ *and all pairs of messages* $m_0, m_1 \in \{0,1\}^n$*, it holds that*

$$\mathsf{Enc}(\mathsf{pk}, m_0) \approx_c \mathsf{Enc}(\mathsf{pk}, m_1),$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*.*

Compactness. *The ciphertext must be compact, i.e. there exists a polynomial* $\mathsf{p}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*, message* $m \in \{0,1\}^n$*, and circuit* f *with input size* n*,*

$$|\mathsf{Eval}(f, \mathsf{Enc}(\mathsf{pk}, m))| = |f(m)| \cdot \mathsf{p}(\lambda).$$

Note that this also implies that a fresh encryption of m *has size* $|m| \cdot \mathsf{p}(\lambda)$*.*

3.5 Somewhere Extractable Hash

A SEH scheme is a tuple of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ described below.

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$**:** A randomized "normal mode" key generation polynomial time algorithm that takes as input the security parameter 1^λ and outputs a hash key hk.

$\mathsf{TGen}(1^\lambda, i) \to (\mathsf{hk}, \mathsf{td})$**:** A randomized "trapdoor mode" key generation polynomial time algorithm that takes as input the security parameter 1^λ, an extraction index i and outputs a hash key hk and a trapdoor td.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{rt}$**:** This is a deterministic polynomial time algorithm that on input the hash key hk and input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, outputs a hash rt.

$\mathsf{Open}(\mathsf{hk}, x, i) \to \rho$**:** This is a deterministic polynomial time algorithm that on input the hash key hk, input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$ and index $i \in [N]$ outputs an opening ρ.

$\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, N, j, b, \rho) \to 0/1$**.** This is a deterministic poly-time algorithm that takes as input a hash key hk, a hash value rt, an input length N, an index $j \in [N]$, a bit $b \in \{0,1\}$ and an opening ρ. It outputs a bit (1 to accept, 0 to reject).

$\mathsf{Extract}(\mathsf{td}, \mathsf{rt}) \to y$**:** This is a deterministic poly-time algorithm that takes as input a trapdoor td and a hash rt and outputs a bit $y \in \{0,1\}$ or $\perp$.

Furthermore, we require the commitment scheme to satisfy the following properties.

Key Indistinguishability. For any non-uniform PPT adversary $\mathcal{A}$, and any index $i \in [N]$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left| \Pr_{\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda)} [\mathcal{D}(\mathsf{hk}) = 1] - \Pr_{(\mathsf{hk},\mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda)} [\mathcal{D}(\mathsf{hk}) = 1] \right| \leq \mathsf{negl}(\lambda)$$

Opening completeness. For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any $x \in \{0,1\}^N$, and any index $j \in [N]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, N, j, x_j, \rho) = 1 \ : \ \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathsf{rt} = \mathsf{Hash}(\mathsf{hk}, x), \\ \rho = \mathsf{Open}(\mathsf{hk}, x, j) \end{array} \right] = 1 - \mathsf{negl}(\lambda).$$

Succinctness. In the completeness experiment above, we have that $|\mathsf{hk}| + |\mathsf{rt}| + |\rho| = \mathsf{poly}(\lambda)$.

Extraction Correctness. For any $N \leq 2^\lambda$, any index $j \in [N]$, any trapdoor key $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, j)$, any hash rt, any bit $y \in \{0,1\}$, and any opening proof ρ, we have

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, N, j, y, \rho) = 1 \ \wedge \\ \mathsf{Extract}(\mathsf{td}, \mathsf{rt}) \neq y \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Remark 5. *As in the case of hash trees, one can extend the above definition to open, and extract strings rather than bits. The input that is hashed is now an N-sized tuple of strings.*

Remark 6 ([HW15] Somewhere Extractable Hash)**.** *In this work we use a specific somewhere extractable hash scheme based on FHE presented in [HW15]. Their construction combines Merkle trees with fully homomorphic encryption to instantiate the primitive. For this work it suffices to specify how the hash key generation and extraction algorithms work.*

$\mathsf{Gen}(1^\lambda)$**:** *Samples an* FHE *key pair* $(\mathsf{pk}, \mathsf{sk})$*, and outputs* $\mathsf{hk} = (\mathsf{Enc}(\mathsf{pk}, 0), \mathsf{pk})$.

$\mathsf{TGen}(1^\lambda, i)$**:** *Samples an* FHE *key pair* $(\mathsf{pk}, \mathsf{sk})$*, and outputs* $\mathsf{hk} = (\mathsf{Enc}(\mathsf{pk}, i), \mathsf{pk})$ *and* $\mathsf{td} = \mathsf{sk}$.

$\mathsf{Extract}(\mathsf{td}, \mathsf{rt}) \to y$**:** *Outputs* $\mathsf{Dec}(\mathsf{sk}, \mathsf{rt})$.

At a high level, the Hash *algorithm ensures that the root is an FHE ciphertext encrypting the i-th bit of x. We refer the reader to [HW15] for full details of the construction. Since the* Gen *and* TGen *algorithms sample an FHE key, we will find it convenient to pass the key pair* $(\mathsf{pk}, \mathsf{sk})$ *explicitly to the algorithm. The properties of the somewhere extractable hash only hold when such keys, now provided as inputs, are sampled according to the correct distribution.*

Further we will use the property that extraction correctness holds, not just for a fresh *encryption of i, but for* any *ciphertext encrypting i.*

Finally, we use a property of extraction correctness for ciphertexts encrypting invalid indices $i \notin [N]$ (in that case, the root should decrypt to $\perp$). That is, for any $N \leq 2^\lambda$, any hash key hk encrypting $i \notin [N]$, any hash rt, any index j and bit $y \in \{0,1\}$, and any opening proof ρ, we have

$$\Pr\left[\begin{array}{l}\mathsf{Verify}(\mathsf{hk},\mathsf{rt},N,j,y,\rho)=1 \wedge \\ \mathsf{Extract}(\mathsf{td},\mathsf{rt}) \neq \perp\end{array}\right] \leq \mathsf{negl}(\lambda).$$

4 One-Round Two-Party Secure Computation

In this section we define a two-party secure computation in the client-server model. The protocol proceeds in one round: a message from the client followed by a message from the server where only the client obtains an output. In the introduction we refer to this notion as non-interactive secure computation (NISC).

Syntax. Let $\mathsf{F} = \{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a two-input polynomial-time computable function. A 1-round 2-party computation protocol for F in the CRS model consist of the following polynomial-time algorithms.

$\mathsf{Gen}(1^\lambda) \to (\mathsf{crs})$ This is a probabilistic algorithm that takes as input a security parameter 1^λ. It outputs a common reference string crs.

$\mathsf{Client}(\mathsf{crs}, x) \to (\mathsf{st}, c)$. This is a probabilistic algorithm that takes as input a common reference string crs and a client input $x \in \{0,1\}^*$. It outputs a state st and a client message c.

$\mathsf{Server}(\mathsf{crs}, y, c) \to s$. This is a probabilistic algorithm that takes as input a common reference string crs, a server input $y \in \{0,1\}^*$ and a client message c. It outputs a server message s.

$\mathsf{Out}(\mathsf{crs}, \mathsf{st}, s) \to z$. This is a deterministic algorithm that takes as input a common reference string crs, a state st and a server message s. It outputs the client output z.

Next we define full security following the standard Real-Ideal paradigm. For convenience we define simulation security for the client and the server separately.

Definition 5 (1-round 2-party Secure Computation)**.** *A 1-round 2-party computation protocol for* F *in the CRS model* $(\mathsf{Gen}, \mathsf{Client}, \mathsf{Server}, \mathsf{Out})$ *is required to satisfy the following properties:*

Correctness. *For any $\lambda \in \mathbb{N}$ and inputs $x, y \in \{0,1\}^*$:*

$$\Pr\left[\mathsf{Out}(\mathsf{crs},\mathsf{st},s) = \mathsf{F}_\lambda(x,y) \quad : \quad \begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x) \\ s \leftarrow \mathsf{Server}(\mathsf{crs}, y, c)\end{array}\right] = 1.$$

Client security. *For every polynomial-size adversary $\mathcal{A}$ corrupting the server, and polynomial P there exists a polynomial-size simulator* Sim *such that for every polynomial-size distinguisher $\mathcal{D}$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and x such that $|x| \leq P(\lambda)$:*

$$|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr[\mathcal{D}(\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)) = 1]| \leq \mu(\lambda) \ .$$

where the experiments REAL *and* IDEAL *are defined as follows:*

$\underline{\mathsf{REAL}_{\mathcal{A}}(\lambda, x)}$
$\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$
$(\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x)$
$(z^*, s) \leftarrow \mathcal{A}(\mathsf{crs}, c)$
$z \leftarrow \mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)$
Output (z^*, z)

$\underline{\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)}$
$(z^*, y) \leftarrow \mathsf{Sim}(1^\lambda)$
If $y = \perp$, *set* $z \leftarrow \perp$
If $y \neq \perp$, *set* $z \leftarrow \mathsf{F}_\lambda(x, y)$
Output (z^*, z)

Server security. *For every polynomial-size adversary* $\mathcal{A}$ *corrupting the client, and polynomial* P *there exists a polynomial-size simulator* Sim *such that for every polynomial-size distinguisher* $\mathcal{D}$ *there exists a negligible function* μ *such that for every* $\lambda \in \mathbb{N}$ *and* y *such that* $|y| \leq P(\lambda)$*:*

$$|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, y)) = 1] - \Pr[\mathcal{D}(\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, y)) = 1]| \leq \mu(\lambda) \ .$$

where the experiments REAL *and* IDEAL *are defined as follows:*

$\underline{\mathsf{REAL}_{\mathcal{A}}(\lambda, y)}$
$\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$
$c \leftarrow \mathcal{A}(\mathsf{crs})$
$s \leftarrow \mathsf{Server}(\mathsf{crs}, y, c)$
Output $\mathcal{A}(\mathsf{crs}, s)$

$\underline{\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, y)}$
$x \leftarrow \mathsf{Sim}(1^\lambda)$
Output $\mathsf{Sim}(1^\lambda, \mathsf{F}_\lambda(x, y))$

4.1 Weak Client Security

In this section we define a weaker notion of security against malicious servers where the view of the client and the server in the real execution are simulated separately. In the introduction we refer to this notion as split-simulation. In more detail, a 1-round 2-party computation protocol for F in the CRS model has weak client security if instead of the client security requirement it satisfies the following two requirements: input privacy and weak simulation-security.

Definition 6 (Input Privacy (for the Client))**.** *A 1-round 2-party computation protocol for* F *in the CRS model* (Gen, Client, Server, Out) *satisfies input privacy if for every polynomial size distinguisher* $\mathcal{D}$ *and polynomial* P *there exists a negligible function* μ *such that for any* $\lambda \in \mathbb{N}$ *and pair of inputs* x_0, x_1 *such that* $|x_0| = |x_1| \leq P(\lambda)$ *it holds that*

$$\Pr\left[\begin{array}{lll} & & b \leftarrow \{0,1\} \\ \mathcal{D}(\mathsf{crs}, c) = b & : & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) \\ & & (\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Definition 7 (Weak Simulation-Security (for the Client))**.** *A 1-round 2-party computation protocol for* F *in the CRS model* (Gen, Client, Server, Out) *satisfies weak simulation-security if for every polynomial-size adversary* $\mathcal{A}$ *corrupting the server, and polynomial* P *there exists a polynomial-size simulator* Sim *such that for every polynomial-size distinguisher* $\mathcal{D}$*, large enough* $\lambda \in \mathbb{N}$ *and* x *such that* $|x| \leq P(\lambda)$*:*

$$|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr[\mathcal{D}(\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)) = 1]| \leq \frac{1}{P(\lambda)} \ .$$

where the experiments REAL *and* IDEAL *are defined as follows:*

$\underline{\mathsf{REAL}_{\mathcal{A}}(\lambda, x)}$
$\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$
$(\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x)$
$s \leftarrow \mathcal{A}(\mathsf{crs}, c)$
$z \leftarrow \mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)$
Output z

$\underline{\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)}$
$y \leftarrow \mathsf{Sim}(1^\lambda)$
If $y = \perp$, *set* $z \leftarrow \perp$
If $y \neq \perp$, *set* $z \leftarrow \mathsf{F}_\lambda(x, y)$
Output z

4.2 Input-Dependent Simulation-Security

In this section we consider one-round client-server secure computation for functions where the server's input is an array. In this setting we introduce a relaxed notion of security against a malicious server where the simulator can depend on the honest client's input, however, the distribution of each entry in the simulated server array appears independent of the client's input. In what follows we use this notion of input-dependent simulation as a stepping stone towards achieving weak-simulation security for the client.

Let $n(\lambda), \ell(\lambda), m(\lambda)$ be polynomially bounded functions. Let F be a polynomial-time computable function,

$$\mathsf{F} = \{\mathsf{F}_\lambda : \{0,1\}^{n(\lambda)} \times (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^*\}_{\lambda \in \mathbb{N}} \ .$$

Definition 8 (Input-dependent Simulation-Security (for the Client))**.** *A 1-round 2-party computation protocol for* F *in the CRS model* $(\mathsf{Gen}, \mathsf{Client}, \mathsf{Server}, \mathsf{Out})$ *satisfies input-dependent simulation-security if for every polynomial-size adversary* $\mathcal{A}$ *corrupting the server, and polynomial* P *there exists a polynomial-size input-dependent simulator* Sim *that satisfies the following requirements:*

Client-output indistinguishability. *For every polynomial-size distinguisher* $\mathcal{D}$*, large enough* $\lambda \in \mathbb{N}$ *and* $x \in \{0,1\}^{n(\lambda)}$*:*

$$|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr[\mathcal{D}(\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)) = 1]| \leq \frac{1}{P(\lambda)} \ .$$

where the experiments REAL *and* IDEAL *are defined as follows:*

$\underline{\mathsf{REAL}_{\mathcal{A}}(\lambda, x)}$
$\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$
$(\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x)$
$s \leftarrow \mathcal{A}(\mathsf{crs}, c)$
$z \leftarrow \mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)$
Output z

$\underline{\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)}$
$y \leftarrow \mathsf{Sim}(1^\lambda, x)$
If $y = \perp$, *set* $z \leftarrow \perp$
If $y \neq \perp$, *set* $z \leftarrow \mathsf{F}_\lambda(x, y)$
Output z

Server-input indistinguishability. *For every polynomial size distinguisher* $\mathcal{D}$ *there exists a negligible function* μ *such that for any* $\lambda \in \mathbb{N}$*, pair of inputs* $x_0, x_1 \in \{0,1\}^{n(\lambda)}$ *and* $i \in [m(\lambda)]$ *it holds that*

$$\Pr\left[\ \mathcal{D}(y_i) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\} \\ y \leftarrow \mathsf{Sim}(1^\lambda, x_b) \end{array} \ \right] \leq \frac{1}{2} + \mu(\lambda).$$

where y_i *is set to* $\perp$ *if* $y = \perp$*.*

Server-input indistinguishability with a global predicate. The notion of server-input indistinguishability requires that for a pair of simulated server inputs y^0 and y^1, and for every i, the i-th entry in the two inputs are computationally indistinguishable. However, the full server inputs y^0 and y^1 may be distinguishable. Next we introduce a strengthening of input-dependent simulation with respect to a global predicate G, where we additionally require that y^0 and y^1 cannot be distinguished by G.

Let G be a function of the server's input:

$$\mathsf{G} = \{\mathsf{G}_\lambda : (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^*\}_{\lambda\in\mathbb{N}} \ .$$

We say that a 1-round 2-party computation protocol for F in the CRS model $(\mathsf{Gen}, \mathsf{Client}, \mathsf{Server}, \mathsf{Out})$ satisfies input-dependent simulation-security with respect to G if the server-input indistinguishability requirement is replaced by the following stronger requirement.

Definition 9 (Server-input indistinguishability with respect to G.)**.** *For every polynomial size distinguisher $\mathcal{D}$ there exists a negligible function μ such that for any $\lambda \in \mathbb{N}$, pair of inputs $x_0, x_1 \in \{0,1\}^{n(\lambda)}$ and $i \in [m(\lambda)]$ it holds that*

$$\Pr\left[\ \mathcal{D}(y_i, \mathsf{G}(y)) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\} \\ y \leftarrow \mathsf{Sim}(1^\lambda, x_b) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

where y_i and $\mathsf{G}(y)$ are set to $\perp$ if $y = \perp$.

5 Technical Tool: Monotone Coupling

In this section, we present a new coupling technique that allows us to construct simulators for predicates that check monotonicity of a database according to some total order. Previously, [BDKP22] constructed vPIR for the class of read-once bounded-space predicates. We also show how to handle the conjunction of a monotonicity and a read-once bounded-space predicate, thus strengthening [BDKP22].

In order to handle read-once bounded-space predicates, [BDKP22] used a coupling lemma that relates couplings and statistical distance. We make use of the following generalized theorem, providing a condition for coupling according to an arbitrary relation.

Definition 10 (Coupling with deficiency)**.** *Let X, Y be random variables over sets $\mathcal{X}$ and $\mathcal{Y}$ respectively. Let $f : \mathcal{X} \to A$ and $f : \mathcal{Y} \to B$ be two polynomial time computable functions, and let $R \subset A \times B$ be a relation. An R-coupling of X, Y with respect to f, f' with deficiency ϵ is a joint distribution $\rho = (X, Y)$ such that for $x \leftarrow X, y \leftarrow Y$ and $(x', y') \leftarrow \rho$ it holds that $x \equiv x', y \equiv y'$ and*

$$\Pr[(f(x'), f'(y')) \in R] = 1 - \epsilon \ .$$

We additionally denote $\Delta_R(f(X), f(Y)) \leq \epsilon$ if there exists an R-coupling of X, Y with respect to f, f' with deficiency $\leq \epsilon$. Finally, an R-coupling that achieves the minimum deficiency $\Delta_R(f(X), f'(Y))$ is called a maximal R-coupling with respect to f, f'.

Theorem 4 (Strassen's Theorem with deficiency [Kop24])**.** *Let X, Y be random variables over sets $\mathcal{X}$ and $\mathcal{Y}$ respectively. Let $f : \mathcal{X} \to A$ and $f : \mathcal{Y} \to B$ be two polynomial time computable functions, and let $R \subset A \times B$ be a relation. Then for any $\epsilon \geq 0$, it holds that $\Delta_R(f(X), f'(Y)) \leq \epsilon$ if and only if*

$$\forall U \subseteq A : \quad \Pr[f(X) \in U] \leq \Pr[f'(Y) \in N_R(U)] + \epsilon \ ,$$

where we define the neighboring set $N_R(U) = \{f'(y) \in B \colon \exists f(x) \in U : \ (f(x), f'(y)) \in R\}$.

Remark 7 (Strassen's Theorem for specific relations)**.** *If R is the equality relation, the above condition is equivalent to $SD(f(X), f'(Y)) \leq \epsilon$ (this is the coupling lemma used in [BDKP22]).*

If R is a total order given by $(f(x), f'(y)) \in R \iff f(x) \leq f'(y)$, the above condition is equivalent to $\forall t \in A : \; F_{f'(Y)}(t) \leq F_{f(X)}(t) + \epsilon$, where $F_{f(X)}, F_{f'(Y)}$ are the cumulative distribution functions of $f(X), f'(Y)$ respectively, $F_{f(X)}(t) = \Pr[f(X) < t]$.

We claim that for the class of binary relations checking monotonicity and equality of a small state, computational indistinguishability suffices to obtain the coupling condition above.

Claim 1 (Coupling according to monotonicity and state equality)**.** *Let $\ell(\lambda)$ be a polynomial and let $s(\lambda) = O(\log \lambda)$. Let $A = \{0,1\}^{s(\lambda)} \times \{0,1\}^{\ell(\lambda)}$. Let $R \subseteq A \times A$ be a binary relation checking monotonicity and state equality: $((s_0, y_0), (s_1, y_1)) \in R \iff s_0 = s_1$ and $y_0 \leq y_1$.*

Let X, Y be distributions over sets $\mathcal{X}$ and $\mathcal{Y}$ and $f : \mathcal{X} \to A$ and $f : \mathcal{Y} \to A$ be two polynomial time computable functions such that $f(X) \approx_c f'(Y)$. Then there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\Delta_R(f(X), f'(Y)) \leq \mathsf{negl}(\lambda)$.

Proof. We use the coupling condition given by Theorem 4. Assume towards contradiction there is a polynomial $p(\lambda)$ and sets $\{U_\lambda\}_{\lambda \in \mathbb{N}}$ such that for infinitely many λ,

$$\Pr[f(X) \in U] > \Pr[f'(Y) \in N_R(U)] + \frac{1}{p(\lambda)} \, .$$

We observe that the neighboring set $N_R(U)$ is given by $(s, y) \in N_R(U)$ iff $s \in S \wedge y \geq z(s)$, where $S \subseteq \{0,1\}^{s(\lambda)}$ and $z \colon S \to \{0,1\}^{\ell(\lambda)}$ are given by

$$\begin{aligned} S &= \{s \colon (s, y) \in U\} \, , \\ z(s) &= \min\{y \colon (s', y) \in U, s = s'\} \, . \end{aligned}$$

Since S is of polynomial size and $\ell(\lambda)$ is polynomial, the condition $(s, y) \in N_R(U)$ can be checked by a polynomial size distinguisher $\mathcal{D}$. Together with the fact that $U \subseteq N_R(U)$, this implies

$$\Pr[\mathcal{D}(f(X)) = 1] > \Pr[\mathcal{D}(f'(Y)) = 1] + \frac{1}{p(\lambda)} \, ,$$

in contradiction to $f(X) \approx_c f'(Y)$. □

5.1 Efficient simulation

claim 1 only proves the existence of a coupling. To obtain an efficient coupling algorithm (and thus an efficient simulator, rather than an unbounded one), we consider empirical distributions (given by uniform sampling from a polynomial-size list).

We first observe that for empirical distributions, the existence of a coupling implies an efficient coupling algorithm.

Claim 2 (Efficient coupling algorithm)**.** *Let $R \subseteq A \times B$ be an efficiently testable relation. Let $L_X \in \mathcal{X}^m$ and $L_Y \in \mathcal{Y}^m$ be lists, and let $\tilde{X}, \tilde{Y}$ be uniform distributions over L_X, L_Y respectively. Let $f : \mathcal{X} \to A$ and $f : \mathcal{X} \to B$ be two polynomial time computable functions. Then there is a $\mathsf{poly}(m)$-time algorithm that outputs a list of pairs $L_{X,Y} \in (\mathcal{X} \times \mathcal{Y})^m$, such that the uniform distribution over $L_{X,Y}$ is a maximal R-coupling of $\tilde{X}, \tilde{Y}$ with respect to f, f'.*

Proof sketch. The coupling algorithm is as follows:

1. Construct a bipartite graph G whose vertices are L_X and L_Y, and edges are defined by $(x_i, y_j) \in E \iff (f(x_i), f'(y_j)) \in R$.
2. Compute a maximum matching $M \subseteq E$.
3. Output $L_{X,Y}$ whose elements are all pairs in M, and an arbitrary pairing of the remaining $m - |M|$ elements of L_X and L_Y.

The uniform distribution over $L_{X,Y}$ has marginals $\tilde{X}$ and $\tilde{Y}$ since each element in L_X and L_Y appears exactly once in $L_{X,Y}$, thus it is a coupling, and its deficiency is $\leq 1 - \frac{|M|}{m}$. Furthermore, the converse also holds: each coupling with respect to f, f' of $\tilde{X}, \tilde{Y}$ of deficiency ϵ defines a matching of size $m(1-\epsilon)$. Thus the algorithm outputs the maximal R-coupling with respect to f, f'. □

Observation 1 (Sequential Coupling)**.** *As noted, one can view the output of the algorithm as simply the coupled list $L_{X,Y}$ where each element in L_X and L_Y appears exactly once in $L_{X,Y}$. This in turn means that one can run the same matching algorithm between $L_{X,Y}$ and $L_Z \in \mathcal{Z}^m$ to obtain a maximal R'-coupling of the distributions uniform on $L_{X,Y}$ and L_Z for a different (efficient) relation $R' \subseteq (A \times B) \times C$ with respect to f'' and f''' where $f'' : \mathcal{X} \times \mathcal{Y} \to A \times B$ and $f'''' : \mathcal{Z} \to C$ are two polynomial time computable functions.*

We claim that the coupling condition for the class of relations in claim 1 is still satisfied when considering two empirical distributions, sampled from two computationally indistinguishable distributions.

Claim 3 (Coupling empirical distributions)**.** *Let $\ell(\lambda)$ be a polynomial and let $s(\lambda) = O(\log \lambda)$. Let $A = \{0,1\}^{s(\lambda)} \times \{0,1\}^{\ell(\lambda)}$. Let $R \subseteq A \times A$ be a binary relation checking monotonicity and state equality: $((s_0, y_0), (s_1, y_1)) \in R \iff s_0 = s_1$ and $y_0 \leq y_1$.*

Let X, Y be distributions over sets $\mathcal{X}$ and $\mathcal{Y}$ and $f : \mathcal{X} \to A$ and $f : \mathcal{Y} \to A$ be two polynomial time computable functions such that $f(X) \approx_c f'(Y)$. For any polynomial $P(\lambda)$, there exists a polynomial $\kappa(\lambda)$ and a negligible function $\mathsf{negl}(\lambda)$ such that if we define L_X, L_Y by sampling $m = \kappa(\lambda)$ elements from X, Y respectively, and consider the empirical distributions $\tilde{X}, \tilde{Y}$ given by uniform sampling over L_X, L_Y, it holds that

$$\Pr\left[\Delta_R(f(\tilde{X}), f'(\tilde{Y})) \leq \frac{1}{P(\lambda)}\right] \geq 1 - \mathsf{negl}(\lambda) ,$$

where the probability is over the randomness used for sampling L_X, L_Y.

Proof. First, we argue that it suffices to prove the claim for the case when $f(X)$ and $f'(Y)$ are identically distributed. The condition $\Delta_R(f(\tilde{X}), f'(\tilde{Y})) \leq \frac{1}{P(\lambda)}$ can be checked efficiently by an algorithm that receives lists L_X, L_Y, runs the coupling algorithm given by claim 2, and checks if the deficiency of the resulting coupling is $\leq \frac{1}{P(\lambda)}$. This requires f and f' to be efficiently computable. Thus, if the claim holds for lists L_X, L'_X sampled independently from X but not for lists L_X, L_Y, then by a hybrid argument, we can distinguish between $f(X)$ and $f'(Y)$, contradicting $f(X) \approx_c f'(Y)$.

We now recall from claim 1, that neighboring sets for our relation are of the form $(s, y) \in N_R(U) \iff s \in S \wedge y \geq z(s)$, for some $S \subseteq \{0,1\}^{s(\lambda)}$ and $z \colon S \to \{0,1\}^{\ell(\lambda)}$. We denote

$$F_{f(X)}(s, z) = \Pr\left[\ s = s' \wedge y \geq z \quad : \quad (s', y) \leftarrow f(X)\ \right] ,$$

and observe that $\Pr[f(X) \in N_R(U)] = \sum_{s \in S} F_{f(X)}(s, z(s))$. Note that by $(s', y) \leftarrow f(X)$, we mean sampling from X and applying the deterministic function f.

Thus, if we have $\sum_{s \in S} F_{f(\tilde{X})}(s, z(s)) - F_{f'(\tilde{Y})}(s, z(s)) \leq \frac{1}{P(\lambda)}$ for any $S, z(\cdot)$ given by some neighboring set, then by the coupling condition in Theorem 4, it holds that $\Delta_R(f(\tilde{X}), f'(\tilde{Y})) \leq \frac{1}{P(\lambda)}$.

We turn to proving this event holds except with negligible probability. Let $t \in \mathbb{N}$. By Chernoff-Hoeffding, we have that for every $s \in \{0,1\}^{s(\lambda)}$ and $z \in \{0,1\}^{\ell(\lambda)}$,

$$\Pr\left[F_{f(X)}(s, z) - F_{f(\tilde{X})}(s, z) > \frac{t}{m}\right] < e^{-\frac{2t^2}{m}} .$$

Applying the same bound on $Y, \tilde{Y}$, using the assumption that $f(X) \equiv f'(Y)$, and applying a union bound, we get

$$\Pr\left[F_{f(\tilde{X})}(s, z) - F_{f'(\tilde{Y})}(s, z) > \frac{2t}{m}\right] < 2e^{-\frac{2t^2}{m}} .$$

Now, we observe that since $f(\tilde{X}), f'(\tilde{Y})$ each have support size $\leq m$, neighboring sets can only be given by $S, z \colon S \to \tilde{Z}$ where $\left|\tilde{Z}\right| \leq 2m$. Applying a union bound, we have

$$\Pr\left[\forall s \in \{0,1\}^{s(\lambda)}, z \in \tilde{Z} : \quad F_{f(\tilde{X})}(s, z) - F_{f'(\tilde{Y})}(s, z) \leq \frac{2t}{m}\right] > 1 - 2^{s(\lambda)} \cdot 4me^{-\frac{2t^2}{m}} .$$

And if the event above holds, then we can sum over subsets $S \subseteq \{0,1\}^{s(\lambda)}$, and get

$$\Pr\left[\forall S \subseteq \{0,1\}^{s(\lambda)}, z \colon S \to \tilde{Z} : \quad \sum_{s \in S} F_{f(\tilde{X})}(s, z) - \sum_{s \in S} F_{f'(\tilde{Y})}(s, z(s)) \leq \frac{2^{s(\lambda)} \cdot 2t}{m}\right] > 1 - 2^{s(\lambda)} \cdot 4me^{-\frac{2t^2}{m}} ,$$

which is exactly the coupling condition we wanted. Setting $\kappa(\lambda)$ to be a large enough polynomial such that $2^{s(\lambda)}\kappa(\lambda)^{-0.49} \leq \frac{1}{P(\lambda)}$, and setting $t = \frac{\kappa(\lambda)^{0.51}}{2}$, we get that the deficiency above is $\leq \frac{1}{P(\lambda)}$, and the probability is $1 - \mathsf{negl}(\lambda)$, and thus

$$\Pr\left[\Delta_R(f(\tilde{X}), f'(\tilde{Y})) \leq \frac{1}{P(\lambda)}\right] \geq 1 - \mathsf{negl}(\lambda) .$$

□

6 General Theorem: Input-Dependent Simulation-Security

In this section we present 1-round 2-party protocols satisfying input-dependent simulation-security, for two general function classes: sorted bounded-state functions, and sorted lookup functions.

We first state our theorem for sorted bounded-state functions. This class of functions suffices to obtain protocols for private set membership.

Definition 11 (Function Computable with Bounded State)**.** *Let $s(\lambda)$ be a polynomially bounded function. A function $\mathsf{F} = \{\mathsf{F}_\lambda : \{0,1\}^{n(\lambda)} \times (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda \in \mathbb{N}}$ is computable with state of size s if there exists an efficiently computable function $\mathsf{f} = \{\mathsf{f}_\lambda : \{0,1\}^* \times \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{s(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda \in \mathbb{N}}$ and $z_0 \in \{0,1\}^{s(\lambda)}$ such that:*

$$\mathsf{F}_\lambda(x, \epsilon) \equiv z_0 ,$$
$$\mathsf{F}_\lambda(x, (y_1, \ldots, y_{m(\lambda)})) \equiv \mathsf{f}_\lambda(x, y_{m(\lambda)}, \mathsf{F}(x, (y_1, \ldots, y_{m(\lambda)-1}))) .$$

Theorem 5. *Let*

$$\mathsf{F} = \{\mathsf{F}_\lambda : \{0,1\}^* \times (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda\in\mathbb{N}}$$
$$\mathsf{G} = \{\mathsf{G}_\lambda : (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda\in\mathbb{N}}$$

be functions computable with state of size $s(\lambda) = O(\log\lambda)$. Let $\mathsf{R} = \{\mathsf{R}_\lambda \subseteq \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{\ell(\lambda)}\}$ be an efficiently computable total order (either strict or non-strict). Let $\mathsf{F}' = \{\mathsf{F}'_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathsf{G}' = \{\mathsf{G}'_\lambda\}_{\lambda\in\mathbb{N}}$ be the functions:

$$\mathsf{F}'_\lambda(x, y = (y_1, \ldots, y_{m(\lambda)})) = \begin{cases} \mathsf{F}_\lambda(x,y), & \text{if } \forall i:\ (y_i, y_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases},$$
$$\mathsf{G}'_\lambda(y = (y_1, \ldots, y_{m(\lambda)})) = \begin{cases} \mathsf{G}_\lambda(y), & \text{if } \forall i:\ (y_i, y_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases}.$$

Assuming NIZK, FHE and BARG, there exists a 1-round 2-party computation protocol for the function F' in the CRS model with input-dependent simulation-security with respect to G' (Definitions 8 and 9) where the client's message is of size $n(\lambda)\cdot\mathsf{poly}(\lambda)$ and the server's message is of size $\ell(\lambda)\cdot\mathsf{poly}(\lambda)$. The protocol additionally satisfies server security (Definition 5) and client input privacy (Definition 6).

To support dictionary lookup (or more generally, VDL with UP search), where dictionary values may be longer than the state, we consider a different class of functions which we call sorted lookup functions.

Theorem 6. *Let*

$$\mathsf{Search} = \{\mathsf{Search}_\lambda : \{0,1\}^* \times (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda\in\mathbb{N}}$$
$$\mathsf{G} = \{\mathsf{G}_\lambda : (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda\in\mathbb{N}}$$

be functions computable with state of size $s(\lambda) = O(\log\lambda)$. Let $\mathsf{R} = \{\mathsf{R}_\lambda \subseteq \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{\ell(\lambda)}\}$ be an efficiently computable total order (either strict or non-strict).

Let $\mathsf{Project}\colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{\ell(\lambda)}$ be a poly-time function. Define the indexing function

$$\mathsf{Index}(x, y) = \begin{cases} \mathsf{Project}(y[x]), & \text{if } x \in [m(\lambda)] \\ \bot, & \text{otherwise} \end{cases}.$$

Let $\mathsf{F} = \{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathsf{G}' = \{\mathsf{G}'_\lambda\}_{\lambda\in\mathbb{N}}$ be the functions:

$$\mathsf{F}_\lambda(x, y = (y_1, \ldots, y_{m(\lambda)})) = \begin{cases} \mathsf{Index}(\mathsf{Search}_\lambda(x,y), y), & \text{if } \forall i:\ (y_i, y_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases},$$
$$\mathsf{G}'_\lambda(y = (y_1, \ldots, y_{m(\lambda)})) = \begin{cases} \mathsf{G}_\lambda(y), & \text{if } \forall i:\ (y_i, y_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases}.$$

Assuming NIZK, FHE and BARG, there exists a 1-round 2-party computation protocol for the function F in the CRS model with input-dependent simulation-security with respect to G' (Definitions 8 and 9) where the client's message is of size $n(\lambda)\cdot\mathsf{poly}(\lambda)$ and the server's message is of size $\ell(\lambda)\cdot\mathsf{poly}(\lambda)$. The protocol additionally satisfies server security (Definition 5) and client input privacy (Definition 6).

6.1 Sorted bounded-state functions

6.1.1 Protocol.

We present the one-round two-party secure computation protocol for the function $\mathsf{F}' = \{\mathsf{F}'_\lambda\}_{\lambda\in\mathbb{N}}$ defined in Theorem 5.

Building Blocks. We use the following components for our construction:

- A hash tree scheme $\mathsf{HT} = (\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Open}, \mathsf{HT.Verify})$.
- A fully homomorphic encryption (FHE) scheme (Definition 4) $\mathsf{FHE} = (\mathsf{FHE.Gen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$.
- A somewhere extractable batch argument (BARG) scheme (Definition 2) $\mathsf{BARG} = (\mathsf{BARG.Gen}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify}, \mathsf{BARG.Extract})$

Overview. The overall idea of the protocol is for the client to send an FHE $\widehat{c}_0$ of its input x to the server, and have the server compute the encrypted output of F' by performing homomorphic operations on the ciphertext. The client can then simply decrypt the ciphertext to learn its output.

In particular, since the function F can be computed in an iterative manner using f_λ, at step i of this iterative computation the server only needs input y_i defining $\widehat{c}_i$ to be the output of the FHE evaluation f_λ on $\widehat{c}_{i-1}$ using additional input y_i. Further, the output of F' is $\perp$ unless for all i, $(y_{i-1}, y_i) \in \mathsf{R}_\lambda$. To prove that the server updated the ciphertext at each step, while also maintaining the relative ordering, the server computes a BARG that every step i is computed correctly. The iterative (or local) property of both R and F allow the server to compute such a proof. The client then only decrypts if the BARG proof it receives verifies.

For the description of this protocol, we abuse the notation for FHE evaluation and state that $\mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{f}_\lambda, \widehat{c}, y, \widehat{c}_i)$ corresponds to the homomorphic evaluation of the function $\mathsf{f}_\lambda(\cdot, y, \cdot)$ on the plaintext underlying $\widehat{c}$ and $\widehat{c}_i$.

Turing machine for BARGs. We next describe the family of Turing machine $\mathsf{M} = \{\mathsf{M}_\lambda\}_{\lambda\in\mathbb{N}}$ that the server will need to provide a BARG proof for. The machine M_λ hardcodes the hash key hk, tree root rt, initial and final states $\mathsf{z}_{\mathsf{g,init}}$ and $\mathsf{z}_{\mathsf{g,final}}$ for g, encrypted input $\widehat{c}$, and encrypted initial and final states $\widehat{c}_{\mathsf{init}}$ and $\widehat{c}_{\mathsf{final}}$ for f_λ.

$\underline{\mathsf{M}_\lambda[\mathsf{hk}, \mathsf{rt}, \mathsf{z}_{\mathsf{g,init}}, \mathsf{z}_{\mathsf{g,final}}, \widehat{c}, \widehat{c}_{\mathsf{init}}, \widehat{c}_{\mathsf{final}}](i, w_i)}$

1. Parse $w_i = (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i)$.
2. if $i = 1$, check $\widehat{c}_0 = \widehat{c}_{\mathsf{init}}$ and $\mathsf{z}_{\mathsf{g},0} = \mathsf{z}_{\mathsf{g,init}}$.
3. if $i = m(\lambda)$, check $\widehat{c}_{m(\lambda)} = \widehat{c}_{\mathsf{final}}$ and $\mathsf{z}_{\mathsf{g},m(\lambda)} = \mathsf{z}_{\mathsf{g,final}}$.
4. Check $\widehat{c}_i = \mathsf{FHE.Eval}(\mathsf{f}_\lambda, \widehat{c}, y_i, \widehat{c}_{i-1})$.
5. Check $\mathsf{z}_{\mathsf{g},i} = \mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1})$.
6. Check $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{rt}, i, (\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i), \rho_i) = 1$.
7. if $i > 1$, check $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{rt}, i-1, (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}), \rho_{i-1}) = 1$ and $(y_{i-1}, y_i) \in \mathsf{R}_\lambda$.

8. Accept if all the checks verify.

Figure 1: Turing Machine M_λ

Note that we can set t, the running time of M, to be the time it takes to perform the above checks, which is polynomial in its inputs.

Protocol. We now describe the protocol below.

$\underline{\mathsf{Gen}(1^\lambda)}$

1. $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{m(\lambda), m(\lambda)\})$.
2. $\mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$.
3. Output $\mathsf{crs} := (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$.

$\underline{\mathsf{Client}(\mathsf{crs}, x)}$

1. $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.
2. $\widehat{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$.
3. $\widehat{c}_{\mathsf{init}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{z}_{\mathsf{f},0})$.
4. $c := (\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}})$.
5. $\mathsf{st} := (c, \mathsf{sk})$.
6. Output (c, st)

$\underline{\mathsf{Server}(\mathsf{crs}, y = (y_1, \ldots, y_{m(\lambda)}), c)}$

1. Parse c as $(\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}})$, and crs as $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$.
2. Set $\widehat{c}_0 := \widehat{c}_{\mathsf{init}}$
3. Set $\mathsf{z}_{\mathsf{g},0} := \mathsf{z}_{\mathsf{g},\mathsf{init}}$
4. For each $i \in [m(\lambda)]$, compute $\widehat{c}_i := \mathsf{FHE.Eval}(\mathsf{f}_\lambda, \widehat{c}, y_i, \widehat{c}_{i-1})$.
5. For each $i \in [m(\lambda)]$, compute $\mathsf{z}_{\mathsf{g},i} := \mathsf{g}(y_i, \mathsf{z}_{\mathsf{g},i-1})$.
6. Set $\mathsf{z}_{\mathsf{g},\mathsf{final}} = \mathsf{z}_{\mathsf{g},m(\lambda)}$ and $\widehat{c}_{\mathsf{final}} = \widehat{c}_{m(\lambda)}$.
7. Compute the hash tree root $\mathsf{rt} = \mathsf{HT.Hash}(\mathsf{hk}, ((\mathsf{z}_{\mathsf{g},1}, \widehat{c}_1, y_1) \cdots, (\mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}, y_{m(\lambda)})))$.
8. Use hk, rt, $\mathsf{z}_{\mathsf{g},\mathsf{init}}$, $\mathsf{z}_{\mathsf{g},\mathsf{final}}$, $\widehat{c}$, $\widehat{c}_{\mathsf{init}}$ and $\widehat{c}_{\mathsf{final}}$ to define the Turing machine M_λ as in figure 1.
9. For every $i \in [m(\lambda)]$,

– $\rho_i = \mathsf{HT.Open}(\mathsf{hk}, ((\mathsf{z}_{\mathsf{g},1}, \widehat{c}_1, y_1) \cdots, (\mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}, y_{m(\lambda)})), i)$.

– Construct witness (i, w_i) for with $w_i := (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i)$.

10. Compute the BARG proof $\pi \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, (w_1, \ldots, w_{m(\lambda)}))$.

11. Output $s := (\pi, \mathsf{rt}, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)})$.

$\underline{\mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)}$

1. Parse s as $(\pi, \mathsf{rt}, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)})$, crs as $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$ and st as (c, sk).

2. Set $\mathsf{z}_{\mathsf{g,final}} = \mathsf{z}_{\mathsf{g},m(\lambda)}$ and $\widehat{c}_{\mathsf{final}} = \widehat{c}_{m(\lambda)}$.

3. Use hk, rt, $\mathsf{z}_{\mathsf{g,init}}$, $\mathsf{z}_{\mathsf{g,final}}$ $\widehat{c}$, $\widehat{c}_{\mathsf{init}}$ and $\widehat{c}_{\mathsf{final}}$ to define the Turing machine M_λ as in figure 1.

4. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 1$, set $z = \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$. Else set $z = \perp$.

5. Output z

Figure 2: Protocol Π for $\mathsf{F}' = \{\mathsf{F}'_\lambda\}$.

Efficiency. The client message is of size $|c| = |\mathsf{pk}| + |\widehat{c}| + |\widehat{c}_{\mathsf{init}}| = n(\lambda) \cdot \mathsf{poly}(\lambda)$ by the compactness of the FHE scheme. The server message is of size $|s| = |\pi| + |\mathsf{rt}| + |\mathsf{z}_{\mathsf{g},m(\lambda)}| + |\widehat{c}_{m(\lambda)}| = O(\ell) \cdot \mathsf{poly}(\lambda)$ by the efficiency of the BARG, where $|\pi| = O(\ell) \cdot \mathsf{poly}(\lambda)$. [9]

Security. For security, first we show that the protocol described in Fig. 2 satisfies input-dependent simulation security (for the client) Definition 8 for F'. In fact we show that it satisfies the stronger notion of server-indistinguishability with respect to G' (Definition 9).

Fix an adversary $\mathcal{A}$, and polynomial $P(\lambda)$. We define the simulator Sim below.

$\underline{\mathsf{Sim}(1^\lambda, x)}$

1. Set $\kappa(\lambda)$ as the polynomial claim 3 for $P'(\lambda) = 1/(8 \cdot m(\lambda) \cdot P(\lambda))$

2. Sample $\mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$.

3. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{m(\lambda), m(\lambda)\})$

4. Sample $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.

5. Compute ciphertexts $\widehat{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$, $\widehat{c}_{\mathsf{init}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, z_0)$ and set $c := (\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}})$.

6. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \widehat{c}_n) \leftarrow \mathcal{A}(\mathsf{crs}, c)$

7. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, output $\perp$. //If $\mathcal{A}$ causes abort sufficiently often, can simply output $\perp$

[9] One can achieve stronger efficiency by using rate-1 BARGs, but they are not required for our result.

8. For $i \in [m(\lambda)]$

   (a) Set the list L_i to be the empty list.

   (b) Repeat the following $\kappa(\lambda) \cdot \lambda \cdot P(\lambda)$ times

      i. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{i, i\})$.

      ii. Set $\mathsf{crs} := (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$. //Reuse the previously sampled hk

      iii. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}) \leftarrow \mathcal{A}(\mathsf{crs}, c)$. //Reuse previously computed ciphertexts

      iv. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, return to Item 8b.

      v. Extract from the proof, $w_i := \mathsf{BARG.Extract}(\tau, \pi)$. //For readability, the second w_i extracted is ignored.

      vi. Parse w_i as $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i)$.

      vii. Set $y'_{i-1} = y_{i-1}$, and $\widehat{c}'_i = \widehat{c}_i$. //to distinguish from y_{i-1} and $\widehat{c}_i$ in L_{i-1}

      viii. Set $\widetilde{\mathsf{z}}_i = \mathsf{z}_{\mathsf{g},m(\lambda)}$ //to distinguish the final state $\mathsf{z}_{\mathsf{g},m(\lambda)}$ output by $\mathcal{A}$ in each list

      ix. Add $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i) \in \{0,1\}^{m'}$ to L_i. //m' is the length of each element in the list

   (c) If $|\mathsf{L}_i| < \kappa(\lambda)$ output $\perp$. Otherwise, truncate its length to exactly $\kappa(\lambda)$.

9. Set $\mathsf{L}'_1 = \mathsf{L}_1 \in \{0,1\}^{m' \cdot \kappa(\lambda)}$.

10. For every $i \in [m(\lambda) - 1]$

   (a) Let $\mathsf{f}'_i : \{0,1\}^{m' \cdot i} \to \{0,1\}^{3 \cdot s(\lambda) + \ell(\lambda)}$ and $\mathsf{f}_{i+1} : \{0,1\}^{m'} \to \{0,1\}^{3 \cdot s(\lambda) + \ell(\lambda)}$ be functions defined as follows, where x and sk are hardcoded in the function:

$$\mathsf{f}'_i((\mathsf{z}_{\mathsf{g},j-1}, \widehat{c}_{j-1}, \widehat{c}'_j, y'_{j-1}, y_j, \widetilde{\mathsf{z}}_j)_{j \in [i]}) = (\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})), \widetilde{\mathsf{z}}_i), y_i$$
$$\mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}, \widetilde{\mathsf{z}}_{i+1}) = (\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_{i+1}), y'_i$$

   (b) Compute $\mathsf{L}'_{i+1} \in \{0,1\}^{m' \cdot (i+1) \cdot \kappa(\lambda)}$, the coupling of L'_i and L_{i+1} using the algorithm in claim 2.

11. Sample uniformly from the list $\mathsf{L}'_{m(\lambda)}$, i.e. $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i)_{i \in [m(\lambda)]} \leftarrow \mathsf{L}'_{m(\lambda)}$.

12. Output $y = (y_1, \ldots, y_{m(\lambda)})$.

Figure 3: Simulator Sim

6.1.2 Client-output Indistinguishability.

To prove client-output indistinguishability, we fix polynomial size distinguisher $\mathcal{D}$. We need to bound the following by $\leq 1/P(\lambda)$.

$$\left| \Pr_{\mathsf{REAL}} [\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr_{\mathsf{IDEAL}} [\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1] \right| \tag{1}$$

where $\mathsf{REAL}_{\mathcal{A}}(\lambda, x)$ and $\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)$ are as described in Definition 8. We abuse notation slightly, to additionally indicate by REAL and IDEAL the randomness used in the real and ideal experiments respectively.

We now define the following events. $\mathsf{ADV}_\perp$ is defined for both the real and ideal experiment, whereas $\mathsf{SHORT}_\perp$, $\mathsf{R}_\perp$ and $\mathsf{SIM}_\perp$ are only defined for the ideal experiment.

$$\begin{aligned}
\mathsf{ADV}_\perp &= \mathcal{A} \text{ sends } \pi \text{ that does not verify} \\
\mathsf{SHORT}_\perp &= \overline{\mathsf{ADV}_\perp} \wedge (\text{ Sim outputs } \perp \text{ in Item 8c for any } i) \\
\mathsf{SIM}_\perp &= \mathsf{ADV}_\perp \vee \mathsf{SHORT}_\perp \\
\mathsf{R}_\perp &= \overline{\mathsf{SIM}_\perp} \wedge (\text{ for some } i,\ (y_{i-1}, y_i) \notin \mathsf{R}_\lambda \ )
\end{aligned}$$

By definition, $\mathsf{ADV}_\perp$, $\mathsf{SHORT}_\perp$ and $\mathsf{R}_\perp$ are disjoint events in the ideal world. In the real world, the client in the real world outputs $\perp$ if the BARG proof π does not verify. Thus, in the real experiment, when the event $\mathsf{ADV}_\perp$ occurs, the output of the experiment is $\perp$. Correspondingly, in the ideal world if either $\mathsf{ADV}_\perp$, $\mathsf{SHORT}_\perp$ or $\mathsf{R}_\perp$ occurs, the ideal experiment output is $\perp$. Further, for F', in the event when none of these events occur, the output of the ideal experiment is $\mathsf{F}(x, y)$, i.e. F' does not evaluate to the $\perp$ case.

Aborting adversary. We separate out Eq. (1) depending on whether the adversary returns a BARG proof π that does not verify. From the description of both the protocol, and the simulator, the input to the distinguisher in the real and ideal experiments when the event $\mathsf{ADV}_\perp$ occurs is $\perp$. Thus, we have

$$\begin{aligned}
&\left|\Pr_{\mathsf{REAL}}[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr_{\mathsf{IDEAL}}[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1]\right| \\
&= \Bigg|\Pr_{\mathsf{REAL}}[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \mid \mathsf{ADV}_\perp] \Pr_{\mathsf{REAL}}[\mathsf{ADV}_\perp] + \Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \Pr_{\mathsf{REAL}}\left[\overline{\mathsf{ADV}_\perp}\right] \\
&\quad - \Pr_{\mathsf{IDEAL}}[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \mid \mathsf{ADV}_\perp] \Pr_{\mathsf{IDEAL}}[\mathsf{ADV}_\perp] - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_\perp}\right]\Bigg| \\
&\leq (\Pr[\mathcal{D}(\perp) = 1]) \left|\Pr_{\mathsf{REAL}}[\mathsf{ADV}_\perp] - \Pr_{\mathsf{IDEAL}}[\mathsf{ADV}_\perp]\right| \qquad (2) \\
&\quad + \left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \Pr_{\mathsf{REAL}}\left[\overline{\mathsf{ADV}_\perp}\right] - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_\perp}\right]\right| \qquad (3)
\end{aligned}$$

where the second inequality follows from the triangle inequality and the fact that the distinguisher gets $\perp$ in both cases.

For Eq. (2), observe that to check if $\mathcal{A}$ aborts, the input to $\mathcal{A}$ comes from identical distributions in the real and ideal experiment. Thus, $\left|\Pr_{\mathsf{REAL}}[\mathsf{ADV}_\perp] - \Pr_{\mathsf{IDEAL}}[\mathsf{ADV}_\perp]\right| = 0$. Thus, to bound Eq. (1) by $1/P(\lambda)$, it suffices to bound Eq. (3) by $1/P(\lambda)$.

Also, note that the above argument also implies that $\Pr_{\mathsf{REAL}}\left[\overline{\mathsf{ADV}_\perp}\right] = \Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_\perp}\right]$. We make the two following observations: (i) without loss of generality we can assume that $\Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_\perp}\right] > 1/P(\lambda)$;

and (ii) to bound Eq. (3), it suffices to show

$$\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]\right|\leq\frac{1}{P(\lambda)}. \tag{4}$$

For (ii), this follows directly from the fact that $\Pr_{\mathsf{REAL}}\left[\overline{\mathsf{ADV}_{\perp}}\right]=\Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_{\perp}}\right]$. For (i), if this were not the case, we can already bound Eq. (3) by $1/P(\lambda)$ since $\Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_{\perp}}\right]=\Pr_{\mathsf{REAL}}\left[\overline{\mathsf{ADV}_{\perp}}\right]\leq 1/P(\lambda)$.

Simulator abort for non-aborting adversary. It may be the case that despite $\mathsf{ADV}_{\perp}$ not occurring in the ideal experiment, the simulator Sim might still lead to an abort in the ideal experiment. Since in the real experiment, conditioned on $\mathsf{ADV}_{\perp}$ not occurring, the real experiment does not output an explicit $\perp$[10], it is important to bound the probability that Sim can lead to the ideal experiment output to be $\perp$. In particular, it can occur either when $\mathsf{SIM}_{\perp}$ occurs, or $\mathsf{R}_{\perp}$ occurs.

Case I: Insufficient List Size. Given that $\mathsf{ADV}_{\perp}$ does not occur, we want to argue that Sim does not abort except with negligible probability. In particular, we can rewrite Eq. (4) as

$$\begin{aligned}
&\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]\right|\\
&=\Bigg|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]\\
&\quad-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\perp)=1\,\middle|\,\mathsf{SHORT}_{\perp}\wedge\overline{\mathsf{ADV}_{\perp}}\right]\Pr_{\mathsf{IDEAL}}\left[\mathsf{SHORT}_{\perp}\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]\\
&\quad-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{SHORT}_{\perp}}\wedge\overline{\mathsf{ADV}_{\perp}}\right]\Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{SHORT}_{\perp}}\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]\Bigg|.
\end{aligned}$$

To bound this, we prove the following claim, that the event $\mathsf{SHORT}_{\perp}$ does not happen (except with negligible probability), if the event $\mathsf{ADV}_{\perp}$ does not occur.

Claim 4.

$$\Pr_{\mathsf{IDEAL}}\left[\mathsf{SHORT}_{\perp}\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]=\mathsf{negl}(\lambda).$$

Proof of claim 4. Fix an index i. Let us bound the probability that Sim aborts due to L_i not containing $\kappa(\lambda)$ many elements after $\kappa(\lambda)\cdot\lambda\cdot P(\lambda)$ iterations. For a single iteration, the probability that no element is added to L_i is,

$$\Pr_{\mathsf{IDEAL}}\left[\text{Element not added to }\mathsf{L}_i\text{ in a single iteration}\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]<1-1/P(\lambda)+\mathsf{negl}(\lambda).$$

An element is not added to the list if the BARG proof π does not verify. We have previously established that $\Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{ADV}_{\perp}}\right]>1/P(\lambda)$, and the process for determining whether to add an element is identical to checking for the event $\overline{\mathsf{ADV}_{\perp}}$, except that when checking for $\overline{\mathsf{ADV}_{\perp}}$, the $\mathsf{crs}_{\mathsf{BARG}}$ was generated for the index set $\{m(\lambda),m(\lambda)\}$ whereas in each iteration it is now generated for the index set $\{i,i\}$.

[10] It is possible that the function F might output a $\perp$, but such $\perp$ are handled as an output rather than than an explicit abort.

It thus suffices to argue that the probability with which $\mathcal{A}$ aborts is only negligibly different even though the $\mathsf{crs}_{\mathsf{BARG}}$ encodes a different index, i.e. the probability is $1 - 1/P(\lambda) + \mathsf{negl}(\lambda)$. This follows directly from the index hiding property (Definition 2) of the BARG. In particular, if the probability was not negligibly different, we can build an adversary $\mathcal{B}$ for the index hiding that uses $\mathcal{A}$ - $\mathcal{B}$ receives $\mathsf{crs}_{\mathsf{BARG}}$ from an external challenger, encoding either index set $\{m(\lambda), m(\lambda)\}$ or $\{i, i\}$. $\mathcal{B}$ computes the client message for $\mathcal{A}$ without requiring the knowledge of the BARG trapdoor τ, and if $\mathcal{A}$ aborts on the input the crs and client message, $\mathcal{B}$ outputs 0, else it chooses its guess at random.

Now, the probability that no element is added to L_i in each of the $\lambda \cdot P(\lambda)$ iterations is

$$\Pr_{\mathsf{IDEAL}}\left[\text{No element added to } \mathsf{L}_i \text{ in } \lambda \cdot P(\lambda) \text{ iterations} \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] < 1/2^{O(\lambda)}.$$

For the final list L_i to be shorter than $\kappa(\lambda)$ elements, it is necessarily the case that at least one of the $\kappa(\lambda)$ "blocks" of size $\lambda \cdot P(\lambda)$ had no elements added, and thus

$$\Pr_{\mathsf{IDEAL}}\left[|\mathsf{L}_i| < \kappa(\lambda) \text{ in } \kappa(\lambda) \cdot \lambda \cdot P(\lambda) \text{ iterations} \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] < \kappa(\lambda) \cdot 1/2^{O(\lambda)}.$$

Finally, by applying a union bound over all the $m(\lambda)$ indices, we have

$$\Pr_{\mathsf{IDEAL}}\left[\mathsf{SHORT}_\perp \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] < m(\lambda) \cdot \kappa(\lambda) \cdot 1/2^{O(\lambda)}$$

which is negligible in λ as desired. □

We can then use this claim to simplify the prior equations.

$$\begin{aligned}
&\left| \Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \right. \\
&\quad - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\perp) = 1 \,\middle|\, \mathsf{SHORT}_\perp \wedge \overline{\mathsf{ADV}_\perp}\right] \Pr_{\mathsf{IDEAL}}\left[\mathsf{SHORT}_\perp \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \\
&\quad \left. - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{SHORT}_\perp} \wedge \overline{\mathsf{ADV}_\perp}\right] \Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{SHORT}_\perp} \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] \right| \\
&\leq \left| \Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{SHORT}_\perp} \wedge \overline{\mathsf{ADV}_\perp}\right] \right| \qquad (5) \\
&\quad + \mathsf{negl}(\lambda)
\end{aligned}$$

Further, note that the event $\overline{\mathsf{SHORT}_\perp} \wedge \overline{\mathsf{ADV}_\perp}$ is in fact the same event as $\overline{\mathsf{SIM}_\perp}$. To bound Eq. (4) by $1/P(\lambda)$ it thus suffices to bound Eq. (5) by $1/2P(\lambda)$.

Case II: Sim **outputs unordered server inputs.** From Eq. (5), we need to bound the advantage given that Sim does not output $\perp$, i.e. $\mathsf{SIM}_\perp$ does not occur. As described previously, even in this case, the ideal functionality for F′ will output $\perp$ is Sim outputs $y = (y_1, \ldots, y_{m(\lambda)})$ such that there exists i such that $(y_{i-1}, y_i) \notin \mathsf{R}_\lambda$, i.e. the event $\mathsf{R}_\perp$ occurs. We bound the probability that this occurs.

Rewriting Eq. (5), we have

$$\begin{aligned}
&\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]\right| \\
&=\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\mathsf{R}_{\perp}\wedge\overline{\mathsf{SIM}_{\perp}}\right]\Pr_{\mathsf{IDEAL}}\left[\mathsf{R}_{\perp}\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]\right. \\
&\qquad\left.-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{R}_{\perp}}\wedge\overline{\mathsf{SIM}_{\perp}}\right]\Pr_{\mathsf{IDEAL}}\left[\overline{\mathsf{R}_{\perp}}\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]\right| \\
&\leq\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{ADV}_{\perp}}\right]-\Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda,x))=1\,\middle|\,\overline{\mathsf{R}_{\perp}}\wedge\overline{\mathsf{SIM}_{\perp}}\right]\right| \\
&\qquad+\Pr_{\mathsf{IDEAL}}\left[\mathsf{R}_{\perp}\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]
\end{aligned}\tag{6}$$

We will prove the stronger claim below, which, looking ahead, will be useful in proving indistinguishability conditioned on $\overline{\mathsf{R}_{\perp}}\wedge\overline{\mathsf{SIM}_{\perp}}$.

Claim 5. *Let* $(\mathsf{z}_{\mathsf{g},i-1},\widehat{c}_{i-1},\widehat{c}'_i,y'_{i-1},y_i,\widetilde{\mathsf{z}}_i)_{i\in[m(\lambda)]}$ *be a random sample from the list* $\mathsf{L}'_{m(\lambda)}$*. Set* $\mathsf{z}_{\mathsf{f},m(\lambda)}\coloneqq\mathsf{FHE.Dec}(\mathsf{sk},\widehat{c}'_{m(\lambda)})$*, and for each* $i\in[m(\lambda)]$*,* $\mathsf{z}_{\mathsf{f},i-1}\coloneqq\mathsf{FHE.Dec}(\mathsf{sk},\widehat{c}_{i-1})$*.*

Then,

$$\Pr_{\mathsf{IDEAL}}\left[\begin{array}{l}\mathsf{z}_{\mathsf{g},0}=\mathsf{z}'_{\mathsf{g},0}\ \wedge\ \mathsf{z}_{\mathsf{f},0}=\mathsf{z}'_{\mathsf{f},0}\ \wedge\\ \forall i\in[m(\lambda)-1],\ (y_i,y_{i+1})\in\mathsf{R}_\lambda\ \wedge\\ \forall i\in[m(\lambda)],\ \mathsf{g}_\lambda(\mathsf{z}_{\mathsf{g},i-1})=\mathsf{z}_{\mathsf{g},i}\ \wedge\\ \forall i\in[m(\lambda)],\ \widetilde{\mathsf{z}}_{i-1}=\widetilde{\mathsf{z}}_i\ \wedge\\ \forall i\in[m(\lambda)],\ \mathsf{f}_\lambda(x,y_i,\mathsf{z}_{\mathsf{f},i-1})=\mathsf{z}_{\mathsf{f},i}\end{array}\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]\geq 1-\frac{1}{8P(\lambda)}$$

where $\mathsf{z}'_{\mathsf{g},0}$ *and* $\mathsf{z}'_{\mathsf{f},0}$ *are the* true *initial states for* g *and* f *respectively.*

We defer the proof of this claim momentarily, but note that the above claim implies that $\Pr_{\mathsf{IDEAL}}\left[\mathsf{R}_{\perp}\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]\leq 1/8P(\lambda)$. And thus only need to bound Eq. (5) by $1/8P(\lambda)$ to complete our proof.

Proof of claim 5. We first rule out the boundary cases. In particular, from the somewhere extractability of BARG at index $\{1,1\}$, except with negligible probability we have that $\mathsf{z}_{\mathsf{g},0}=\mathsf{z}'_{\mathsf{g},0}$ and $\mathsf{z}_{\mathsf{f},0}=\mathsf{z}'_{\mathsf{f},0}$ where $\mathsf{z}'_{\mathsf{g},0}$ and $\mathsf{z}'_{\mathsf{f},0}$ denote the true initial states. We use here the description of the Turing machine M_λ where Sim computes $\widehat{c}_0$ as an encryption of $\mathsf{z}'_{\mathsf{f},0}$.

Then, to prove the claim it suffices to show that for each $i\in[m(\lambda)]$,

$$\Pr_{\mathsf{IDEAL}}\left[\begin{array}{l}(y_i,y_{i+1})\in\mathsf{R}_\lambda\ \wedge\\ \widetilde{\mathsf{z}}_{i-1}=\widetilde{\mathsf{z}}_i\ \wedge\\ \mathsf{g}_\lambda(\mathsf{z}_{\mathsf{g},i-1})=\mathsf{z}_{\mathsf{g},i}\ \wedge\\ \mathsf{f}_\lambda(x,y_i,\mathsf{z}_{\mathsf{f},i-1})=\mathsf{z}_{\mathsf{f},i}\end{array}\,\middle|\,\overline{\mathsf{SIM}_{\perp}}\right]\geq 1-\frac{1}{8\cdot m(\lambda)\cdot P(\lambda)}.\tag{7}$$

The claim then follows from a direct application of the union bound over all i. We note that while $y_{m\lambda+1}$ does not exists, for notation convenience, we do not separate out the case for $i=m\lambda$ in the above.

At a high level, we want to use the monotone coupling claims in Section 5 to prove the above. In the description of Sim, we use the algorithm in claim 2 to couple the lists L'_i and L_{i+1} with respect to functions f'_i and f_{i+1}.

As noted in Observation 1, that the output of the coupled algorithm from claim 2, in step $i-1$, is the list L'_i where each element in L'_{i-1} and L_i appears exactly once in L'_i. Further, f'_i is a function, where the output on $(\mathsf{z}_{\mathsf{g},j-1}, \widehat{c}_{j-1}, \widehat{c}'_j, y'_{j-1}, y_j, \widetilde{\mathsf{z}}_j)_{j\in[i]})$ only depends on $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i)$, the L_i "component" of L'_i, and we abuse notation to denote this by $\mathsf{f}'_i(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i)$. Thus, going forward, to complete the above argument, it will be sufficient to think of coupling L_i (rather than L'_i) with L_{i+1} with respect to functions f'_i and f_{i+1}.

To do so, we first define an experiment $\mathsf{EXP}_{i_0,i_1,b}$ that will be helpful in establishing Eq. (7). This experiment will correspond to sampling the BARG crs to be extractable at positions $\{i_0, i_1\}$, and then using the extracted witness w_{i_b} as follows.

$\underline{\mathsf{EXP}_{i_0,i_1,b}(\mathsf{hk}, \widehat{c})}$ *//$\widehat{c}$ and hk are passed as inputs*

1. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{i_0, i_1\})$.
2. Set $\mathsf{crs} := (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$.
3. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \widehat{c}_{m(\lambda)}, \mathsf{z}_{\mathsf{g},m(\lambda)}) \leftarrow \mathcal{A}(\mathsf{crs}, c)$.
4. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, output $\perp$.
5. Extract from the proof, $w_{i_0}, w_{i_1} := \mathsf{BARG.Extract}(\tau, \pi)$.
6. Set $w_i := w_{i_b}$. *//Choose witness for index i_b*
7. Parse w_i as $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i)$.
8. Set $y'_{i-1} = y_{i-1}$, and $\widehat{c}'_i = \widehat{c}_i$.
9. Set $\widetilde{\mathsf{z}}_i = \mathsf{z}_{\mathsf{g},m(\lambda)}$
10. Output $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i) \in \{0,1\}^{m'}$.

Figure 4: Experiment $\mathsf{EXP}_{i_0,i_1,b}$

Thus experiment $\mathsf{EXP}_{i,i,b}$ for any $b \in \{0,1\}$ corresponds to a single iteration for the i-th index, where the output element is added to L_i if the BARG proof does not verify.

Since we want to argue coupling for elements in L_i, it suffices to consider the output of $\mathsf{EXP}_{i,i,b}$ conditioned on it not outputting $\perp$. We also drop the inputs to the experiment in the description for ease of readability.

Next, we establish the following lemma.

Lemma 1. *The following two distributions are computationally indistinguishable*

$$\{\mathsf{f}'_i(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i)\}_{\mathsf{EXP}_{i,i,0}} \text{ and } \{\mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}, \widetilde{\mathsf{z}}_i)\}_{\mathsf{EXP}_{i+1,i+1,0}}$$

Proof of Lemma 1. For $\mathsf{EXP}_{i,i,0}$, by the definition of f'_i, we have

$$\mathsf{f}'_i(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i) = (\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})), \widetilde{\mathsf{z}}_i), y_i.$$

Now, we switch to the experiment $\mathsf{EXP}_{i,i+1,0}$. Here, we still use only the witness w_i, and thus since we do not need the witness w_{i+1}, we do not need to extract it. We have the following,

$$\begin{aligned}&\{(\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})), \widetilde{\mathsf{z}}_i), y_i\}_{\mathsf{EXP}_{i,i,0}}\\ &\quad\approx_c \{(\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})), \widetilde{\mathsf{z}}_i), y_i\}_{\mathsf{EXP}_{i,i+1,0}}.\end{aligned}$$

Note that from Remark 3, we have that the $\mathsf{crs}_{\mathsf{BARG}}$ in $\mathsf{EXP}_{i,i+1,0}$ consists of $\mathsf{crs}_{\mathsf{BARG}} = (\mathsf{crs}_{\mathsf{BARG},i}, \mathsf{crs}_{\mathsf{BARG},i+1})$ with trapdoor $\tau = (\tau_i, \tau_{i+1})$. As observed above, since we do not to extract witness w_{i+1}, we do not require the trapdoor τ_{i+1}. Thus, the only difference in experiments $\mathsf{EXP}_{i,i,0}$ and $\mathsf{EXP}_{i,i+1,0}$ is that the BARG crs correspond to $\mathsf{crs}_{\mathsf{BARG}} = (\mathsf{crs}_{\mathsf{BARG},i}, \mathsf{crs}'_{\mathsf{BARG},i})$ and $\mathsf{crs}_{\mathsf{BARG}} = (\mathsf{crs}_{\mathsf{BARG},i}, \mathsf{crs}_{\mathsf{BARG},i+1})$. And since we do not use the trapdoor for $\mathsf{crs}'_{\mathsf{BARG},i}$ and $\mathsf{crs}_{\mathsf{BARG},i+1}$ in their respective experiments, they can be received externally. Thus, the above computational indistinguishability follows from the index hiding property of BARG (Definition 2).

Let the witnesses extracted be,

$$\begin{aligned}w_i &= (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y'_{i-1}, \rho_{i-1}, \mathsf{z}'_{\mathsf{g},i}, \widehat{c}'_i, y_i, \rho_i)\\ w_{i+1} &= (\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y'_i, \rho_i, \mathsf{z}'_{\mathsf{g},i+1}, \widehat{c}'_{i+1}, y_{i+1}, \rho_i)\end{aligned}$$

By the somewhere extractability of BARG for M_λ (Fig. 1), and the collision resistance of HT, we have that in $\mathsf{EXP}_{i,i+1,0}$, except with negligible probability

$$\begin{aligned}\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}) &= \mathsf{z}'_{\mathsf{g},i} = \mathsf{z}_{\mathsf{g},i}\\ \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})) &= \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}'_i) = \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i)\\ y_i &= y'_i\end{aligned}$$

Therefore we have

$$\begin{aligned}&\{(\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})), \widetilde{\mathsf{z}}_i), y_i\}_{\mathsf{EXP}_{i,i+1,0}}\\ &\quad\approx_c \{(\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_i), y'_i\}_{\mathsf{EXP}_{i,i+1,1}}.\end{aligned}$$

Again, by BARG index indistinguishability we have

$$\begin{aligned}&\{(\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_i), y'_i\}_{\mathsf{EXP}_{i,i+1,1}}\\ &\quad\approx_c \{(\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_{i+1}), y'_i\}_{\mathsf{EXP}_{i+1,i+1,1}}.\end{aligned}$$

Here we simply rename the output $\mathsf{z}_{\mathsf{g},m(\lambda)}$ sent by $\mathcal{A}$ from $\widetilde{\mathsf{z}}_i$ to $\widetilde{\mathsf{z}}_{i+1}$

And finally, by somewhere extractability of BARG we have

$$\begin{aligned}&\{(\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_{i+1}), y'_i\}_{\mathsf{EXP}_{i+1,i+1,1}}\\ &\quad\approx_c \{(\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_{i+1}), y'_i\}_{\mathsf{EXP}_{i+1,i+1,0}}.\end{aligned}$$

By definition of f_{i+1}, we have

$$(\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_{i+1}), y'_i = \mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}, \widetilde{\mathsf{z}}_{i+1}).$$

This completes the proof of computational indistinguishability. □

To invoke the claims for monotone coupling on lists L_i and L_{i+1}, we need to define the relation R, which we define as follow

$$(((\mathsf{z}_{\mathsf{g},0}, \mathsf{z}_{\mathsf{f},0}, \widetilde{\mathsf{z}}_0), y_0), ((\mathsf{z}_{\mathsf{g},1}, \mathsf{z}_{\mathsf{f},1}, \widetilde{\mathsf{z}}_0), y_1)) \in R \Leftrightarrow (\mathsf{z}_{\mathsf{g},0}, \mathsf{z}_{\mathsf{f},0}, \widetilde{\mathsf{z}}_0) = (\mathsf{z}_{\mathsf{g},1}, \mathsf{z}_{\mathsf{f},1}, \widetilde{\mathsf{z}}_1) \ \wedge \ y_0 \leq y_1.$$

This is indeed the relation R for which claim 3 is proven. By our choice of $\kappa(\lambda)$ set to be the polynomial in claim 3 for $P'(\lambda) = 1/(8 \cdot m(\lambda) \cdot P(\lambda))$, the lists L_i and L_{i+1} are $\kappa(\lambda)$ sized lists constructed by sampling elements from $\mathsf{EXP}_{i,i,0}$ and $\mathsf{EXP}_{i+1,i+1,0}$ respectively such that

$$\{\mathsf{f}'_i(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i), \widetilde{\mathsf{z}}_i\}_{\mathsf{EXP}_{i,i,0}} \text{ and } \{\mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}, \widetilde{\mathsf{z}}_{i+1})\}_{\mathsf{EXP}_{i+1,i+1,0}}.$$

Since we have met all the conditions for claim 3, by setting $\tilde{X}$ and $\tilde{Y}$ to be the uniform sampling over L_i and L_{i+1} respectively, we can bound the deficiency by

$$\Pr\left[\Delta_R(\mathsf{f}'_i(\tilde{X}), \mathsf{f}_{i+1}(\tilde{Y})) \leq \frac{1}{8 \cdot m(\lambda) \cdot P(\lambda)} \middle| \overline{\mathsf{SIM}_\perp}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Now by claim 2, the output of the algorithm is a *coupled* list $\mathsf{L}_{i,i+1}$ (corresponding to L'_{i+1} in Sim) such that when sampling an element $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i), (\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_i, y'_{i+1}, y_{i+1}, \widetilde{\mathsf{z}}_{i+1})$ uniformly at random from $\mathsf{L}_{i,i+1}$ it follows that the marginals of $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i)$ is identical to sampling uniformly from L_i (and same for L_{i+1}) and

$$\Pr\left[(\mathsf{f}'_i(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i), \mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}), \widetilde{\mathsf{z}}_{i+1}) \in R \middle| \overline{\mathsf{SIM}_\perp}\right] \geq 1 - \frac{1}{8 \cdot m(\lambda) \cdot P(\lambda)}.$$

From the definitions of R, f'_i and f_{i+1} we have

$$\Pr_{\mathsf{IDEAL}}\left[\begin{array}{l} y_i \leq y'_i \wedge \\ \widetilde{\mathsf{z}}_{i-1} = \widetilde{\mathsf{z}}_i \wedge \\ \mathsf{g}_\lambda(\mathsf{z}_{\mathsf{g},i-1}) = \mathsf{z}_{\mathsf{g},i} \wedge \\ \mathsf{f}_\lambda(x, y_i, \mathsf{z}_{\mathsf{f},i-1}) = \mathsf{z}_{\mathsf{f},i} \end{array} \middle| \overline{\mathsf{SIM}_\perp}\right] \geq 1 - \frac{1}{8 \cdot m(\lambda) \cdot P(\lambda)}.$$

Notice that the only difference from Eq. (7) is that we have the guarantee $y_i \leq y'_i$ instead of $(y_i, y_{i+1}) \in \mathsf{R}_\lambda$. However, from the somewhere extractability of BARG, we have that except with negligible probability $(y'_i, y_{i+1}) \in \mathsf{R}_\lambda$. And thus by the transitivity of the ordering relation R_λ we have that

$$\Pr_{\mathsf{IDEAL}}\left[\begin{array}{l} (y_i, y_{i+1}) \in \mathsf{R}_\lambda \wedge \\ \widetilde{\mathsf{z}}_{i-1} = \widetilde{\mathsf{z}}_i \wedge \\ \mathsf{g}_\lambda(\mathsf{z}_{\mathsf{g},i-1}) = \mathsf{z}_{\mathsf{g},i} \wedge \\ \mathsf{f}_\lambda(x, y_i, \mathsf{z}_{\mathsf{f},i-1}) = \mathsf{z}_{\mathsf{f},i} \end{array} \middle| \overline{\mathsf{SIM}_\perp}\right] \geq 1 - \frac{1}{8 \cdot m(\lambda) \cdot P(\lambda)}.$$

This completes the proof. □

Case III: Ideal Experiment computes F. Conditioned on $\overline{\mathsf{R}_\perp} \wedge \overline{\mathsf{SIM}_\perp}$, we note that due to $\overline{\mathsf{SIM}_\perp}$, Sim outputs $y \neq \perp$, and thus the ideal experiment computes F' using y. Further, since $\overline{\mathsf{R}_\perp}$ occurs as well, the output of F' is the output of F. We now bound

$$\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \middle| \overline{\mathsf{ADV}_\perp}\right] - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \middle| \overline{\mathsf{R}_\perp} \wedge \overline{\mathsf{SIM}_\perp}\right]\right|.$$

Let $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \mathsf{z}_{\mathsf{g},i})_{i\in[m(\lambda)]}$ be a random sample from the list $\mathsf{L}'_{m(\lambda)}$ that Sim uses to output $y = (y_1, \ldots, y_{m(\lambda)})$. By claim 5, we know that, except with probability $1/8P(\lambda)$, $\forall i \in [m(\lambda)]$, $\mathsf{f}_\lambda(x, y_i, \mathsf{z}_{\mathsf{f},i-1}) = \mathsf{z}_{\mathsf{f},i}$ where $\mathsf{z}_{\mathsf{f},m(\lambda)} \coloneqq \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}'_{m(\lambda)})$, and for each $i \in [m(\lambda)]$, $\mathsf{z}_{\mathsf{f},i-1} \coloneqq \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})$. Thus, by correctness of $\mathsf{F} = \{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$, computable by bounded state function $\{\mathsf{f}_\lambda\}_{\lambda\in\mathbb{N}}$, we have that the ideal experiment output $z = \mathsf{F}_\lambda(x, y)$ is indeed equal to $\mathsf{z}_{\mathsf{f},m(\lambda)}$.

Indistinguishability of the real and ideal output. We are finally left with arguing that the output of real and ideal experiment conditioned on the events not occurring thus far, are only $1/4P(\lambda)$ apart.

As stated above, $\mathsf{z}_{\mathsf{f},m(\lambda)}$, the output of the ideal experiment, is computed by decrypting $\widehat{c}'_{m(\lambda)}$. Further, $\widehat{c}'_{m(\lambda)}$ is added to the list by extracting the witness $w_{m(\lambda)}$. By the description of the Turing machine M_λ (Fig. 1), and by the somewhere extractability of BARG (Definition 2), we have that except with negligible probability, $\mathsf{z}_{\mathsf{f},m(\lambda)} = \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$ where $\widehat{c}_{m(\lambda)}$ is the ciphertext sent by $\mathcal{A}$.

Correspondingly, in the real experiment, conditioned on $\overline{\mathsf{ADV}_\perp}$, the output of the experiment is $\mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$ where $\widehat{c}_{m(\lambda)}$ is the ciphertext sent by $\mathcal{A}$ in the real experiment.

Thus, except for probability $1/8P(\lambda) + \mathsf{negl}(\lambda)$, the output of the real and ideal experiments are from identical distributions. We have,

$$\left|\Pr_{\mathsf{REAL}}\left[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{ADV}_\perp}\right] - \Pr_{\mathsf{IDEAL}}\left[\mathcal{D}(\mathsf{IDEAL}_{\mathcal{A}}(\lambda, x)) = 1 \,\middle|\, \overline{\mathsf{R}_\perp} \wedge \overline{\mathsf{SIM}_\perp}\right]\right| \leq \frac{1}{8P(\lambda)} + \mathsf{negl}(\lambda).$$

Giving us the necessary bound and completing the proof. □

6.1.3 Server-input Indistinguishability with respect to G′.

Fix an adversary $\mathcal{A}$, and polynomial $P(\lambda)$. Sim is defined as in Fig. 3. Further, we fix distinguisher $\mathcal{D}$ and index i. For the server-input indistinguishability with respect to G′, it suffices to show

$$\left|\Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}}(i, \lambda, x_0)) = 1\right] - \Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}}(i, \lambda, x_1)) = 1\right]\right| \leq \frac{1}{P(\lambda)}, \tag{8}$$

where

$$\begin{array}{l}\underline{\mathsf{IDEAL}'_{\mathsf{Sim}}(i, \lambda, x)} \\ y \leftarrow \mathsf{Sim}(1^\lambda, x) \\ \text{If } y = \perp, \text{ set } (y_i, z) \leftarrow (\perp, \perp) \\ \text{If } y \neq \perp, \text{ set } z \leftarrow \mathsf{G}'_\lambda(y) \\ \text{Output } (y_i, z)\end{array}$$

Note that except for running Sim using x, IDEAL′ does not use x to compute its output. Further, note that for G′ defined in Theorem 5, we can rewrite IDEAL′ as

$\underline{\mathsf{IDEAL}'_{\mathsf{Sim}}(i,\lambda,x)}$
$y \leftarrow \mathsf{Sim}(1^\lambda, x)$
If $y = \bot$, set $(y_i, z) \leftarrow (\bot, \bot)$
If $y \neq \bot$, set $z \leftarrow \bot$ if $\exists i$ s.t.$(y_i, y_{i+1}) \notin \mathsf{R}_\lambda$
If $z \neq \bot$, set $z \leftarrow \mathsf{G}_\lambda(y)$
Output (y_i, z)

We now prove Eq. (8) via a sequence of steps. First, we will use a variation of claim 5 which will focus only on the states that are relevant to IDEAL′, i.e. on the states required to compute G. In particular, we have the following modified claim,

Claim 6. *Let* $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \mathsf{z}_{\mathsf{g},i})_{i\in[m(\lambda)]}$ *be a random sample from the list* $\mathsf{L}'_{m(\lambda)}$. *Set* $\mathsf{z}_{\mathsf{f},m(\lambda)} \coloneqq \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}'_{m(\lambda)})$, *and for each* $i \in [m(\lambda)]$, $\mathsf{z}_{\mathsf{f},i-1} \coloneqq \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})$.

Then,

$$\Pr_{\mathsf{IDEAL}}\left[\begin{array}{l} \mathsf{z}_{\mathsf{g},0} = \mathsf{z}'_{\mathsf{g},0} \wedge \\ \forall i \in [m(\lambda)-1],\ (y_i, y_{i+1}) \in \mathsf{R}_\lambda \wedge \\ \forall i \in [m(\lambda)],\ \widetilde{\mathsf{z}}_{i-1} = \widetilde{\mathsf{z}}_i \wedge \\ \forall i \in [m(\lambda)],\ \mathsf{g}_\lambda(\mathsf{z}_{\mathsf{g},i-1}) = \mathsf{z}_{\mathsf{g},i} \wedge \end{array} \middle| \overline{\mathsf{SIM}_\bot} \right] \geq 1 - \frac{1}{8P(\lambda)}$$

where $\mathsf{z}'_{\mathsf{g},0}$ *is the* true *initial state for* g.

The claim directly follows from claim 5. Thus, when it is in fact that case that $\overline{\mathsf{SIM}_\bot}$ occurs and for all i $(y_i, y_{i+1}) \in \mathsf{R}_\lambda$ and $\mathsf{g}_\lambda(\mathsf{z}_{\mathsf{g},i-1}) = \mathsf{z}_{\mathsf{g},i}$, by the correctness of G the output of IDEAL′ is indeed $(y_i, \mathsf{z}_{\mathsf{g},m(\lambda)})$. Further, since for all i, (i) $\widetilde{\mathsf{z}}_{i-1} = \widetilde{\mathsf{z}}_i$; and (ii) $\widetilde{\mathsf{z}}_i = \mathsf{z}_{\mathsf{g},m(\lambda)}$); the output of IDEAL′ is $(y_i, \widetilde{\mathsf{z}}_i)$.

Simulation without coupling on f**.** Since the ideal functionality no longer computes F, it is not necessary for Sim to couple on f. We thus define a new intermediate simulator Sim′ that does not couple on f, but also importantly *does not* use the FHE secret key for its coupling.

$\underline{\mathsf{Sim}'(1^\lambda, x)}$

1. Set $\kappa(\lambda)$ as the polynomial claim 3 for $P'(\lambda) = 1/(8 \cdot m(\lambda) \cdot P(\lambda))$
2. Sample $\mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$.
3. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{m(\lambda), m(\lambda)\})$
4. Sample $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.
5. Compute ciphertexts $\widehat{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$, $\widehat{c}_{\mathsf{init}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, z_0)$ and set $c \coloneqq (\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}})$.
6. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \widehat{c}_{m(\lambda)}, \mathsf{z}_{\mathsf{g},m(\lambda)}) \leftarrow \mathcal{A}(\mathsf{crs}, c)$
7. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, output $\bot$.
8. For $i \in [m(\lambda)]$
   (a) Set the list L_i to be the empty list.

(b) Repeat the following $\kappa(\lambda) \cdot \lambda \cdot P(\lambda)$ times

i. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{i, i\})$.

ii. Set $\mathsf{crs} := (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$.

iii. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \widehat{c}_n) \leftarrow \mathcal{A}(\mathsf{crs}, c)$.

iv. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, return to Item 8b.

v. Extract from the proof, $w_i := \mathsf{BARG.Extract}(\tau, \pi)$.

vi. Parse w_i as $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i)$.

vii. Set $y'_{i-1} = y_{i-1}$, and $\widehat{c}'_i = \widehat{c}_i$.

viii. Set $\widetilde{\mathsf{z}}_i = \mathsf{z}_{\mathsf{g},m(\lambda)}$.

ix. Add $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i) \in \{0,1\}^{m'}$ to L_i.

(c) If $|\mathsf{L}_i| < \kappa(\lambda)$ output $\perp$. Otherwise, truncate its length to exactly $\kappa(\lambda)$.

9. Set $\mathsf{L}'_1 = \mathsf{L}_1 \in \{0,1\}^{m' \cdot \kappa(\lambda)}$.

10. For every $i \in [m(\lambda) - 1]$

(a) Let $\mathsf{f}'_i : \{0,1\}^{m' \cdot i} \to \{0,1\}^{2 \cdot s(\lambda) + \ell(\lambda)}$ and $\mathsf{f}_{i+1} : \{0,1\}^{m'} \to \{0,1\}^{2 \cdot s(\lambda) + \ell(\lambda)}$ be functions defined as follows, where x is hardcoded in the function:

$$\mathsf{f}'_i((\mathsf{z}_{\mathsf{g},j-1}, \widehat{c}_{j-1}, \widehat{c}'_j, y'_{j-1}, y_j, \widetilde{\mathsf{z}}_j)_{j \in [i]}) = (\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \widetilde{\mathsf{z}}_i), y_i$$
$$\mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}, \widetilde{\mathsf{z}}_{i+1}) = (\mathsf{z}_{\mathsf{g},i}, \widetilde{\mathsf{z}}_{i+1}), y'_i$$

(b) Compute $\mathsf{L}'_{i+1} \in \{0,1\}^{m' \cdot (i+1) \cdot \kappa(\lambda)}$, the coupling of L'_i and L_{i+1} using the algorithm in claim 2.

11. Sample uniformly from the list $\mathsf{L}'_{m(\lambda)}$, i.e. $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i)_{i \in [m(\lambda)]} \leftarrow \mathsf{L}'_{m(\lambda)}$.

12. Output $y = (y_1, \ldots, y_{m(\lambda)})$.

We can now bound the change in $\mathcal{D}$'s output when we switch the IDEAL′ from using Sim to Sim′

$$\left| \Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}}(i, \lambda, x_0)) = 1\right] - \Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}'}(i, \lambda, x_0)) = 1\right] \right| \leq 1/8P(\lambda). \tag{9}$$

We first observe that claim 6 holds with respect to Sim′ as well. This follows in a manner identical to the proof of claim 5, where the functions in Sim′ are different than those in Sim, but the coupling argument still holds as we are only coupling on values that do not require knowledge of the FHE key. Further, note that the changes made to Sim are only in the construction of L', while the lists L have an identical distribution to Sim.

Thus: (i) The event $\mathsf{SIM}_\perp$ happen with identical probability for Sim and Sim′; (ii) conditioned on $\overline{\mathsf{SIM}_\perp}$, except with probability $1/8P(\lambda)$, the output of IDEAL′ has identical distributions - this follows from the output in both cases being $(y_i, , \widetilde{\mathsf{z}}_i)$ where the marginal distributions are identical. Eq. (9) then follows.

Switching inputs to Sim'**.** We now switch the input from x_0 to x_1. We will use the security of the FHE scheme (Definition 4) to argue the following

$$\left|\Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}'}(i,\lambda,x_0))=1\right]-\Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}'}(i,\lambda,x_1))=1\right]\right|\leq \mathsf{negl}(\lambda). \tag{10}$$

Recall from our description that Sim' only uses the input x to compute the FHE ciphertext $\widehat{c}$, and thereafter does not require knowledge of the FHE secret key sk. Therefore, Sim' can be emulated given only $\widehat{c}$, rather than x.

Thus, we can now prove Eq. (10) by constructing an adversary $\mathcal{B}$ for the (semantic) security of FHE. In the reduction, the challenger provides $\mathcal{B}$ a ciphertext $\widehat{c}$ to either x_0 or x_1. As argued above, $\mathcal{B}$ can: (i) emulate Sim' using only $\widehat{c}$; (ii) Compute the output of IDEAL' given the output of Sim'; and (iii) run $\mathcal{D}$ on the output of IDEAL' and return as guess the output of $\mathcal{D}$.

It is easy to see that depending on the ciphertext provided to $\mathcal{B}$, the input to $\mathcal{D}$ corresponds exactly to $\mathsf{IDEAL}'_{\mathsf{Sim}'}(i,\lambda,x_0)$ or $\mathsf{IDEAL}'_{\mathsf{Sim}'}(i,\lambda,x_1)$. Thus, if Eq. (10) is not negligible, $\mathcal{B}$ breaks FHE security.

Finally, note that we can switch Sim' back to Sim in a manner identical to Eq. (9), we get

$$\left|\Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}'}(i,\lambda,x_1))=1\right]-\Pr\left[\mathcal{D}(\mathsf{IDEAL}'_{\mathsf{Sim}}(i,\lambda,x_1))=1\right]\right|\leq 1/8P(\lambda).$$

Putting it together. Now, by putting together Eqs. (9) and (10) and Section 6.1.3, we have the required Eq. (8).

6.1.4 Server Security.

Thus far, we have only described client security, against a malicious server. We now show how we can add server security generically, using a non-interactive zero-knowledge against a malicious client to any protocol that is of a specific structure, that we call *admissible*.

Definition 12 (Admissible One-round Two-party secure computation protocol)**.** *A one-round two-party secure computation protocol* $\Pi = (\mathsf{Gen}, \mathsf{Client}, \mathsf{Server}, \mathsf{Out})$ *is said to be admissible if the protocol has the additional following structure: (i) client state* $\mathsf{st} = (c, \mathsf{st}')$ *includes the client message* c *and secret state* st'*; and (ii) server message* s *is of the form* $s = (s_1, s_2)$*. Further* $\mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)$ *be decomposed into two steps: (i) checking* $\mathsf{Verify}(\mathsf{crs}, c, s_1, s_2) = 1$*, a verification procedure that* does not *depend on the secret state* st'*; and (ii) an output computation step* $\mathsf{Output}(\mathsf{crs}, s_2, \mathsf{st})$ *that* does not *depend on* s_1*, and simply decrypts* s_2 *using* st.

Claim 7. *The protocol in Fig. 2 is admissible.*

Proof sketch. This follows directly by observing that $\widehat{c}_{m(\lambda)}$ is s_2, and everything else sent by the server, including the BARG proof is s_1. The secret state is simply the secret key sk. The verification procedure is the BARG verification, which only requires the client message c from st, and Output is the decryption of s_2 which requires the secret state st'. □

We now prove the following theorem.

Theorem 7. *Assuming NIZK and any admissible one-round two-party secure computation protocol* $\Pi = (\mathsf{Gen}, \mathsf{Client}, \mathsf{Server}, \mathsf{Out})$ *satisfying Definition 8, there is a protocol* $\Pi' = (\mathsf{Gen}', \mathsf{Client}', \mathsf{Server}', \mathsf{Out}')$ *in the CRS model satisfying Definition 8 and server-security (Definition 5). Further, for* Π *if the client's message is of size* $n(\lambda) \cdot \mathsf{poly}(\lambda)$ *and the server's message is of size* $\ell(\lambda) \cdot \mathsf{poly}(\lambda)$*, this is preserved in* Π'*.*

Proof. We prove the above theorem by presenting the transformation. We will use $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract})$ for two languages. The first language, $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ is parameterized by crs, and is defined as

$$\mathcal{L}_\lambda = \{c \mid \exists (x, r) \text{ s.t } c = \mathsf{Client}(\mathsf{crs}, x; r)\}.$$

It is used to establish that the client message c is computed honestly.

The second language $\mathcal{L}' = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ is parameterized by crs, and is defined as

$$\mathcal{L}'_\lambda = \{(c, s_2) \mid \exists s_1 \text{ s.t } \mathsf{Verify}(\mathsf{crs}, c, s_1, s_2) = 1\}.$$

We will find it convenient to define two separate NIZKs, one for each language, where the language is implicit for each.

$\underline{\mathsf{Gen}'(1^\lambda)}$

1. $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$.
2. $(\mathsf{crs}_{\mathsf{NIZK},\mathsf{Client}}, \tau_1) \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$. //For $\mathcal{L}$
3. $(\mathsf{crs}_{\mathsf{NIZK},\mathsf{Server}}, \tau_2) \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$. //For $\mathcal{L}'$
4. Output $\mathsf{crs}' := (\mathsf{crs}, \mathsf{crs}_{\mathsf{NIZK},\mathsf{Client}}, \mathsf{crs}_{\mathsf{NIZK},\mathsf{Server}})$.

$\underline{\mathsf{Client}'(\mathsf{crs}', x)}$

1. $(c, \mathsf{st}) := \mathsf{Client}(\mathsf{crs}, x; r)$.
2. $\pi_{\mathsf{Client}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK},\mathsf{Client}}, c, (x, r))$.
3. $c' := (c, \pi_{\mathsf{Client}})$.
4. Output (c', st)

$\underline{\mathsf{Server}'(\mathsf{crs}', y, c')}$

1. $\mathsf{Verify\ NIZK.Verify}(\mathsf{crsNIZK}, \mathsf{Client}, c, \pi_{\mathsf{Client}}) = 1$.
2. $(s_1, s_2) \leftarrow \mathsf{Server}(\mathsf{crs}, y, c)$.
3. $\pi_{\mathsf{Server}} \leftarrow \mathsf{Prove}(\mathsf{crs}_{\mathsf{NIZK},\mathsf{Server}}, (c, s_2), s_1)$.
4. Output $s' := (s_2, \pi_{\mathsf{Server}})$.

$\underline{\mathsf{Out}'(\mathsf{crs}', \mathsf{st}, s')}$

1. $\mathsf{Verify\ NIZK.Verify}(\mathsf{crsNIZK}, \mathsf{Server}, s_2, \pi_{\mathsf{Server}}) = 1$.
2. Output $z = \mathsf{Output}(\mathsf{crs}, (c, s_2), \mathsf{st})$.

Figure 5: Transformation to Π' for server security

We now prove that the protocol Π' satisfies the requisite properties. Since this is fairly straightforward, we only provide a sketch of the proof for each.

First, we note that the transformation preserves efficiency.

Claim 8. *The efficiency of* Π *is preserved in* Π'.

Proof sketch. The client message is of size $|c'| = |c| + |\pi_{\mathsf{Client}}| = n(\lambda) \cdot \mathsf{poly}(\lambda)$ by the efficiency of the NIZK. The server message is of size $|s'| = |s_2| + |\pi_{\mathsf{Server}}| = O(\ell) \cdot \mathsf{poly}(\lambda)$ by the efficiency of the NIZK. □

Next, we observe that the input-dependent simulation (for the client) as in Definition 8 is preserved. For this is suffices to show that any adversary $\mathcal{A}'$ for Π' can be transformed to an adversary for Π.

Claim 9. *Assuming the security of the NIZK, for every adversary* $\mathcal{A}'$ *corrupting the server in* Π'*, there is an adversary* $\mathcal{A}$ *for* Π.

Proof sketch. $\mathcal{A}$ generates $\mathsf{crs}_{\mathsf{NIZK,Client}}$ and $\mathsf{crs}_{\mathsf{NIZK,Server}}$ as specified. Given a client message c for Π, $\mathcal{A}$ simulates π_{Client} using the trapdoor τ_1 (for $\mathsf{crs}_{\mathsf{NIZK,Client}}$) and hands it to $\mathcal{A}'$. On receiving $(s_2, \pi_{\mathsf{Server}})$ from $\mathcal{A}'$, $\mathcal{A}$ uses the trapdoor τ_2 (for $\mathsf{crs}_{\mathsf{NIZK,Server}}$) to extract the witness w, which by the extraction correctness is s_1. $\mathcal{A}$ then outputs $s = (s_1, s_2)$. □

Claim 10. *The protocol* Π' *satisfies server security as in Definition 5.*

Proof sketch. Here it suffices to show a simulator Sim. Sim generates $\mathsf{crs}_{\mathsf{NIZK,Client}}$ and $\mathsf{crs}_{\mathsf{NIZK,Server}}$ as specified. Given a client message c' where π_{Client} verifies, Sim extracts (x, r) from π_{Client} using the trapdoor τ_1 (for $\mathsf{crs}_{\mathsf{NIZK,Client}}$). It sends x to the functionality, that returns z. Sim then computes s_2 as an encryption of z, and simulates the proof π_{Server} using τ_2 (for $\mathsf{crs}_{\mathsf{NIZK,Server}}$). □

□

6.1.5 Client Input Privacy.

We show that both the protocol Π in Fig. 2 and consequently Π' satisfy client input privacy (Definition 6).

Claim 11. *Protocol* Π *in Fig. 2 satisfies client input privacy (Definition 6).*

Proof sketch. Since the only change in the client message c from input x_0 to x_1 is FHE ciphertext $\widehat{c}$, the client input privacy follows directly from the semantic security of the FHE scheme (Definition 4). □

Claim 12. *For every* Π *that satisfies client input privacy, the protocol* Π' *in Fig. 5 also satisfies client input privacy.*

Proof sketch. The client message in Π' now includes the client message c of the underlying protocol, and a proof π_{Client}. The client input privacy of Π' follows from the client input privacy of Π and the zero knowledge property of the NIZK. In particular, this is shown by considering a sequence of hybrid experiments where we slowly replace the client message c' on input x_0 to x_1. In the first hybrid, we replace π_{Client} with a simulated proof - this is done relying on the zero knowledge

property of the NIZK. Next, we change the client message c on input x_0 to x_1, relying here on the client input privacy of Π, and finally we switch the simulated proof with an honestly computed proof using input x_1. □

6.2 Sorted lookup functions

6.2.1 Protocol.

The protocol builds on the one in Section 6.1.1, with an additional building block: a somewhere-extractable hash family

$$\mathsf{SEH} = (\mathsf{SEH.Gen}, \mathsf{SEH.TGen}, \mathsf{SEH.Hash}, \mathsf{SEH.Open}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract}) .$$

Specifically, we use the SEH scheme of [HW15] based on FHE, which has certain additional properties (described in detail in the full version). Namely, we use the fact that the hash key contains an encryption of an index under the FHE scheme, and extraction correctness holds for any ciphertext encrypting i (not only for a fresh ciphertext).

Overview. The protocol is identical to Section 6.1.1, except that we use the SEH scheme to perform the final indexing operation. The server evaluates the FHE ciphertext containing the output of Search, and uses it to construct a hash key for the SEH scheme, programmed on that index. It then computes the SEH of its input y using the evaluated key, and augments the BARG proof to also give openings of its input with respect to this hash. The client then extracts the value from the hash as its final output.

We now describe the protocol, highlighting the changes from Section 6.1.1.

Denote by f_λ, g_λ the state-computing functions for $\mathsf{Search}_\lambda, \mathsf{G}_\lambda$ respectively (as in Definition 11).

Turing machine for BARGs. We next describe the family of Turing machine $\mathsf{M} = \{\mathsf{M}_\lambda\}_{\lambda\in\mathbb{N}}$ that the server will need to provide a BARG proof for. The machine M_λ hardcodes the hash key hk, tree root rt, SEH key hk^*, SEH root rt^*, and encrypted start state $\widehat{c}_0$.

$\mathsf{M}_\lambda[\mathsf{hk}, \mathsf{rt}, \mathsf{hk}^*, \mathsf{rt}^*, \mathsf{z}_{\mathsf{g},\mathsf{init}}, \mathsf{z}_{\mathsf{g},\mathsf{final}}, \widehat{c}, \widehat{c}_{\mathsf{init}}, \widehat{c}_{\mathsf{final}}](i, w_i)$

1. Parse $w_i = (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i, \rho_i^*)$.
2. if $i = 1$, check $\widehat{c}_0 = \widehat{c}_{\mathsf{init}}$ and $\mathsf{z}_{\mathsf{g},0} = \mathsf{z}_{\mathsf{g},\mathsf{init}}$.
3. if $i = m(\lambda)$, check $\widehat{c}_{m(\lambda)} = \widehat{c}_{\mathsf{final}}$ and $\mathsf{z}_{\mathsf{g},m(\lambda)} = \mathsf{z}_{\mathsf{g},\mathsf{final}}$.
4. Check $\widehat{c}_i = \mathsf{FHE.Eval}(\mathsf{f}_\lambda, \widehat{c}, y_i, \widehat{c}_{i-1})$.
5. Check $\mathsf{z}_{\mathsf{g},i} = \mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1})$.
6. Check $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{rt}, i, (\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i), \rho_i) = 1$.
7. if $i > 1$, check $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{rt}, i-1, (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}), \rho_{i-1}) = 1$ and $(y_{i-1}, y_i) \in \mathsf{R}_\lambda$.
8. Check $\mathsf{SEH.Verify}(\mathsf{hk}^*, \mathsf{rt}^*, m(\lambda), i, \mathsf{Project}(y_i), \rho_i^*) = 1$.
9. Accept if all the checks verify.

Figure 6: Turing Machine M_λ

Note that we can set t, the running time of M, to be the time it takes to perform the above checks, which is polynomial in its inputs.

$\underline{\mathsf{Gen}(1^\lambda)}$

1. $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{m(\lambda), m(\lambda)\})$.
2. $\mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$.
3. Output $\mathsf{crs} := (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$.

$\underline{\mathsf{Client}(\mathsf{crs}, x)}$

1. $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.
2. $(\mathsf{hk}^*, \mathsf{td}^*) \leftarrow \mathsf{SEH.Gen}(1^\lambda, (\mathsf{sk}, \mathsf{pk}))$.
3. $\widehat{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$.
4. $\widehat{c}_{\mathsf{init}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mathsf{z}_{\mathsf{f},0})$.
5. $\mathsf{st} := (\mathsf{sk}, \mathsf{hk}^*, \mathsf{td}^*)$.
6. $c := (\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}}, \mathsf{hk}^*)$.
7. Output (c, st)

$\underline{\mathsf{Server}(\mathsf{crs}, y = (y_1, \ldots, y_{m(\lambda)}), c)}$

1. Parse c as $(\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}}, \mathsf{hk}^*)$, and crs as $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$.
2. Set $\widehat{c}_0 := \widehat{c}_{\mathsf{init}}$ and $\mathsf{z}_{\mathsf{g},0} := \mathsf{z}_{\mathsf{g},\mathsf{init}}$
3. For each $i \in [m(\lambda)]$, compute $\widehat{c}_i := \mathsf{FHE.Eval}(\mathsf{f}_\lambda, \widehat{c}, y_i, \widehat{c}_{i-1})$.
4. For each $i \in [m(\lambda)]$, compute $\mathsf{z}_{\mathsf{g},i} := \mathsf{g}(y_i, \mathsf{z}_{\mathsf{g},i-1})$.
5. Set $\mathsf{z}_{\mathsf{g},\mathsf{final}} = \mathsf{z}_{\mathsf{g},m(\lambda)}$ and $\widehat{c}_{\mathsf{final}} = \widehat{c}_{m(\lambda)}$.
6. Compute the hash tree root $\mathsf{rt} = \mathsf{HT.Hash}(\mathsf{hk}, ((\mathsf{z}_{\mathsf{g},1}, \widehat{c}_1, y_1) \cdots, (\mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}, y_{m(\lambda)})))$.
7. Replace the ciphertext in hk^* with $\widehat{c}_{m(\lambda)}$.
8. Let $\mathsf{Project}(y) = (\mathsf{Project}(y_1), \ldots, \mathsf{Project}(y_{m(\lambda)}))$.
9. Compute the SEH root $\mathsf{rt}^* = \mathsf{SEH.Hash}(\mathsf{hk}^*, \mathsf{Project}(y))$.
10. Use hk, rt,hk^*, rt^*, $\mathsf{z}_{\mathsf{g},\mathsf{init}}$, $\mathsf{z}_{\mathsf{g},\mathsf{final}}$, $\widehat{c}$, $\widehat{c}_{\mathsf{init}}$ and $\widehat{c}_{\mathsf{final}}$ to define the Turing machine M_λ as in figure 6.
11. For every $i \in [m(\lambda)]$,
    - $\rho_i = \mathsf{HT.Open}(\mathsf{hk}, ((\mathsf{z}_{\mathsf{g},1}, \widehat{c}_1, y_1) \cdots, (\mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}, y_{m(\lambda)})), i)$.

– $\rho_i^* = \mathsf{SEH.Open}(\mathsf{hk}^*, \mathsf{Project}(y), i)$.

– Construct witness (i, w_i) for with $w_i := (\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i, \rho_i^*)$.

12. Compute the BARG proof $\pi \leftarrow \mathsf{BARG.Prove}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, (w_1, \ldots, w_{m(\lambda)}))$.

13. Output $s := (\pi, \mathsf{rt}, \mathsf{rt}^*, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)})$.

$\underline{\mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)}$

1. Parse s as $(\pi, \mathsf{rt}, \mathsf{rt}^*, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)})$, crs as $(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$ and st as $(\mathsf{sk}, \mathsf{hk}^*, \mathsf{td}^*)$.

2. Set $\mathsf{z}_{\mathsf{g,final}} = \mathsf{z}_{\mathsf{g},m(\lambda)}$ and $\widehat{c}_{\mathsf{final}} = \widehat{c}_{m(\lambda)}$.

3. Replace the ciphertext in hk^* with $\widehat{c}_{m(\lambda)}$.

4. Use hk, rt, hk^*, rt^*, $\mathsf{z}_{\mathsf{g,init}}$, $\mathsf{z}_{\mathsf{g,final}}$, $\widehat{c}$, $\widehat{c}_{\mathsf{init}}$ and $\widehat{c}_{\mathsf{final}}$ to define the Turing machine M_λ as in figure 6.

5. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 1$, set $z = \mathsf{SEH.Extract}(\mathsf{td}^*, \mathsf{rt}^*)$. Else set $z = \perp$.

6. Output z

Figure 7: Protocol for $\mathsf{F}' = \{\mathsf{F}'_\lambda\}$.

6.2.2 Analysis.

We now show that the protocol described in Fig. 7 satisfies input-dependent simulation security Definition 8 for F. In fact we show that it satisfies the stronger notion of server-indistinguishability with respect to G′ (Definition 9).

Remark 8. *We note that upgrading to server security (Definition 5) and client input privacy (Definition 6) follows in an identical manner to Theorem 5, and we skip these here for the sake of readability. For server security, the server message is split to $s = (s_1, s_2)$ where $s_2 = \mathsf{rt}^*$.*

The proof builds on the proof of Theorem 5 in Section 6.1.1. The simulator is defined identically to Fig. 3, except that we additionally couple on $\mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$. Written in full:

$\underline{\mathsf{Sim}(1^\lambda, x)}$

1. Set $\kappa(\lambda)$ as the polynomial claim 3 for $P'(\lambda) = 1/(8 \cdot m(\lambda) \cdot P(\lambda))$.

2. Sample $\mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$.

3. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{m(\lambda), m(\lambda)\})$

4. Sample $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.

5. Sample $(\mathsf{hk}^*, \mathsf{td}^*) \leftarrow \mathsf{SEH.Gen}(1^\lambda, (\mathsf{sk}, \mathsf{pk}))$.

6. Compute ciphertexts $\widehat{c} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$, $\widehat{c}_{\mathsf{init}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, z_0)$ and set $c := (\mathsf{pk}, \widehat{c}, \widehat{c}_{\mathsf{init}}, \mathsf{hk}^*)$.

7. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \mathsf{rt}^*, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}) \leftarrow \mathcal{A}(\mathsf{crs}, c)$.
8. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, output $\perp$. //If $\mathcal{A}$ causes abort sufficiently often, can simply output $\perp$
9. For $i \in [m(\lambda)]$
   (a) Set the list L_i to be the empty list.
   (b) Repeat the following $\kappa(\lambda) \cdot \lambda \cdot P(\lambda)$ times
      i. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{i, i\})$.
      ii. Set $\mathsf{crs} := (\mathsf{crs}_{\mathsf{BARG}}, \mathsf{hk})$. //Reuse the previously sampled hk
      iii. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \mathsf{rt}^*, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}) \leftarrow \mathcal{A}(\mathsf{crs}, c)$. //Reuse previously computed ciphertexts
      iv. If $\mathsf{BARG.Verify}(\mathsf{crs}_{\mathsf{BARG}}, \mathsf{M}_\lambda, 1^t, m(\lambda), \pi) = 0$, return to Item 9b.
      v. Extract from the proof, $w_i := \mathsf{BARG.Extract}(\tau, \pi)$. //For readability, the second w_i extracted is ignored.
      vi. Parse w_i as $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i, \rho_i^*)$.
      vii. Set $y'_{i-1} = y_{i-1}$, and $\widehat{c}'_i = \widehat{c}_i$. //to distinguish from y_{i-1} and $\widehat{c}_i$ in L_{i-1}
      viii. Set $\widetilde{\mathsf{z}}_i = \mathsf{z}_{\mathsf{g},m(\lambda)}$. //to distinguish the final state $\mathsf{z}_{\mathsf{g},m(\lambda)}$ output by $\mathcal{A}$ in each list
      ix. Add $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i, \widehat{c}_{m(\lambda)}) \in \{0,1\}^{m'}$ to L_i. //m' is the length of each element in the list
   (c) If $|\mathsf{L}_i| < \kappa(\lambda)$ output $\perp$. Otherwise, truncate its length to exactly $\kappa(\lambda)$.
10. Set $\mathsf{L}'_1 = \mathsf{L}_1 \in \{0,1\}^{m' \cdot \kappa(\lambda)}$.
11. For every $i \in [m(\lambda) - 1]$
   (a) Let $\mathsf{f}'_i : \{0,1\}^{m' \cdot i} \to \{0,1\}^{3 \cdot s(\lambda) + \ell(\lambda)}$ and $\mathsf{f}_{i+1} : \{0,1\}^{m'} \to \{0,1\}^{3 \cdot s(\lambda) + \ell(\lambda)}$ be functions defined as follows, where x and sk are hardcoded in the function:

$$\begin{aligned}
&\mathsf{f}'_i((\mathsf{z}_{\mathsf{g},j-1}, \widehat{c}_{j-1}, \widehat{c}'_j, y'_{j-1}, y_j, \widetilde{\mathsf{z}}_j, \widehat{c}_{m(\lambda)})_{j \in [i]}) \\
&= (\mathsf{g}_\lambda(y_i, \mathsf{z}_{\mathsf{g},i-1}), \mathsf{f}_\lambda(x, y_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{i-1})), \widetilde{\mathsf{z}}_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})), y_i \\
&\mathsf{f}_{i+1}(\mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, \widehat{c}'_{i+1}, y'_i, y_{i+1}, \widetilde{\mathsf{z}}_{i+1}, \widehat{c}_{m(\lambda)}) \\
&= (\mathsf{z}_{\mathsf{g},i}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_i), \widetilde{\mathsf{z}}_{i+1}, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})), y'_i
\end{aligned}$$

   (b) Compute $\mathsf{L}'_{i+1} \in \{0,1\}^{m' \cdot (i+1) \cdot \kappa(\lambda)}$, the coupling of L'_i and L_{i+1} using the algorithm in claim 2.
12. Sample uniformly from the list $\mathsf{L}'_{m(\lambda)}$, i.e.

$$(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, \widehat{c}'_i, y'_{i-1}, y_i, \widetilde{\mathsf{z}}_i, \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)}))_{i \in [m(\lambda)]} \leftarrow \mathsf{L}'_{m(\lambda)} .$$

13. Output $y = (y_1, \ldots, y_{m(\lambda)})$.

Figure 8: Simulator Sim

Client-output indistinguishability. We consider the function

$$\mathsf{F}'_\lambda(x, y = (y_1, \ldots, y_{m(\lambda)})) = \begin{cases} \mathsf{Search}_\lambda(x, y), & \text{if } \forall i : \ (y_i, y_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases}.$$

First we argue that Sim still satisfies client-output indistinguishability with respect to F′ (as in the original protocol). Formally, we consider Out′ that outputs $z = \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$, as in the original protocol. We argue that the following experiments are indistinguishable:

$\underline{\mathsf{REAL}^{\mathsf{F}'}_{\mathcal{A}}(\lambda, x)}$	$\underline{\mathsf{IDEAL}^{\mathsf{F}'}_{\mathsf{Sim}}(\lambda, x)}$
$\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$	$y \leftarrow \mathsf{Sim}(1^\lambda, x)$
$(\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x)$	If $y = \bot$, set $i^* \leftarrow \bot$
$s \leftarrow \mathcal{A}(\mathsf{crs}, c)$	If $y \neq \bot$, set $i^* \leftarrow \mathsf{F}'_\lambda(x, y)$
$i^* \leftarrow \mathsf{Out}'(\mathsf{crs}, \mathsf{st}, s)$	Output i^*
Output i^*	

This follows as in Theorem 5. Indeed, the only modification is the additional coupling. We can argue that the coupling can be done with an identical claim to Lemma 1, where indistinguishability of $\mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$ follows just by using index hiding of the BARG. The rest of the proof is identical.

Now, let $\mathcal{D}$ be a polynomial-size distinguisher. We first split the real world experiment based on $i^* = \mathsf{Out}'(\mathsf{crs}, \mathsf{st}, s)$. We have

$$\begin{aligned} \Pr\left[\ \mathcal{D}(z) = 1 \quad : \quad z \leftarrow \mathsf{REAL}^{\mathsf{F}}_{\mathcal{A}}(\lambda, x)\ \right] = \sum_{i=1}^{m(\lambda)} \Pr\left[\begin{array}{c} \mathcal{D}(z) = 1 \\ \wedge\ i^* = i \end{array} \quad : \quad (z, i^*) \leftarrow \mathsf{REAL}^{\mathsf{F},\mathsf{F}'}_{\mathcal{A}}(\lambda, x)\right] \\ + \Pr\left[\begin{array}{c} \mathcal{D}(z) = 1 \\ \wedge\ i^* \notin [m(\lambda)] \end{array} \quad : \quad (z, i^*) \leftarrow \mathsf{REAL}^{\mathsf{F},\mathsf{F}'}_{\mathcal{A}}(\lambda, x)\right], \end{aligned}$$

where $\mathsf{REAL}^{\mathsf{F},\mathsf{F}'}$ denotes the output of both Out and Out′ in the real world experiment.

Case I: Search **finds an index.** Let $i \in [m(\lambda)]$. We consider the experiment of extracting from the BARG, expanding the output of the real world experiment $\mathsf{REAL}^{\mathsf{F},\mathsf{F}'}_{\mathcal{A}}$:

$\underline{\mathsf{EXP}_i(1^\lambda, x)}$

1. Sample $(\mathsf{crs}_{\mathsf{BARG}}, \tau) \leftarrow \mathsf{BARG.Gen}(1^\lambda, \{i, i\})$.
2. Sample $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$ keeping the above $\mathsf{crs}_{\mathsf{BARG}}$.
3. Generate $(\mathsf{st}, c) \leftarrow \mathsf{Client}(\mathsf{crs}, x)$.
4. Run the adversary to obtain $s = (\pi, \mathsf{rt}, \mathsf{rt}^*, \mathsf{z}_{\mathsf{g},m(\lambda)}, \widehat{c}_{m(\lambda)}) \leftarrow \mathcal{A}(\mathsf{crs}, c)$.
5. Extract from the proof, $w_i, w'_i := \mathsf{BARG.Extract}(\tau, \pi)$.
6. Parse w_i as $(\mathsf{z}_{\mathsf{g},i-1}, \widehat{c}_{i-1}, y_{i-1}, \rho_{i-1}, \mathsf{z}_{\mathsf{g},i}, \widehat{c}_i, y_i, \rho_i, \rho^*_i)$.

7. Set $z \leftarrow \mathsf{Out}(\mathsf{crs}, \mathsf{st}, s)$ and $i^* \leftarrow \mathsf{Out}'(\mathsf{crs}, \mathsf{st}, s)$.

8. Output z, i^*, y_i.

By index hiding of the BARG, we have

$$\left| \Pr\left[\begin{array}{c} \mathcal{D}(z) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (z, i^*) \leftarrow \mathsf{REAL}_{\mathcal{A}}^{\mathsf{F},\mathsf{F}'}(\lambda, x) \right] - \Pr\left[\begin{array}{c} \mathcal{D}(z) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (z, i^*, y_i) \leftarrow \mathsf{EXP}_i(1^\lambda, x) \right] \right| \leq \mathsf{negl}(\lambda) \ .$$

By the somewhere extractability of the BARG for M_λ (Fig. 6), we have that in EXP_i, except with negligible probability

$$\mathsf{SEH.Verify}(\mathsf{hk}^*, \mathsf{rt}^*, m(\lambda), i, \mathsf{Project}(y_i), \rho_i^*) = 1 \ ,$$

where recall that the BARG proof is verified with respect to hk^* that contains the ciphertext $\widehat{c}_{m(\lambda)}$. Furthermore, the Out' algorithm outputs $i^* = \mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$, and the Out algorithm outputs $z = \mathsf{SEH.Extract}(\mathsf{td}^*, \mathsf{rt}^*)$. Therefore, by extraction correctness of the SEH, it follows that in the above experiment, except with negligible probability $z = \mathsf{Project}(y_i)$. Therefore,

$$\left| \Pr\left[\begin{array}{c} \mathcal{D}(z) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (z, i^*, y_i) \leftarrow \mathsf{EXP}_i(1^\lambda, x) \right] - \Pr\left[\begin{array}{c} \mathcal{D}(\mathsf{Project}(y_i)) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (z, i^*, y_i) \leftarrow \mathsf{EXP}_i(1^\lambda, x) \right] \right| \leq \mathsf{negl}(\lambda) \ .$$

We now consider the ideal world. We recall that the simulator Sim does additional coupling based on $\mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)})$. Below we include this as part of the simulator output, named i^*. By construction of the simulator Sim (which uses EXP_i and coupling), we have that the simulated i^*, y_i are distributed identically to i^*, y_i above. Therefore,

$$\left| \Pr\left[\begin{array}{c} \mathcal{D}(\mathsf{Project}(y_i)) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (z, i^*, y_i) \leftarrow \mathsf{EXP}_i(1^\lambda, x) \right] - \Pr\left[\begin{array}{c} \mathcal{D}(\mathsf{Project}(y_i)) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (i^*, y) \leftarrow \mathsf{Sim}(1^\lambda, x) \right] \right| \leq \mathsf{negl}(\lambda) \ .$$

We now use client-output indistinguishability of Sim with respect to F'. In fact, we use the fact that in the simulated world, we have $i^* = \mathsf{F}'(x, y)$ and therefore $\mathsf{Project}(y_{i^*}) = \mathsf{F}(x, y)$. Indeed, this follows since the simulated output is by definition the state in the final row, which is i^*. We get

$$\left| \Pr\left[\begin{array}{c} \mathcal{D}(\mathsf{Project}(y_i)) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (i^*, y) \leftarrow \mathsf{Sim}(1^\lambda, x) \right] - \Pr\left[\begin{array}{c} \mathcal{D}(\mathsf{F}(x, y)) = 1 \\ \wedge\ i^* = i \end{array} \;:\; (i^*, y) \leftarrow \mathsf{Sim}(1^\lambda, x) \right] \right| \leq \mathsf{negl}(\lambda) \ .$$

Case II: Search **does not find a match.** We now handle the case of $i^* \notin [m(\lambda)]$. In the real world experiment, this means $\mathsf{FHE.Dec}(\mathsf{sk}, \widehat{c}_{m(\lambda)}) = \mathsf{Out}'(\mathsf{crs}, \mathsf{st}, s) \notin [m(\lambda)]$. As above, considering the experiment of extracting from the BARG (at an arbitrary index, say $j = 1$), we obtain a valid opening of rt^*. From SEH extraction correctness for invalid indices (Section 3.5), we get that except with negligible probability, $\mathsf{Out}(\mathsf{crs}, \mathsf{st}, s) = \mathsf{SEH.Extract}(\mathsf{td}^*, \mathsf{rt}^*) = \bot$.

In the ideal experiment, by definition of Index, the output of F is $\bot$ iff $y = \bot$ or $\mathsf{F}'(x, y) = i^* \notin [m(\lambda)]$ (we assume that Project cannot output the special string $\bot$). Using these observations, and client-output indistinguishability of Sim with respect to F', we get

$$\begin{aligned}
&\left|\Pr\left[\begin{array}{c}\mathcal{D}(z)=1\\ \wedge\, i^* \notin [m(\lambda)]\end{array} \;:\; (z, i^*) \leftarrow \mathsf{REAL}^{\mathsf{F},\mathsf{F}'}_{\mathcal{A}}(\lambda, x)\right] - \Pr\left[\begin{array}{c}\mathcal{D}(z)=1\\ \wedge\, z=\bot\end{array} \;:\; z \leftarrow \mathsf{IDEAL}^{\mathsf{F}}_{\mathsf{Sim}}(\lambda, x)\right]\right| \\
=&\left|\Pr\left[\begin{array}{c}\mathcal{D}(\bot)=1\\ \wedge\, i^* \notin [m(\lambda)]\end{array} \;:\; i^* \leftarrow \mathsf{REAL}^{\mathsf{F}'}_{\mathcal{A}}(\lambda, x)\right] - \Pr\left[\begin{array}{c}\mathcal{D}(\bot)=1\\ \wedge\, i^* \notin [m(\lambda)]\end{array} \;:\; i^* \leftarrow \mathsf{IDEAL}^{\mathsf{F}'}_{\mathsf{Sim}}(\lambda, x)\right]\right| \\
=&\mathsf{negl}(\lambda)\ .
\end{aligned}$$

Finally, adding up for all $i \in [m(\lambda)]$, and for the case of $i \notin [m(\lambda)]$, we get

$$\begin{aligned}
&\Big|\Pr\left[\ \mathcal{D}(z)=1 \quad : \quad z \leftarrow \mathsf{REAL}^{\mathsf{F}}_{\mathcal{A}}(\lambda, x)\ \right] \\
&\quad - \Pr\left[\ \mathcal{D}(z)=1 \quad : \quad z \leftarrow \mathsf{IDEAL}^{\mathsf{F}}_{\mathsf{Sim}}(1^\lambda, x)\ \right]\Big| \leq (m(\lambda)+1)\cdot \mathsf{negl}(\lambda)\ ,
\end{aligned}$$

which concludes the proof.

Server-input indistinguishability with respect to G'. This follows identically to Theorem 5.

This concludes the proof of Theorem 6.

7 Main Result: Private Set Membership and VDL with UP Search

Our first result is a protocol for the *dictionary lookup* function. Let $n(\lambda), \ell(\lambda), m(\lambda)$ be polynomially bounded functions. The dictionary lookup function:

$$\mathsf{DL} = \{\mathsf{DL}_\lambda : \{0,1\}^{n(\lambda)} \times (\{0,1\}^{n(\lambda)+\ell(\lambda)})^{m(\lambda)} \to \{0,1\}^*\}_{\lambda \in \mathbb{N}}\ ,$$

takes as input a key $k \in \{0,1\}^{n(\lambda)}$ and a dictionary D represented as a sequence of key-value pairs:

$$D = ((k_1, v_1), \ldots, (k_{m(\lambda)}, v_{m(\lambda)})) \in \left(\{0,1\}^{n(\lambda)} \times \{0,1\}^{\ell(\lambda)}\right)^{m(\lambda)}\ .$$

The output is the value v such that $(k, v) \in D$, or $\bot$ if no such value exists, or if the keys $k_1, \ldots, k_{m(\lambda)}$ are not pairwise distinct.

Theorem 8. *Assuming NIZK, FHE and BARG, there exists a 1-round 2-party computation protocol for the function* DL *in the CRS model with weak client security (Section 4.1) where the client's message is of size* $n(\lambda) \cdot \mathsf{poly}(\lambda)$ *and the server's message is of size* $(n(\lambda) + \ell(\lambda)) \cdot \mathsf{poly}(\lambda)$.

We also consider an extension that we call *verifiable dictionary lookup* where the server's input D is required to satisfy some predicate. To specify the type of predicates we consider, we first define the notion of a bounded space predicate.

Definition 13 (Bounded State Predicate)**.** *Let $s(\lambda)$ be a polynomially bounded function. A predicate $\mathsf{P} = \{\mathsf{P}_\lambda : (\{0,1\}^{\ell(\lambda)})^{m(\lambda)} \to \{0,1\}\}_{\lambda\in\mathbb{N}}$ is computable with state of size s if there exists an efficiently computable function $\Gamma = \{\Gamma_\lambda : \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{s(\lambda)} \to \{0,1\}^{s(\lambda)}\}_{\lambda\in\mathbb{N}}$ and special states $\mathsf{st}, \mathsf{st}' \in \{0,1\}^{s(\lambda)}$ such that:*

$$\mathsf{P}_\lambda(y_1, \ldots, y_{m(\lambda)}) = 1 \iff \Gamma_\lambda(y_{m(\lambda)}, \Gamma_\lambda(y_{m(\lambda)-1}, \Gamma_\lambda(\ldots, \Gamma_\lambda(y_1, \mathsf{st})))) = \mathsf{st}' \ .$$

Definition 14 (UP Relation)**.** *A relation $\mathsf{Rel} = \left\{\mathsf{Rel}_\lambda \subseteq \{0,1\}^{n(\lambda)} \times \{0,1\}^{w(\lambda)}\right\}$ is a UP relation if it is an NP relation (i.e. there is a polynomial-time algorithm for deciding $(x,w) \in \mathsf{Rel}_\lambda$), and for every instance $x \in \{0,1\}^{n(\lambda)}$, there exists at most one witness $w \in \{0,1\}^{w(\lambda)}$ such that $(x,w) \in \mathsf{Rel}_\lambda$.*

Theorem 9. *Let $\mathsf{P} = \{\mathsf{P}_\lambda : (\{0,1\}^{n(\lambda)+\ell(\lambda)})^{m(\lambda)} \to \{0,1\}\}_{\lambda\in\mathbb{N}}$ be a predicate computable with state of size $s(\lambda) = O(\log\lambda)$. Let $\mathsf{R} = \left\{\mathsf{R}_\lambda \subseteq \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{\ell(\lambda)}\right\}$ be an efficiently computable strict total order, and let Rel be a UP relation. Let $\mathsf{VDL} = \{\mathsf{VDL}_\lambda\}_{\lambda\in\mathbb{N}}$ be the function:*

$$\mathsf{VDL}_\lambda\left(x, D = ((w_i, v_i))_{i\in[m(\lambda)]}\right) = \begin{cases} \mathsf{DL}_\lambda^{\mathsf{Rel}}(x, D), & \text{if } \mathsf{P}_\lambda(D) = 1 \ \wedge \ \forall i : \ (w_i, w_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases} ,$$

where $\mathsf{DL}_\lambda^{\mathsf{Rel}}(x, D)$ outputs v such that $(w, v) \in D$ and $(x, w) \in \mathsf{Rel}$, or $\bot$ if no such value exists.

Assuming NIZK, FHE and BARG, there exists a 1-round 2-party computation protocol for the function VDL in the CRS model with weak client security (Section 4.1) where the client's message is of size $n(\lambda) \cdot \mathsf{poly}(\lambda)$ and the server's message is of size $(n(\lambda) + \ell(\lambda)) \cdot \mathsf{poly}(\lambda)$.

Corollary 1. *Theorem 9 implies Theorem 8.*

Proof sketch. The corollary follows from appropriately setting the predicate P, total order R and UP relation Rel in the function VDL such that a protocol for VDL is also a protocol for DL. Specifically, for all λ, we set P_λ to be the tautology predicate, i.e. $\mathsf{P}_\lambda(\cdot) = 1$, which can be computed with state of size $O(1)$. Next, we set R_λ to be the *strict* total order, i.e. $\forall k, k' \in \{0,1\}^{\ell(\lambda)}$, $(k, k') \in \mathsf{R}_\lambda \iff k < k'$, where $k < k'$ is interpreted as the standard numerical comparison of the integers represented by the binary strings k and k', and thus checking membership in R_λ efficient. The UP relation Rel is the equality relation, $(x, w) \in \mathsf{Rel}$ if $x = w$.

The server, on input a database $D = ((w_i, v_i))_{i\in[m(\lambda)]}$ first converts D into a database $D' = ((w'_i, v'_i))_{i\in[m(\lambda)]}$, which is simply the original database D sorted by the key value. The client and the server then run the protocol for VDL, where the client input remains unchanged, and the server input is D'. □

Given the above corollary, we focus on proving Theorem 9.

Remark 9. *We note that while we explicitly assume the existence of NIZKs in addition to BARGs, it was recently shown [BKP+24, BWW24] that BARGs, along with one-way functions, imply NIZKs.*

7.1 Input-Dependent Simulation to Weak Simulation

In this section, we show how to obtain weak simulation security for verifiable dictionary lookup (VDL).

We show this via a meta-theorem, going from input-dependent to weak simulation-security: for an appropriate choice of G′, input-dependent simulation-security for VDL with respect to G′ implies weak simulation-security.

Specifically, we define

$$\mathsf{G}_\lambda(D = ((w_i, v_i))_{i\in[m(\lambda)]}) = \begin{cases} 1, & \text{if } \mathsf{P}_\lambda(D) = 1 \\ \bot, & \text{otherwise} \end{cases},$$

and

$$\mathsf{G}'_\lambda(D = ((w_i, v_i))_{i\in[m(\lambda)]}) = \begin{cases} \mathsf{G}(D), & \text{if } \forall i: \ (w_i, w_{i+1}) \in \mathsf{R}_\lambda \\ \bot, & \text{otherwise} \end{cases}.$$

Observe that if $\mathsf{G}'_\lambda(D) = 1$, then $\mathsf{VDL}_\lambda(x, D) = \mathsf{DL}^{\mathsf{Rel}}_\lambda(x, D)$ and if $\mathsf{G}'_\lambda(D) = \bot$, then $\mathsf{VDL}_\lambda(x, D) = \bot$. G'_λ then effectively acts as an indicator to whether the output of VDL is $\bot$ due to either the total ordering or predicate being violated on D.

Before proving this meta-theorem, we argue that our general theorems (Section 6) indeed give us input-dependent simulation-secure protocols for VDL, G′. In particular, we show that VDL, G′ can be represented as sorted lookup functions (Theorem 6). For completeness, we first show that for VDL with short values $|v_i| = O(\log \lambda)$, we can represent VDL, G′ as sorted bounded-state functions (Theorem 5).

Sorted bounded-state functions. Define $\mathsf{F} = \{\mathsf{F}_\lambda\}_{\lambda\in\mathbb{N}}$ as

$$\mathsf{F}_\lambda(x, D = ((w_i, v_i))_{i\in[m(\lambda)]}) = \begin{cases} \mathsf{DL}^{\mathsf{Rel}}_\lambda(x, D), & \text{if } \mathsf{P}_\lambda(D) = 1 \\ \bot, & \text{otherwise} \end{cases}$$

Observe that F′ instantiated with F described above is indeed VDL. Next we show that F is computable with state of size $s(\lambda) = O(\log \lambda)$. For Theorem 9, we work with predicates $\mathsf{P} = \{\mathsf{P}_\lambda\}_{\lambda\in\mathbb{N}}$ that are computable with state of size $s(\lambda)$. Thus, all that is left to show is that $\mathsf{DL}^{\mathsf{Rel}}_\lambda(x, D)$ can be computed with a state of size $O(\log \lambda)$. This is easy to see when $|v_i| = O(\log \lambda)$ for all i. In particular, we define $\mathsf{f} = \{\mathsf{f}_\lambda\}_{\lambda\in\mathbb{N}}$ as below,

$$\mathsf{f}_\lambda(x, (w_i, v_i), z_{i-1}) = \begin{cases} v_i & \text{if } (x, w_i) \in \mathsf{Rel} \\ z_{i-1} & \text{otherwise} \end{cases}$$

where z_{i-1} is the state so far, and $z_0 = 0$ (or a special string to indicate no match). In words, f simply updates the state to v_i if there is a match, otherwise leaves the state unchanged.

Next, it is easy to observe that G is computable with state of size $O(\log \lambda)$ given that P is computable with state of size $O(\log \lambda)$.

Sorted lookup functions. Define $\mathsf{Search} = \{\mathsf{Search}_\lambda\}_{\lambda\in\mathbb{N}}$ as follows: it simply returns the index of the match, if one exists.

$$\mathsf{Search}_\lambda(x, D = ((w_i, v_i))_{i\in[m(\lambda)]}) = \begin{cases} i, & \text{if } (x, w_i) \in \mathsf{Rel} \wedge \mathsf{P}_\lambda(D) = 1 \\ \bot, & \text{otherwise.} \end{cases}$$

For predicates P computable in $O(\log \lambda)$ space, Search can also be computed in $O(\log \lambda)$ space. This follows from the fact that the matched index can be computed using the function f_λ using $O(\log \lambda)$ space as defined below,

$$\mathsf{f}_\lambda(x, (w_i, v_i), (b_{i-1}, z_{i-1})) = \begin{cases} (1, z_{i-1} + 1) & \text{if } \mathsf{Rel}(x, w_i) = 1 \\ (0, z_{i-1} + 1) & \text{otherwise}, \end{cases}$$

where $(b_0, z_0) = (0, 0)$ is the initial state.

Finally, we define Project such that $\mathsf{Project}(w, v) = v$.

We now prove the meta-theorem that shows that for our choice of F', G', input-dependent simulation-security can be upgraded to weak-simulation security. This theorem, along with the input privacy of the client directly implies Theorem 9.

Theorem 10. *For the functions $\mathsf{F}' = \{\mathsf{F}'_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathsf{G}' = \{\mathsf{G}'_\lambda\}_{\lambda\in\mathbb{N}}$ defined above, any protocol in the CRS model function for F' in the CRS model with input-dependent simulation-security with respect to G' (Definitions 8 and 9) also satisfies weak simulation-security (Definition 7).*

Proof. Fix adversary $\mathcal{A}$ and polynomial P. Let Sim' be the simulator that satisfies input-dependent simulation-security with respect to G'.

We now define the weak simulation-security simulator Sim below, which simply uses the input 0 to Sim' when an input is not specified.

$\underline{\mathsf{Sim}(1^\lambda)}$	$\underline{\mathsf{Sim}(1^\lambda, x)}$
1. Output $\mathsf{Sim}'(1^\lambda, 0)$.	1. Output $\mathsf{Sim}'(1^\lambda, x)$.

We want to show that for any polynomial size distinguisher $\mathcal{D}$, the following hold for large enough λ,

$$|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr[\mathcal{D}(\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)) = 1]| \leq \frac{1}{P(\lambda)} \ .$$

Here the ideal experiment is defined in Definition 7.

To prove the above bound, we define the following function $\mathsf{IDEAL}'(\lambda, x, y)$ to simultaneously capture the ideal experiments in both the input-dependent simulation-security and weak simulation-security.

$\underline{\mathsf{IDEAL}'(\lambda, x, D)}$
If $D = \bot$, set $z \leftarrow \bot$
If $D \neq \bot$, set $z \leftarrow \mathsf{F}'_\lambda(x, D)$
Output z

Thus, from the description of Sim, we have that:

1. $\mathsf{IDEAL}'(\lambda, x, D)$ where $D \leftarrow \mathsf{Sim}'(0)$ corresponds to the ideal experiment $\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)$ in weak simulation-security definition (Definition 7).
2. $\mathsf{IDEAL}'(\lambda, x, D)$ where $D \leftarrow \mathsf{Sim}'(x)$ corresponds to the ideal experiment $\mathsf{IDEAL}_{\mathsf{Sim}}(\lambda, x)$ in input-dependent simulation-security definition (Definition 8).

For notational convenience, we drop the security parameter 1^λ from the simulator input.

Reducing to input-dependent simulation. Thus, we can rewrite the distinguishing advantage as

$$\begin{aligned}
&\left|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr_{D\leftarrow\mathsf{Sim}'(0)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda, x, D)) = 1]\right| \\
&\quad\leq \left|\Pr[\mathcal{D}(\mathsf{REAL}_{\mathcal{A}}(\lambda, x)) = 1] - \Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda, x, D)) = 1]\right| \qquad (11) \\
&\quad + \quad \left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda, x, D)) = 1] - \Pr_{D\leftarrow\mathsf{Sim}'(0)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda, x, D)) = 1]\right| \qquad (12)
\end{aligned}$$

By the client-output indistinguishability of input-dependent simulation-security (Definition 8), we have that Eq. (11) is bounded above by $\frac{1}{4P(\lambda)}$. Thus it suffices for us to argue that Eq. (12) is bounded above by $\frac{1}{4P(\lambda)} + \mathsf{negl}(\lambda)$ to argue weak-simulation security.

Ideal World Events. We now describe events in the ideal world based on the simulator output, that will be useful to bound Eq. (12).

$$\begin{aligned}\mathsf{SIM}_\perp &= \text{ Sim output } \perp \\ \mathsf{G}_\perp &= \overline{\mathsf{SIM}_\perp} \wedge \mathsf{G}'_\lambda(D) = \perp \\ \mathsf{Z}_\perp &= \mathsf{SIM}_\perp \vee \mathsf{G}_\perp\end{aligned}$$

Note that events $\mathsf{SIM}_\perp$ and $\mathsf{G}_\perp$ are disjoint by definition. Further, the event $\overline{\mathsf{Z}_\perp}$ corresponds to the event where Sim output $D \neq \perp$ *and* $\mathsf{G}'_\lambda(D) = 1$, i.e. $\overline{\mathsf{Z}_\perp} = \overline{\mathsf{SIM}_\perp} \wedge \overline{\mathsf{G}_\perp}$. When $\overline{\mathsf{Z}_\perp}$ occurs, this corresponds to $\mathsf{F}'_\lambda(x, D) = \mathsf{DL}^{\mathsf{Rel}}_\lambda(x, D)$ whereas when $\mathsf{Z}_\perp$ occurs, $\mathsf{G}'_\lambda(D) = \perp$ and this corresponds to $\mathsf{F}'_\lambda(x, D) = \perp$.

Separating into two cases. To bound Eq. (12), it will be useful to separate it out into two cases, depending on whether the output is $\perp$.

$$\begin{aligned}&\left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1] - \Pr_{D\leftarrow\mathsf{Sim}'(0)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1]\right| \\ &\leq \left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1 \mid \mathsf{Z}_\perp]\Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathsf{Z}_\perp]\right. \\ &\quad \left. - \Pr_{D\leftarrow\mathsf{Sim}'(0)}[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1 \mid \mathsf{Z}_\perp]\Pr[\mathsf{Z}_\perp]\right| \\ &\quad + \left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\overline{\mathsf{Z}_\perp}\right]\right. \\ &\quad \left. - \Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\overline{\mathsf{Z}_\perp}\right]\right| \\ &\leq \left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow\mathsf{Sim}'(0)}[\mathsf{Z}_\perp]\right| \qquad (13) \\ &\quad + \left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\overline{\mathsf{Z}_\perp}\right]\right. \\ &\quad \left. - \Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\mathcal{D}(\mathsf{IDEAL}'(\lambda,x,D))=1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\overline{\mathsf{Z}_\perp}\right]\right| \qquad (14)\end{aligned}$$

where the second inequality uses the fact that the distinguisher gets $\perp$ in both experiments.

We now separately bound the probability of Eq. (13) and Eq. (14).

Case I: Output is $\perp$. We start with Eq. (13), bounding the difference in probability that the output in the two ideal worlds are $\perp$. We will use the server-input indistinguishability of Sim′ with respect to G′ (Definition 9) to argue this. Specifically, we bound Eq. (13) to be

$$\left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow\mathsf{Sim}'(0)}[\mathsf{Z}_\perp]\right| \leq \frac{1}{8P(\lambda)}.$$

For the sake of contradiction, let us assume that

$$\left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}[\mathsf{Z}_\perp]\right| > \frac{1}{8P(\lambda)}.$$

We construct a distinguisher $\mathcal{D}'$ for Definition 9 on inputs x and 0 and any i in such that $\mathcal{D}$ wins with probability at least $\frac{1}{2}+\frac{1}{8P(\lambda)}$. $\mathcal{D}$ on input $(\cdot,\perp)$ outputs 1 else it outputs a bit b at random. We use the shorthand D_i below to indicate (w_i, v_i).

$$\begin{aligned}
&\left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1\right] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1\right]\right| \\
&= \left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \mathsf{Z}_\perp\right]\Pr_{D\leftarrow \mathsf{Sim}'(x)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \mathsf{Z}_\perp\right]\Pr_{D\leftarrow \mathsf{Sim}'(0)}[\mathsf{Z}_\perp]\right. \\
&\qquad \left. + \Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\overline{\mathsf{Z}_\perp}\right]\right| \\
&= \left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}[\mathsf{Z}_\perp] + \frac{1}{2}\left(\Pr_{D\leftarrow \mathsf{Sim}'(0)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow \mathsf{Sim}'(x)}[\mathsf{Z}_\perp]\right)\right| \\
&= \frac{1}{2}\left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}[\mathsf{Z}_\perp] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}[\mathsf{Z}_\perp]\right| > \frac{1}{16P(\lambda)}
\end{aligned}$$

The second equality follows from the fact that when the event $\mathsf{Z}_\perp$, by definition, the second input to $\mathcal{D}'$ is $\perp$ and by construction $\mathcal{D}'$ outputs 1 in this case. Additionally, when $\overline{\mathsf{Z}_\perp}$ occurs, $\mathcal{D}'$ receives as input $(\cdot, 1)$ and simply outputs 1 with probability $1/2$.

Since the distinguishing and prediction advantage are equivalent within a factor of 2, we have that $\mathcal{D}'$ wins with probability at least $\frac{1}{2}+\frac{1}{32P(\lambda)}$. This contradicts the server-input indistinguishability of with respect to G', which is bounded by a negligible function μ.

Case II: Output is not $\perp$. Now we look to bound Eq. (14). In fact, following the above bound for Eq. (13) we can bound the probability $\left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\overline{\mathsf{Z}_\perp}\right]\right|$ by any inverse polynomial, and therefore it suffices to consider the bound on the following expression

$$\left|\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\mathcal{D}(\mathsf{IDEAL}'(\lambda, x, D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] - \Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\mathcal{D}(\mathsf{IDEAL}'(\lambda, x, D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\right| \tag{15}$$

Note that if $\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\overline{\mathsf{Z}_\perp}\right] \leq \frac{1}{32P(\lambda)}$, then we have that $\Pr_{D\leftarrow \mathsf{Sim}'(0)}\left[\overline{\mathsf{Z}_\perp}\right] \leq \frac{1}{32P(\lambda)} + \mu$, and this already allows us to bound Eq. (13) with probability $\frac{1}{8P(\lambda)}$. Thus, without loss of generality assume that $\Pr_{D\leftarrow \mathsf{Sim}'(x)}\left[\overline{\mathsf{Z}_\perp}\right] > \frac{1}{32P(\lambda)}$.

Conditioned on $\overline{\mathsf{Z}_\perp}$, we have that $\mathsf{F}'_\lambda(x, D) = \mathsf{DL}^{\mathsf{Rel}}_\lambda(x, D)$ for databases D where the witnesses in D are all distinct - this is enforced by the total ordering of the witnesses when the event $\overline{\mathsf{Z}_\perp}$ occurs. Thus the output of $\mathsf{IDEAL}'(\lambda, x, D = ((w_i, v_i))_{i\in[m(\lambda)]})$ conditioned on $\overline{\mathsf{Z}_\perp}$ can be written as $\sum_{i=1}^{m(\lambda)}(\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\cdot v_i)$. If there is no match, i.e. $\sum_{i=1}^{m(\lambda)}(\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\cdot v_i) = 0$, we denote this event by NoMatch. It will now be convenient to explicitly write out $D \leftarrow \mathsf{Sim}'(x)$ and $D \leftarrow \mathsf{Sim}'(0)$ as $((w_i, v_i)_i) \leftarrow \mathsf{Sim}'(x)$ and $((w_i, v_i)_i) \leftarrow \mathsf{Sim}'(0)$ respectively. Here $(w_i, v_i)_i$ is a shorthand for $((w_i, v_i))_{i\in[m(\lambda)]}$.

Therefore, rewriting the above Eq. (15), we get

$$\left|\Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)}\left[\mathcal{D}\left(\sum_{i=1}^{m(\lambda)}(\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\cdot v_i)\right) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)}\left[\mathcal{D}\left(\sum_{i=1}^{m(\lambda)}(\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\cdot v_i)\right) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right]\right|$$

We now use the fact that conditioned on $\overline{\mathsf{Z}_\perp}$, all the witnesses $w_1, \ldots, w_{m(\lambda)}$ are strictly monotonically increasing, and thus all *distinct*. Thus, since Rel is a UP relation, for any fixed x there is at most one match and we can separate the above into disjoint events depending on which index i there was a match, or the case where there was no match at all.

$$
\begin{aligned}
&\left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathcal{D}\Big(\sum_{i=1}^{m(\lambda)} (\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \cdot v_i) \Big) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathcal{D}\Big(\sum_{i=1}^{m(\lambda)} (\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \cdot v_i) \Big) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \\
&\leq \left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathsf{NoMatch} \wedge (\mathcal{D}(0) = 1) \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathsf{NoMatch} \wedge (\mathcal{D}(0) = 1) \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \\
&\quad + \left| \sum_{i=1}^{m(\lambda)} \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \sum_{i=1}^{m(\lambda)} \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \\
&\leq \left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathsf{NoMatch} \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathsf{NoMatch} \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \\
&\quad + \sum_{i=1}^{m(\lambda)} \left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \\
&\leq \sum_{i=1}^{m(\lambda)} \left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \qquad (16) \\
&\quad + \sum_{i=1}^{m(\lambda)} \left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \qquad (17)
\end{aligned}
$$

where the third inequality follow from the fact that $\Pr\left[\mathsf{NoMatch} \,\middle|\, \overline{\mathsf{Z}_\perp}\right]$, the probability there is no match, is equal to $1 - \sum_i \Pr\left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \,\middle|\, \overline{\mathsf{Z}_\perp}\right]$.

For each i, we bound

$$
\left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| \leq \frac{1}{16 \cdot m(\lambda) \cdot P(\lambda)}
$$

and the bound for Eq. (16) and Eq. (17) then follows from summing up the $2 \cdot m(\lambda)$ terms. Note that the terms in Eq. (17) are a special case of the above.

As before, we shall rely on the server-input indistinguishability of Sim′ with respect to G′ (Definition 9) by constructing a distinguisher $\mathcal{D}'$ that depends on $\mathcal{D}$ from above. It will also have hardcoded as input instance x from above. Let us assume for the sake of contradiction that

$$
\left| \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] - \Pr_{((w_i,v_i)_i)\leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in \mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp} \right] \right| > \frac{1}{16 \cdot m(\lambda) \cdot P(\lambda)}
$$

We describe $\mathcal{D}'$ below.

$\mathcal{D}'((w_i, v_i), g = \mathsf{G}'_\lambda(D))$

If $g = \perp$, set $b \leftarrow 0$

Else if $g = 1$, set

$b \leftarrow 1$ if $(x, w_i) \in \mathsf{Rel} \wedge \mathcal{D}(v_i) = 1$,

$b \leftarrow 0$ otherwise

Output b

We can compute the distinguishing advantage of $\mathcal{D}$, again by separating into whether the event $\mathsf{Z}_\perp$ occurs or not.

$$\left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1\right] \right| \tag{18}$$

$$\begin{aligned}
&= \left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \mathsf{Z}_\perp\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} [\mathsf{Z}_\perp] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \mathsf{Z}_\perp\right] \Pr_{D \leftarrow \mathsf{Sim}'(0)} [\mathsf{Z}_\perp] \right. \\
&\quad \left. + \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right| \\
&= \left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathcal{D}'(D_i, \mathsf{G}'_\lambda(D)) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right|
\end{aligned} \tag{19}$$

Here we use the fact that in our description of the distinguisher $\mathcal{D}'$, it never outputs 1 conditioned on the event $\mathsf{Z}_\perp$. Next, we make explicit the event that leads to $\mathcal{D}'$ to output 1 conditioned on $\overline{\mathsf{Z}_\perp}$, i.e. $\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1$. We can then rewrite Eq. (19) as

$$\begin{aligned}
&\left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right| \\
&= \left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] \right. \\
&\quad \left. + \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right| \\
&\geq \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \right| && (20) \\
&\quad - \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right| && (21)
\end{aligned}$$

Next, we show a lower bound for Eq. (20), and an upper bound for Eq. (21) to provide a lower bound for the distinguishing advantage in Eq. (18).

For Eq. (21), we rely on the fact that $\left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right|$ is bounded by a negligible function μ, and thus

$$\Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}} \wedge \mathcal{D}(v_i) = 1 \,\middle|\, \overline{\mathsf{Z}_\perp}\right] \left| \Pr_{D \leftarrow \mathsf{Sim}'(x)} \left[\overline{\mathsf{Z}_\perp}\right] - \Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] \right| \leq \mathsf{negl}(\lambda).$$

For Eq. (20) we use the fact that $\Pr_{D \leftarrow \mathsf{Sim}'(0)} \left[\overline{\mathsf{Z}_\perp}\right] > \frac{1}{32 \cdot P(\lambda)}$. Combining with the our assumption,

we have

$$\begin{aligned}
&\Big|\Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\overline{\mathsf{Z}_\perp}\right]\Big|\Big|\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\wedge\mathcal{D}(v_i)=1\,\middle|\,\overline{\mathsf{Z}_\perp}\right]-\Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\wedge\mathcal{D}(v_i)=1\,\middle|\,\overline{\mathsf{Z}_\perp}\right]\Big|\\
&\quad>\frac{1}{32\cdot P(\lambda)}\Big|\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\wedge\mathcal{D}(v_i)=1\,\middle|\,\overline{\mathsf{Z}_\perp}\right]-\Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\wedge\mathcal{D}(v_i)=1\,\middle|\,\overline{\mathsf{Z}_\perp}\right]\Big|\\
&\quad>\frac{1}{32\cdot P(\lambda)}\cdot\frac{1}{16\cdot m(\lambda)\cdot P(\lambda)}
\end{aligned}$$

Putting it all together gives us a lower bound on the distinguishing advantage of our constructed $\mathcal{D}'$,

$$\left|\Pr_{D\leftarrow\mathsf{Sim}'(x)}\left[\mathcal{D}'(D_i,\mathsf{G}'_\lambda(D))=1\right]-\Pr_{D\leftarrow\mathsf{Sim}'(0)}\left[\mathcal{D}'(D_i,\mathsf{G}'_\lambda(D))=1\right]\right|>\frac{1}{512\cdot m(\lambda)\cdot P^2(\lambda)}-\mathsf{negl}(\lambda).$$

This contradicts the server-input indistinguishability of Sim' with respect to G' (Definition 9), and it must be the case that

$$\left|\Pr_{((w_i,v_i)_i)\leftarrow\mathsf{Sim}'(x)}\left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\wedge\mathcal{D}(v_i)=1\,\middle|\,\overline{\mathsf{Z}_\perp}\right]-\Pr_{((w_i,v_i)_i)\leftarrow\mathsf{Sim}'(0)}\left[\mathbb{1}_{(x,w_i)\in\mathsf{Rel}}\wedge\mathcal{D}(v_i)=1\,\middle|\,\overline{\mathsf{Z}_\perp}\right]\right|\leq\frac{1}{16\cdot m(\lambda)\cdot P(\lambda)}.$$

This completes the proof as we have bounded Eq. (12), showing weak simulation-security. □

Acknowledgements

Part of this work was done while Abhishek Jain and Omer Paneth were visiting the Simons Institute for the Theory of Computing.

Abhishek Jain was supported in part by NSF CAREER 1942789, Johns Hopkins University Catalyst award, JP Morgan Faculty Award, and research gifts from Ethereum Foundation, Stellar Development Foundation, and Cisco.

Arka Rai Choudhuri was supported by a grant by Open Philanthropy/Coefficient Giving.

Omer Paneth is a member of the CheckPoint Institute of Information Security and is supported by Len Blavatnik and the Blavatnik Foundation and by AFOSR Award FA9550-23-1-0312. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or AFOSR.

References

[ABJ+19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From FE combiners to secure MPC and back. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 199–228, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland. 6

[App21] Apple Inc. Password monitoring - apple support, 2021. https://support.apple.com/guide/security/password-monitoring-sec78e79fc3b/web. 4

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017. 3

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. 3

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 505–514, New York, NY, USA, May 31 – June 3, 2014. ACM Press. 3

[BDKP22] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable private information retrieval. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part III*, volume 13749 of *Lecture Notes in Computer Science*, pages 3–32, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland. 4, 5, 6, 8, 22, 23

[BDS25] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. Rate-1 statistical non-interactive zero-knowledge. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part VII*, volume 16006 of *Lecture Notes in Computer Science*, pages 514–547, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland. 16

[BGJP25] James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. Laconic PSI on authenticated inputs and applications. In *ASIACRYPT*, Lecture Notes in Computer Science. Springer, 2025. 6

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 474–482, Montreal, QC, Canada, June 19–23, 2017. ACM Press. 4

[BKP+24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron D. Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. In Bojan Mohar, Igor Shinkar, and

Ryan O'Donnell, editors, *56th Annual ACM Symposium on Theory of Computing*, pages 435–443, Vancouver, BC, Canada, June 24–28, 2024. ACM Press. 14, 51

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press. 4

[BWW24] Eli Bradley, Brent Waters, and David J. Wu. Batch arguments to NIZKs from one-way functions. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part II*, volume 15365 of *Lecture Notes in Computer Science*, pages 431–463, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland. 51

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. 5

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 635–668, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 15

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd Annual Symposium on Foundations of Computer Science*, pages 68–79, Denver, CO, USA, February 7–10, 2022. IEEE Computer Society Press. 4, 7, 14, 15

[CMOS25] Michele Ciampi, Ankit Kumar Misra, Rafail Ostrovsky, and Akash Shah. Black-box constant-round secure 2PC with succinct communication. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part V*, volume 15605 of *Lecture Notes in Computer Science*, pages 360–389, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland. 6

[DGGM19] Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In David Zuckerman, editor, *60th Annual Symposium on Foundations of Computer Science*, pages 661–685, Baltimore, MD, USA, November 9–12, 2019. IEEE Computer Society Press. 6

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. 3

[GHJ+23] Sanjam Garg, Mohammad Hajiabadi, Abhishek Jain, Zhengzhong Jin, Omkant Pandey, and Sina Shiehian. Credibility in private set membership. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13941 of *Lecture Notes in Computer*

Science, pages 159–189, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland. 6

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004. 4

[Goo19] Google Inc. Protect your accounts from data breaches with password checkup, 2019. https://security.googleblog.com/2019/02/protect-your-accounts-from-data.html. 4

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 520–549, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. 15

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery. 18, 44

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 406–425, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany. 3, 7

[JKKR17] Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Ron Rothblum. Distinguisher-dependent simulation in two rounds and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 158–189, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. 6

[KLM21] Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. Password monitor: Safeguarding passwords in microsoft edge, 2021. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/. 4

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM Symposium on Theory of Computing*, pages 1545–1552, Orlando, FL, USA, June 20–23, 2023. ACM Press. 15

[Kop24] Twan Koperberg. Couplings and matchings: Combinatorial notes on strassen's theorem. *Statistics & Probability Letters*, 209:110089, 2024. 22

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1115–1124, Phoenix, AZ, USA, June 23–26, 2019. ACM Press. 4

[Mar14] Moxie Marlinspike. The difficulty of private contact discovery, 2014. https://whispersystems.org/blog/contact-discovery/. 4

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, August 16–20, 1988. Springer Berlin Heidelberg, Germany. 13, 14

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. 3

[MPP20] Andrew Morgan, Rafael Pass, and Antigoni Polychroniadou. Succinct non-interactive secure computation. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 216–245, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. 6

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press. 5

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 433–463, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. 15