

Communication Preserving SFE from Obfuscation*

Omkant Pandey, Christopher Smith, and Yuhao Tang

Stony Brook University

{omkant, csasmith, yuhtang}@cs.stonybrook.edu

Abstract. The communication complexity of computing a function in the two party setting can be much smaller than requiring one of the parties to send its input. We revisit how this communication complexity must change if the function must be computed securely.

Hubáček and Wichs show that any protocol for computing a function securely must communicate at least as many bits as the length of the function’s output. However, all positive results in this direction require much more communication than optimal. In particular, the state of the art for general protocols requires a *multiplicative* overhead polynomial in the security parameter, achieving only “rate 0” asymptotically.

We show that, assuming the existence of input-succinct indistinguishability obfuscation for Turing machines, it is possible to achieve constant rate for this task. This compiler works even when the original protocol communicates single bits in each round, which is the main obstacle in prior works, and necessitates the use of obfuscation. The compiler can be based on standard non-succinct indistinguishability obfuscation for circuits (instead of Turing machines) in the common random string or random oracle models.

1 Introduction

In communication complexity, we are concerned with the number of bits two parties must exchange in order to compute a function over their joint inputs. There are interesting functions, such as “comparison” or “greater-than” function, for which the communication complexity can be much smaller than the length of either party’s input [Yan88, KN96, NN01]. Such protocols, however, are not concerned with security, often leaking unnecessary information. How does the communication complexity change if the function must be computed *securely*?

This problem was first considered by Naor and Nissim [NN01], who show that securely computing any function requires at most a multiplicative overhead in the security parameter.¹ More specifically, they show a compiler to transform any “plain” (possibly insecure) two-party protocol to compute a function f into a secure one, such that $\tilde{cc} = \text{poly}(\lambda) \cdot cc$ where cc and $\tilde{cc}$ denote the communication complexity of the plain and the secure protocol respectively, poly is a polynomial, and λ is the security parameter. After Gentry’s work on fully homomorphic encryption (FHE) [Gen09], such a compiler becomes straightforward where parties simply run an “encrypted” version of the original protocol and use secure function evaluation (SFE) to decrypt the output in polynomial time. In both of these cases, the *rate* of the compiled protocol, defined as $cc/\tilde{cc}$, is $1/\text{poly}(\lambda)$ or “rate 0” asymptotically. It is natural to inquire about better compilers.

Hubáček and Wichs [HW15] proved that any protocol for f that is secure according to the standard simulation-paradigm where the simulator and the honest parties are all polynomial time, must require at least as much communication as the length of the output(s). They also show that this bound can be avoided for *semi-honest* security using indistinguishability obfuscation (iO) [BGI⁺01, GGH⁺13, JLS21], and that some form of obfuscation is necessary to avoid it. The aforementioned FHE-based compiler depends on

* Supported in part by AFOSR Award FA9550-24-1-0156, NSF CAREER Award 2144303, and NSF grants 2106263 and 2524208. 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, AFOSR, or NSF.

¹ Their compiler requires the honest parties to run in exponential-time.

the output length when using SFE to decrypt the output. Thus we cannot hope to do better than $\tilde{\text{cc}}_f = \max(\text{cc}_f, \ell_f) + \text{poly}(\lambda)$ where ℓ_f is size of f 's output, cc_f and $\tilde{\text{cc}}_f$ are the communication-complexity of best plain and secure protocols for f , respectively.

The FHE based approach was significantly improved by Döttling, Goyal, Malavolta and Raizes [DGM22] who presented a compiler with communication complexity $\text{cc} + o(\text{cc}) + (L + \ell_f) \cdot \text{poly}(\lambda)$ where L and cc are the round- and communication- complexity of the original protocol, respectively. Although this protocol is still “rate 0” for the general case (where $L \approx \text{cc}$), it has “rate 1” for a large class of protocols, e.g., any *constant-round* protocol for *boolean* (or constant-length output) functions. The core difficulty in improving this compiler for the general case stems from the use of a rate-1 multi-key FHE for encrypting messages in each round which makes the ciphertexts at least as long as λ . This gets amortized over long messages (e.g., for constant round protocols) but not for short or single bit messages. The difficulty persists *even if* we switch to the Random Oracle Model (ROM) [BR93], or allow extractability-type non-falsifiable assumptions [Dam92, Nao03, BP04], since here the core issue is to be able to “compute over an encrypted protocol transcript” (as opposed to, say, extraction for simulation). A similar phenomenon is observed in the interactive-coding literature where the goal is to make protocols robust against a *noisy* channel. The primary challenge in this case too, is to have a “good” encoding for protocols sending *short* messages in each round, such as single bit messages [Sch93, Sch96, GMS11, BK12, CHS18]. Here “good” usually means a constant-rate protocol that can tolerate a constant-fraction of errors.

In this work, we explore whether good rate compilers are possible in the secure computation setting. That is, the compiled protocol should have *constant* rate even if the original protocol involves sending *single* bit messages in each round, and have standard simulation security. We show that such compilers exist assuming the existence of indistinguishability obfuscation for Turing machines, with appropriate parameters. We now describe our results in more detail.

1.1 Our Results

As discussed above, due to the lower bound in [HW15], the *optimal* communication for any secure² protocol for f must be at least $\max(\text{cc}_f, \ell_f)$ bits where ℓ_f is the output length of f . In fact, it is possible that the communication complexity cc of the given protocol (say π) is worse than that of the best possible protocol (which may not even be explicitly known). The optimal communication that any compiler can thus hope to achieve is $\text{opt}_\pi := \max(\text{cc}, \ell_f)$. Therefore, we will measure the rate of any compiler by comparing it to opt_π .

We say that a compiler for transforming a plain protocol π for f with communication complexity cc into a secure one for f with communication complexity $\tilde{\text{cc}}$ has *constant* rate if $\tilde{\text{cc}} = O(\text{opt}_\pi)$ and the constant in the O -notation does *not* depend on the security parameter.

We will need to assume the existence of (subexponentially secure) indistinguishability obfuscation (iO) for *Turing machines* where the size of the obfuscated program is polynomial only in the size of the original program and the security parameter; in particular, it should be independent of the length of the inputs it can accept, as well as the running time of the program. We refer to such obfuscations as *input-succinct* iO for Turing machines (defined formally in Def. 2). Our main result can be stated as follows.

Theorem 1 (Constant-Rate in Plain Model (Informal)). *Assuming the existence of input-succinct indistinguishability obfuscation for Turing machines (and other standard assumptions), there exists a constant-rate compiler that transforms any two-party protocol for a function f into a secure protocol for f .*

Some parts of our compiled protocol are reusable, and thus can be pushed to a short and uniform common reference string (CRS) to get a slightly better rate. By “short” we mean that the length of the CRS depends only on opt_π and λ . In fact the CRS in this case is only of length $\text{opt}_\pi + \text{poly}(\lambda)$, and for sake of clarity, we adopt this CRS-based approach to present our main compiler.

On input-succinct obfuscation. Candidate constructions of input-succinct iO for Turing machines (iOM) are known with various trade-offs, which we discuss next to put our result in context.

² I.e., secure as per the “ideal/real paradigm” under *polynomial* simulation (Def. 15).

In the plain model, the general constructions [ABG⁺13, BCP14, IPS15] of input-succinct iOM are only known from the unfalsifiable assumption that *differing-inputs* obfuscation (diO) for circuits exists [BGI⁺01] (as well as other standard assumptions which will be ignored for this discussion). Though the existence of diO was shown to be a subtle, and sometimes false, assumption by Garg, Gentry, Halevi and Wichs [GGHW17], the formulation of *public-coin* diO by Ishai, Pandey, and Sahai [IPS15] mitigates some of these concerns, and in particular, provides a candidate construction of iOM with no known attacks. Though public-coin diO is not efficiently falsifiable, it does give us an explicit compiler for which there are no known attacks.

Our result only requires iO (as opposed to diO) for Turing machines. Thus we can actually get a result under *falsifiable* assumptions in the common reference string model by relying on the construction of Badrinarayanan *et al.* [BFK⁺19]. They show that input-succinct iOM can be constructed directly from iO for circuits given access to a CRS whose length is allowed to depend on the length of the input accepted by the program. The size of the actual obfuscated TM is independent of its running time or the input-length. We will refer to such a CRS as “long” to distinguish it from the “short” CRS case mentioned earlier. Note that iO for circuits can be based on well-founded assumptions after the seminal work of Jain, Lin, and Sahai [JLS21].

Theorem 2 (Constant Rate, Falsifiable Assumptions, using CRS (Informal)). *Assuming the existence of indistinguishability obfuscation for circuits (and other standard assumptions), there exists a constant-rate compiler in the (long) CRS model that transforms any two-party protocol for a function f into a secure protocol for f .*

A sequence of results, starting from the work of Jain and Jin [JJ22], show that input-succinct iOM can be constructed from falsifiable assumptions *without* a long CRS for special (but still quite general and useful) classes of Turing machines [JJ22, MDS25, JJMP25] that admit a “PV proof” of equivalence [Coo75]. Further progress made in this direction, together with our results, may yield a constant-rate compiler under falsifiable assumptions in the plain model.

Finally, both the short and the long types of CRS we have mentioned in the results above, are only required to be random (as opposed to structured) strings. Thus, it is possible to get rid of them using a random oracle.

Theorem 3 (Constant Rate in ROM (Informal)). *Assuming the existence of indistinguishability obfuscation for circuits (and other standard assumptions), there is a constant rate compiler in the Random Oracle Model to transform any two-party protocol for a function f into a secure protocol for f .*

1.2 Other Related Work

Our work as well as much of the prior work we have mentioned so far, are most useful when considering functions that admit plain protocols with *sublinear* communication. In many situations though the input of one of the parties may be quite short, so the optimal plain protocol in such cases is for that party to simply send its input to the other. There are many interesting questions that arise in this setting though they are not directly relevant to the model we study since the communication of the secure protocol here can depend on the input length (but not on the size of the circuit being computed). FHE can be used to secure such protocols trivially in a rate-1 manner [Gen09] and these are sometimes referred to as “Alice optimized” protocols. Quach, Wee, and Wichs [QWW18], building upon the work of Cho *et al.* [CDG⁺17], introduce protocols with similar communication guarantees but they are “Bob optimized” (meaning that “Bob” does all the work instead of “Alice”). The recent work of Ciampi, Misra, Ostrovsky and Shah [CMOS25] shows that such communication-efficient secure transformations can be performed using only black-box access to cryptographic primitives (thus freeing $\tilde{c}$ from the “size” of these primitives). Badrinarayanan *et al.* [BFK⁺19] use their input-succinct TM obfuscator to achieve “compact” MPC in the (programmable) Random Oracle model, wherein $\tilde{c}$ grows with the inputs but avoids the output-length dependence inherent in the plain and CRS models.

2 Technical Overview

In this technical overview we present the main ideas behind our compiler and its security proof. Our first attempt will have very poor rate, and we will be unable to prove its security due to an unacceptable security loss caused by an exponential number of hybrids. Our second attempt introduces our new “tree-pruning” technique to reduce the number of hybrids, but the rate will still be too low. In the third attempt, we incorporate statistically binding hash techniques to bring the rate to constant. We conclude our overview with a brief discussion on achieving malicious security.

2.1 First Attempt: Obfuscate

Let $\pi = \langle A, B \rangle$ be an insecure two-party protocol for computing some deterministic function f where parties communicate one bit at a time over L total rounds. It is easy to see that any interactive protocol with arbitrary message lengths can be transformed into such a bit-by-bit protocol while at most doubling the communication complexity. Further assume that the output length is constant. We discuss how to remove these restrictions (*i.e.*, deterministic f , single-bit messages, short outputs) in [Sec. 8](#). We assume for simplicity that $|x_A| = |x_B|$ (as is often standard in literature and w.l.o.g. since we can pad appropriately).

We describe our first attempt at a semi-honest protocol between parties P_A and P_B with respective inputs x_A and x_B , and security parameter λ . We make use of an input-succinct TM obfuscator iOM ([Def. 2](#)), and a puncturable pseudorandom function $\text{PPRF} = (\text{KGen}, \text{F}, \text{Punc})$. The protocol is as follows:

1. **Setup Phase:** P_A and P_B participate in a protocol Π_{setup} securely computing the following setup functionality.

$f_{\text{setup}}(x_A, x_B) :$

- (a) Secret share $k \leftarrow \text{KGen}(1^\lambda)$ as $k = [k]_A \oplus [k]_B$.
- (b) $\text{obNext}^A \leftarrow \text{iOM}(1^\lambda, \text{Next}^A)$ and $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}^B)$, where Next^B is described below in [Algo. 2.1](#) (Next^A is defined analogously).
- (c) Output $([k]_A, \text{obNext}^A)$ to P_A and $([k]_B, \text{obNext}^B)$ to P_B .

Algorithm 2.1: Next^B // First Attempt

Inputs: Local input x , encrypted transcript $\tilde{\text{tr}}^A = (\tilde{m}_1^A, \dots, \tilde{m}_\ell^A)$.

Constants: Machine descriptions B, F ; PPRF key k ; honest local input x_B ; some short value τ_0 correlated with x_A and x_B (discussed shortly).

If $x \neq x_B$: output $\perp$.

For $i \in [\ell]$:

$m_i^A \leftarrow \tilde{m}_i^A \oplus \text{F}_k(\tau_0, \tilde{m}_1^A, \tilde{m}_1^B, \dots, \tilde{m}_{i-1}^A, \tilde{m}_{i-1}^B)$ // decrypt

$m_i^B \leftarrow B(x, (m_1^A, \dots, m_i^A))$ // compute next message

$\tilde{m}_i^B \leftarrow m_i^B \oplus \text{F}_k(\tau_0, \tilde{m}_1^A, \tilde{m}_1^B, \dots, \tilde{m}_{i-1}^B, \tilde{m}_{i-1}^A)$ // encrypt

Output $\tilde{m}_\ell^B$

2. **Encrypted Protocol Phase:** P_A and P_B compute and send each other messages using their obfuscations to produce an encrypted transcript $\tilde{\text{tr}} = (\tilde{m}_1^A, \tilde{m}_1^B, \dots, \tilde{m}_L^A, \tilde{m}_L^B)$, where L is the number of rounds in the insecure protocol $\langle A, B \rangle$. If at any point a party receives $\perp$ from their obfuscation, that party should abort.
3. **Output Decryption Phase:** Let $y_A, y_B = f(x_A, x_B)$. We assume for simplicity parties explicitly send each other their (short) outputs in $\langle A, B \rangle$, so the final L_{out} rounds contain the encrypted outputs $\tilde{y}_A, \tilde{y}_B$. P_A and P_B obtain their outputs by participating in a secure protocol Π_{dec} for the following output decryption functionality.

$f_{\text{dec}}([k]_A, [k]_B) :$

- (a) Reconstruct k from $[k]_A, [k]_B$.

- (b) Use k, τ_0 to decrypt $\tilde{y}_A, \tilde{y}_B$ as y_A, y_B from last L_{out} bits of $\tilde{\text{tr}}$.
- (c) Output y_A to P_A and y_B to P_B .

Some quick remarks on the above protocol are in order. We chose to hardcode the honest input x_B as protection against trivial attacks where an adversary queries its obfuscation on $x \neq x_B$ in order to learn information on x_A ³. Of course, hardcoding x_B results in large obfuscation and therefore poor rate in the malicious setting due to [HW15], but we will see later how to fix this with hashing. Similarly, the F input was chosen to be “local” to the current input and execution in order to prevent trivial attacks where an adversary uses F values to learn information on plaintext protocol messages in the current or subsequent executions. In the full construction [Prot. 6.1](#), parties communicate several objects prior to Π_{setup} , some of which are correlated with x_A, x_B . A subset of these objects forms the “preamble” τ_0 hardcoded in obfuscation.

Now let us see why we are unable to prove security, even in the semi-honest setting. Towards this goal, consider the simulator Sim^B for semi-honest P_B^* . Intuitively, it should look something like the following:

$\text{Sim}^B(x_B, y_B) :$

1. Simulate the setup phase to obtain simulated transcript $\text{simtr}_{\text{setup}}^A$.
2. Generate $\text{simtr}^A = (\tilde{m}_1^A, \dots, \tilde{m}_L^A)$ as L random values.
3. Simulate the output decryption phase to obtain simulated transcript $\text{simtr}_{\text{dec}}^A$.
4. Output $(x_B, \text{simtr}_{\text{setup}}^A, \text{simtr}^A, \text{simtr}_{\text{dec}}^A)$ as the simulated view of P_B^* .

Direct Approaches Fail. Given this simulator, what should our hybrids look like? We can invoke the simulators $\text{Sim}_{\text{setup}}, \text{Sim}_{\text{dec}}$ for the sub-protocols straightaway, so it would appear that the challenge lies in changing the $\{\tilde{m}_i^A\}$ from real to random. Let us focus on changing $\tilde{m}_1^A$ from real (*i.e.*, $m_1^A \oplus F_k(\tau_0)$) to random. Obviously we cannot directly switch $F_k(\tau_0)$ to random and reduce to pseudorandomness of F because the reduction needs to have a key for F in order to faithfully prepare the hybrids.

Instead, we could turn to standard $i\mathcal{OM}$ and puncturing techniques. For example, we could try hardcoding $r^* \leftarrow F_k(\tau_0)$ in obfuscation (*i.e.*, Next^B) and modifying its code to use hardcoded r^* to decrypt $\tilde{m}_1^A$, then puncture k at τ_0 and hardcode the punctured key k^* instead of k . Since the functionality of obfuscation is unchanged, this change is indistinguishable by the security of $i\mathcal{OM}$. We could then switch r^* to uniformly random; indistinguishability follows by the pseudorandomness of PPRF at punctured points. At this point though, it is unclear how to remove m_1^A from $\tilde{m}_1^A = m_1^A \oplus r^*$, since this value is correlated with the r^* hardcoded in Next^B . For example, consider a hybrid identical to the last one (where r^* is uniform), but where the hybrid simulator computes $\tilde{m}_1^A = u$ for some fresh uniform u independent of r^* . Then a distinguisher might be able to distinguish between the two hybrids by inspecting the r^* in obNext^B (recall the security of $i\mathcal{OM}$ does not preclude such an inspection) and XORing it against $\tilde{m}_1^A$. In the old hybrid the XOR’d value is distributed according to m_1^A , while in the new hybrid it is distributed uniformly.

Alternatively, we could try to hardcode $\tilde{m}_1^A = m_1^A \oplus r^*, \tilde{m}_1^B = m_1^B \oplus F_k(\tau_0, \tilde{m}_1^A)$ in obfuscation. Modify the code to check whether the incoming $\tilde{m}_1^A$ equals the hardcoded $\tilde{m}_1^A$; if equality holds, output hardcoded $\tilde{m}_1^B$; else, compute $\tilde{m}_1^B$ as before. Functionality of obfuscation is unchanged so the change is indistinguishable. With the aim of switching r^* to random, we would like to puncture k at τ_0 next. But if we do so, we lose the ability to decrypt $\tilde{m}_1^A$. If we hardcode r^* to enable decryption, we end up in the same situation as above where the incoming $\tilde{m}_1^A$ is correlated with r^* .

Strategy: Make Outgoing Messages Pseudorandom. It appears that, in order to change $\tilde{m}_1^A$ to random, we must ensure its value is uncorrelated with anything else in obfuscation. Even without hardcoding r^* in obfuscation, however, $\tilde{m}_1^A$ is correlated with quite a few things in obfuscation. Namely, every outgoing message $\tilde{m}_i^B$ depends on the plaintext value m_1^A via the plaintext value $m_i^B = B(x_B, m_1^A, \dots, m_i^A)$. This suggests a new approach where we first modify *all* the outgoing $\tilde{m}_i^B$ by removing information about the plaintext m_i^B . More

³ For example, an adversarial P_B^* knows that x_B and x'_B result in the same output if P_A has x_A , but different outputs if P_A has x'_A . Without any validation of local input, P_B^* could use obNext^B to distinguish P_A ’s inputs.

specifically, we want to introduce a new key k_{dummy} , and compute $\tilde{m}_i^B \leftarrow F_{k_{dummy}}(\tau_0, \tilde{m}_1^A, \tilde{m}_1^B, \dots, \tilde{m}_{\ell-1}^B, \tilde{m}_\ell^A)$ as “dummy” pseudorandomness *for every possible input to obfuscation that does not result in $\perp$* . Intuitively, this protects against a semi-honest adversary that locally queries its obfuscation on specially crafted transcripts in hopes of learning information on x_A .

If we could somehow do this, then since k is unused by Next^B , perhaps we could remove it from Next^B and directly change all the $(\tilde{m}_i^A)_{i \in [L]}$ to random by the pseudorandomness of F . Unfortunately, changing obfuscation’s output to dummy turns out not to be so simple.

Too Many Hybrids. Even if we could somehow change $\text{Next}^B(x_B, (\tilde{m}_i^A)_{i \in [\ell]})$ to dummy with a sequence of $O(1)$ hybrids for any $(\tilde{m}_i^A)_{i \in [\ell]}$, $\ell \leq L$, repeating this sequence of hybrids for all $2^{L+1} - 2$ (non-aborting) inputs to $\text{Next}^B(x_B, \cdot)$ incurs a non-negligible security loss due to an exponential number of hybrids.

It is conceptually useful to frame the foregoing issue in terms of a fully balanced binary tree with $2^{L+1} - 2$ edges corresponding to the input-output behavior of $\text{Next}^B(x_B, \cdot)$ with respect to each possible incoming transcript. Edges represent inputted messages $\tilde{m}_i^A$ (e.g., a left edge corresponds to $\tilde{m}_i^A = 0$, right edge corresponds to $\tilde{m}_i^A = 1$). Nodes represent outputs $\tilde{m}_i^B$, with the value of a node determined by the sequence of edges from root to that node. That is, given a sequence of ℓ edges $(\tilde{m}_i^A)_{i \in [\ell]}$ from the root to a node, we assign that node the value $\text{Next}^B(x_B, (\tilde{m}_i^A)_{i \in [\ell]})$ ⁴. Circumventing the need to individually change every node in the tree to dummy⁵ appears to require a new idea; we describe ours now.

2.2 Tree Pruning with DPSig

The DPSig Primitive. Our insight is to “prune” this tree into a number of much smaller sub-trees through the use of our “dual-mode programmable one-time signature” scheme. At a high level, this primitive allows us to generate key pairs in a dual mode, indistinguishable from the normal mode, where the verification key is programmed to only accept signatures on specific strings⁶. Our construction DPSig (see Def. 4) is quite simple: it is essentially a Lamport one-time signature with a pseudorandom generator PRG instead of a one-way function. To program a verification key on a string m , we replace all the entries in the Lamport verification key not corresponding to m with uniformly random strings instead of pseudorandom. The programmed key is indistinguishable by the pseudorandomness of PRG, and it will not accept signatures on any $m' \neq m$ except with negligible probability because uniformly random strings are not in the image of PRG except with negligible probability. See Sec. 5 for a formal construction and proof.

Pruning with DPSig. Armed with DPSig, we modify our compiler as follows. We introduce a parameter $\eta = \text{poly}(\lambda)$. The L rounds of the protocol’s main phase are divided into $N := L/\eta$ full “segments” such that each party sends η messages per segment, followed by a DPSig one-time signature on these η messages. The obfuscated machines are responsible for verifying the signatures on each segment, and they output $\perp$ if verification fails for a signature on any segment. The DPSig key pairs $\{(\text{sk}, \text{vk})_j^A\}_{j \in [N]}$ and $\{(\text{sk}, \text{vk})_j^B\}_{j \in [N]}$ are generated by P_A and P_B respectively before execution of Π_{setup} , where Π_{setup} now ensures that the $\{\text{vk}_j^A\}_{j \in [N]}$ are hardcoded in Next^B , and vice versa for Next^A .

⁴ We could define the value of the root node as $\text{Next}^B(x_B, \perp)$, but this is largely irrelevant since we assume A speaks first and we simulate an adversarial P_B^* .

⁵ It is tempting to try changing multiple nodes at once by puncturing the key at multiple points. Unfortunately, all PPRF (or more generally, constrained PRF [BW13]) constructions we are aware of require the punctured (or constrained) key to grow with the number of punctured points (or the size of the circuit describing the “accepting set”) Because punctured keys are hardcoded in Next^B , which must be padded to the size of the largest Next^B throughout the hybrids, this approach hurts rate.

⁶ This is reminiscent of the “splittable signatures” defined and constructed in [KLW15], also used in [BFK⁺19]. Our notion, however, only supports something like their “one-message” verification key functionality. Their construction assumes iO for circuits, a PPRF, and a PRG, while ours only assumes a PRG.

The key observation is that if verification fails on a certain j -th segment, $j \in \{1, \dots, N\}$, then the output of obfuscation is $\perp$ on *any* input with j -th segment matching the failing segment. In the security proof against P_B^* , we can program verification keys to only accept signatures on segments sent by the simulator. The intuition here is to “prune” the large $O(2^L)$ -sized tree into N smaller sub-trees stacked on top of each other, each of height η . Very informally, we hope to begin the security proof by considering the first sub-tree, which is just the large tree truncated to the first η layers. After switching all nodes in the sub-tree to dummy values with $O(2^\eta)$ hybrids, with $O(\eta)$ hybrids we change the incoming $\tilde{m}_1^A, \dots, \tilde{m}_\eta^A$ to random⁷. This random segment defines a path through the sub-tree from root to a leaf $\tilde{m}_\eta^B$. Next we program vk_1^A on this segment; this has the effect of switching the values of all the nodes below layer η to $\perp$, except for those nodes with $\tilde{m}_\eta^B$ as an ancestor. Next we consider the second sub-tree of height η rooted at $\tilde{m}_\eta^B$. Change the nodes in this sub-tree to dummy, then change the incoming message segment to random, and program vk_2^A on the random segment. Continuing in this fashion for all N segments, we end up with the aforementioned stack of N precariously balanced sub-trees, threaded through by a strand corresponding to the random simulated $\{\tilde{m}_i^A\}$. See Fig. 1.

Of course, with this approach we are still incurring a security loss exponential in η , so we choose η carefully and complexity leverage appropriately in the final construction.

2.3 Improving Rate with Statistically Binding Hashes

The rate of our construction is still quite poor for a number of reasons. Notably, x_B is hardcoded in obfuscation, and each of the N verification keys and signatures sent by each party has size $\Omega(\eta)$. We address these rate issues by incorporating statistically binding hash techniques into the construction.

Next-Message Binding. Let us focus on replacing the hardcoded x_B with a hash $H(x_B)$ (and Next^B now outputs $\perp$ if $H(x) \neq H(x_B)$). To see why collision resistant hashing is insufficient, consider the following example where we try to change the lexicographically first node in the first sub-tree $\tilde{m}_1^B = \text{Next}^B(x_B, \tilde{m}_1^A = 0)$ to dummy⁸. We modify Next^B by replacing k with $k^* \leftarrow \text{Punc}(k, (\tau_0, 0))$, hardcoding $\tilde{m}_1^B$, and using the hardcoded $\tilde{m}_1^B$ whenever the inputted $\tilde{m}_1^A = 0$. For hardcoded x_B , this modification is indistinguishable since the behavior of Next^B is clearly unchanged. But with hardcoded $H(x_B)$, there exist many $x^* \neq x_B$ such that $H(x^*) = H(x_B)$, and (crucially) it may be that $\text{Next}^B(x^*, 0) \neq \text{Next}^B(x_B, 0)$. That is, the behavior of Next^B changes at all such “bad” colliding points x^* . It may be tempting to argue that this modification is indistinguishable because colliding points are hard to find, but recall that doing so implies a non-falsifiable diO-type assumption. In terms of our tree analogy, replacing x_B with $H(x_B)$ introduces a tree rooted at every possible x , and now when we modify Next^B we must be careful about what happens at *every* tree.

Our solution is to ensure that $H(x)$ statistically binds on the “next message” with respect to x and some fixed $(\tilde{m}_i^A)_{i \in [\ell]}$ identifying the target node (a.k.a next message) in the tree. That is, whenever $H(x^*) = H(x_B)$, it is guaranteed that $\text{Next}^B(x^*, (\tilde{m}_i^A)_{i \in [\ell]}) = \text{Next}^B(x_B, (\tilde{m}_i^A)_{i \in [\ell]})$. Actually, it suffices to guarantee that $B(x^*, (m_i^A)_{i \in [\ell]}) = B(x_B, (m_i^A)_{i \in [\ell]})$, where $(m_i^A)_{i \in [\ell]}$ corresponds to the decryption of $(\tilde{m}_i^A)_{i \in [\ell]}$. This is accomplished through the use of a *function-binding hash* (see Def. 3) that statistically binds on an efficiently computable function of the input, while computationally hiding the function it is binding on (with respect to a certain machine class). Specifically, we would like our hash to bind on the function computing $B(\cdot, (m_i^A)_{i \in [\ell]})$. Unfortunately, this function description leads to a hash key hardcoded in Next^B that grows with L (and we would have to deal with the fact that we hardcoded information correlated with the $(m_i^A)_{i \in [\ell]}$). The function we bind on, denoted $\mathcal{F}_B$, thus accepts two inputs: one intended for a local input x and the other for the target path $(\tilde{m}_i^A)_{i \in [\ell]}$, and uses these inputs to compute B ’s next message. We denote the resulting hash key (and hash function), which will be sent by P_A , as H_{NM}^A (to be clear, H_{NM}^A binds on $\mathcal{F}_B$).

⁷ We defer details on how to change nodes and incoming messages to the full proof. Suffice it to say for now that both of these are accomplished through standard (though subtle) sequences of iOM + puncturing + statistically binding hashing techniques.

⁸ Technically, Next^B also takes a list of signatures $(\sigma_i^A)_{i \in [\ell/\eta]}$ as input. We suppress these here for clarity.

Using a Common Random String. Of course, it is now unclear where $H_{\text{NM}}^A(\cdot, \cdot)$ obtains its second input from, keeping in mind that (a) the resulting hash value h_B is hardcoded in Next^B during Π_{setup} ; (b) Next^B needs to recompute the hash and compare it with h_B on every input; and (c) this second input should correspond to the target path $(\tilde{m}_i^A)_{i \in [\ell]}$ for the current hybrid (or junk for the real execution). We solve this issue with introduction of a common random string crs . In Π_{setup} , we hardcode $h_{\text{NM}}^B = H_{\text{NM}}^A(x_B, \text{crs})$ in Next^B . Next^B now takes an additional input intended for crs , and outputs $\perp$ if $H_{\text{NM}}^A(x, \text{crs}) \neq h_B$. In the hybrids, we can switch crs to encryptions of the $(m_i^A)_{i \in [L]}$ under a key k_{crs} , and have $\mathcal{F}_B$ decrypt the first ℓ entries of crs with k_{crs} to $(m_i^A)_{i \in [\ell]}$ (both ℓ and k_{crs} hardcoded in $\mathcal{F}_B$) and output the next message $B(x, (m_i^A)_{i \in [\ell]})$ (see [Algo. 6.1](#))

While we just motivated the introduction of crs via the next-message binding hash, this crs ends up being useful in other parts of the proof. In particular, when we change an incoming message to (pseudo)random, we need to preserve the distribution of subsequent messages with respect to the original incoming messages. Thankfully, crs already has encryptions of the original messages, so Next^B can try decrypting these from crs and use them to compute any outgoing messages that have not already been changed to dummy. The CRS does not have to be trusted: parties can send sufficiently long uniform strings, say $\text{crs}^A, \text{crs}^B$. These are for their own protection, so we need not sample them via coin-tossing.

SSB Hashing. Beyond x_B , there remain several objects we still wish to replace with hashes. For all these, however, the standard notion of SSB hashing suffices, as we only need the hash function to statistically bind on indices of its input in order to preserve the behavior of Next^B . We briefly list the rest of the hashing modifications, and leave details (*e.g.*, what the hash binds on) for the full construction. Instead of signing full message segments, parties now send each other signatures on hashes of the message segments. The hash keys for these segment hashes are hardcoded in Next^B so Next^B can hash incoming segments and verify signatures. Each party also hashes all of its verification keys. The appropriate hash and hash key is hardcoded in Next^B , which now takes in all verification keys as input and aborts if their hash does not match the hardcoded value. Similarly, the crs itself is hashed, the hash value and hash key are hardcoded in Next^B , and Next^B aborts if the hash of its alleged crs input does not match the hardcoded value. Finally, we hash the input to the PPRF within Next^B since previously this input grew with L , and so punctured keys would also grow with L . As we calculate in [Sec. 6.2](#), our protocol is now constant-rate.

2.4 Malicious Security.

Towards achieving malicious security, Naor and Nissim [\[NN01\]](#) pointed out that the canonical GMW semi-honest to malicious compiler [\[GMW87\]](#) does not preserve the communication complexity of the semi-honest protocol. This is in large part due to the use of a statistically binding commitment scheme for the input-commitment phase of GMW. Naor and Nissim instead opt for a computationally binding extractable commitment scheme with succinct $\text{poly}(\lambda)$ -sized communication independent of the committed value. They also replace the GMW zero-knowledge proofs with a succinct (polylogarithmic communication in the witness) zero-knowledge argument of knowledge. Both primitives are based off ideas from Kilian [\[Kil92, Kil95, CDG⁺24\]](#).

We adopt essentially the same primitives and approach, with one notable modification. For rate reasons we cannot afford to have parties execute a proof of honest behavior after each message. Instead, parties prove they behaved honestly in each segment. This way, the $N \cdot \text{poly}(\lambda)$ communication overhead of proof execution matches that of signatures, which is forced to $o(\text{cc})$ by appropriate choice of η .

3 Preliminaries

3.1 Shortlex

We recall the shortlex order on binary strings. Let $\text{int} : \{0, 1\}^* \rightarrow \mathbb{N}$ be the canonical map from a binary string to its unsigned integer representation, and let $\text{str} : \mathbb{N} \rightarrow \{0, 1\}^*$ be its inverse. The “shortlex” order on strings is induced by the function $\text{lex} : \{0, 1\}^* \rightarrow \mathbb{N}$ given by $z \mapsto 2^{|z|} + \text{int}(z) - 1$, with inverse $y \mapsto$

$\text{str}(y - \text{lex}(0^{\lfloor \log(y+1) \rfloor}))$). In plain terms, the shortlex order of binary strings is given by the familiar sequence $0, 1, 00, 01, 10, 11, 000, 001, 010, \dots$. Given binary strings a and b , $a < b$ means a comes before b in shortlex order.

3.2 Communication model.

We denote an interactive protocol Π between parties A and B as a pair $\langle A, B \rangle$. Parties A and B enter the protocol with initial local inputs x_A and x_B , respectively. We assume A “speaks first” by sending B a message m_1^A . B responds with a message m_1^B . The two parties exchange messages in alternation for L rounds, producing a transcript $(m_1^A, m_1^B, \dots, m_L^A, m_L^B)$ (we assume B speaks last), at which point they output their respective final outputs y_A and y_B . We can identify parties A and B with their associated next-message Turing machines. That is, B can be thought of as a machine mapping input x_B and a (partial) incoming transcript $(m_1^A, \dots, m_\ell^A)$ of length $\ell \leq L$ to an outgoing next message m_ℓ^B . We assume the number of rounds L and the length of each message can be deduced from Π , $|x_A|$, and $|x_B|$.

3.3 Subexponential Indistinguishability.

Because our work relies on subexponential security assumptions, we borrow a more general definition of computational (t, ε) -indistinguishability (see Def. 1) from the work of [JLS21]. Unlike [JLS21] however, we opt for the interpretation of [WW24] in saying two ensembles are subexponentially indistinguishable if they are $(1, 2^{-\lambda^\varepsilon})$ -indistinguishable (*i.e.*, PPT adversary with subexponential distinguishing advantage) for some constant $\varepsilon \in (0, 1)$. Because we only consider PPT adversaries, we may drop the $t = 1$ and just refer to ε -indistinguishability. We extend this notion to say a primitive is ε -secure if its security definition relies on ε -indistinguishability of two ensembles (*e.g.*, Def. 6, Def. 2). If these ensembles are subexponentially indistinguishable, then we say the primitive is subexponentially secure.

Definition 1 ((t, ε)-indistinguishability [JLS21]). We say that two ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ are (t, ε) -indistinguishable where $t : \mathbb{N} \rightarrow \mathbb{N}$ and $\varepsilon : \mathbb{N} \rightarrow [0, 1]$ if for every non-negative polynomial $\text{poly}(\cdot, \cdot)$ and any adversary $\mathcal{A}$ running in time bounded by $t(\lambda)\text{poly}(\lambda)$ it holds that: for every sufficiently large $\lambda \in \mathbb{N}$,

$$\left| \Pr_{x \leftarrow \mathcal{X}_\lambda} [\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y \leftarrow \mathcal{Y}_\lambda} [\mathcal{A}(1^\lambda, y) = 1] \right| \leq \varepsilon(\lambda)$$

3.4 Input-Succinct Turing Machine Obfuscation

The following essential definition is adapted from [JJMP25]. See Sec. 1.1 above for discussion.

Definition 2 (Input-succinct Turing Machine Obfuscation). An input-succinct indistinguishability obfuscator for a class of polynomial-time Turing machines $\{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of a PPT algorithm iOM with the following syntax:

- $\text{iOM}(1^\lambda, M) \rightarrow \text{obM}$: On input the security parameter λ , and a Turing machine description $M \in \mathcal{M}_\lambda$, output an obfuscated Turing machine description obM .

and satisfying the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$, for all $M \in \mathcal{M}_\lambda$, and for all inputs x ,

$$\Pr [\text{obM}(x) = M(x) \mid \text{obM} \leftarrow \text{iOM}(1^\lambda, M)] = 1$$

- **Security:** For any $\lambda \in \mathbb{N}$, and any pair of machines $M_0, M_1 \in \mathcal{M}_\lambda$ of the same size such that $M_0(x) = M_1(x)$ for all inputs x , the following distribution ensembles are computationally indistinguishable:

$$\{\text{iOM}(1^\lambda, M_0)\}_\lambda \stackrel{c}{\approx} \{\text{iOM}(1^\lambda, M_1)\}_\lambda$$

- **Succinctness and Efficiency:** If $\text{obM} \leftarrow \text{iOM}(1^\lambda, M)$, we require $|\text{obM}| = \text{poly}(\lambda, |M|)$, and we require that obM run in polynomial time.

4 Function Binding Hash

We consider a variant of somewhere statistically binding hashes that statistically bind to a function of the input, rather than to a bit of the input. Our definition and construction are actually quite simple as we do not require properties like local opening proofs or succinctness as in other works [BBK⁺23, FWW23]. This relaxation allows us to capture more general function families, whereas [BBK⁺23] only captured the family of “bit-fixing” predicates, and [FWW23] only captured “disjunctions of block functions”. In addition, our function binding hashes work for Turing machine representations, rather than circuits.

Definition 3 (Function Binding Hash). *A function binding hash for a machine class $\{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple of PPT algorithms $\mathcal{H} = (\text{Gen}, \text{Hash})$ with the following syntax:*

- $\text{Gen}(1^\lambda, f, \ell_{in}, t) \mapsto (\text{hk}, \text{td})$: *On input the security parameter 1^λ , a Turing machine description $f \in \mathcal{M}_\lambda$, input-length bound ℓ_{in} , and an alleged runtime bound t for f on length ℓ_{in} inputs, output a hash key hk and trapdoor td .*
- $\text{Hash}(\text{hk}, x) \mapsto v$: *On input a hash key hk and a message $x \in \{0, 1\}^{\leq \ell_{in}}$ (where ℓ_{in} is parsed from hk), deterministically output a hash value v .*

satisfying the following properties:

- **Function Hiding:** *For any $\lambda \in \mathbb{N}$, for all poly-sized $\mathcal{A}$, for any two Turing machine descriptions $f_0, f_1 \in \mathcal{M}_\lambda$, for any $\ell_{in}, t = \text{poly}(\lambda)$, there exists a negligible function negl such that*

$$\Pr \left[\mathcal{A}(\text{hk}) = b \mid \begin{array}{l} b \leftarrow \{0, 1\} \\ (\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, f_b, \ell_{in}, t(\ell_{in})) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

- **Statistical Binding:** *For any $\lambda, \ell_{in} \in \mathbb{N}$, for any Turing machine description $f \in \mathcal{M}_\lambda$ such that f is computable in at most t steps on length ℓ_{in} inputs, and for any x, x' , there exists a negligible function negl such that*

$$\Pr \left[f(x) = f(x') \mid \begin{array}{l} (\text{hk}, -) \leftarrow \text{Gen}(1^\lambda, f, \ell_{in}, t) \\ \text{Hash}(\text{hk}, x) = \text{Hash}(\text{hk}, x') \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

Construction. We provide a construction of Def. 3 for any class $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ such that for any $f_0, f_1 \in \mathcal{M}_\lambda$, $|f_0| = |f_1|$, and for any x_0, x_1 , it holds that $|f_0(x_0)| = |f_1(x_1)|$. Our construction relies on a (subexponentially secure) fully homomorphic encryption scheme FHE Def. 12.

Our construction of a function-binding hash Def. 3 for $\mathcal{M}$ is as follows.

$\text{Gen}(1^\lambda, f, \ell_{in}, t)$:

- $(\text{pk}, \text{sk}) \leftarrow \text{FHE.KeyGen}(1^\lambda)$
- $\tilde{f} \leftarrow \text{FHE.Enc}(\text{pk}, f)$
- Output $(\text{hk} = (\text{pk}, \tilde{f}, \ell_{in}, t), \text{td} = \text{sk})$

$\text{Hash}(\text{hk}, x)$:

- Parse hk as $\text{pk}, \tilde{f}, \ell_{in}, t$. Abort if $|x| \neq \ell_{in}$.
- Let $U_{x,t}$ be a Turing machine that, given as input a machine description M , runs M on x for t steps. Apply the Pippenger-Fischer transformation [PF79] to $U_{x,t}$ to obtain a size $O(t \log t)$ circuit $C_{U_{x,t}}$ with identical functionality to $U_{x,t}$. Output $v \leftarrow \text{FHE.Eval}(\text{pk}, C_{U_{x,t}}, \tilde{f})$.

Proof. We prove function hiding by reduction to the semantic security of FHE. Suppose $\mathcal{A}_\mathcal{H}$ is any (non-uniform) adversary against the function hiding of $\mathcal{H}$. We specify an adversary $\mathcal{A}_{\text{FHE}}$ against the semantic security of FHE that works as follows. Let $(\text{sk}, \text{pk}) \leftarrow \text{FHE.KeyGen}(1^\lambda)$ be sampled by the challenger. On input 1^λ and pk , $\mathcal{A}_{\text{FHE}}$ computes $f_0, f_1, \ell_{in}, t \leftarrow \mathcal{A}_\mathcal{H}(1^\lambda)$ and forwards f_0, f_1 to the challenger. Notice $\mathcal{A}_{\text{FHE}}$

is responsible for ensuring $f_0, f_1 \in \mathcal{M}$; it can do so by computing the output length $\ell_{out} = |f_0(0)|$ (aborting if $\ell_{out} \neq |f_1(0)|$), and then modifying the descriptions of both machines so that they always pad/truncate their outputs to ℓ_{out} bits. The challenger chooses a uniform bit b , and encrypts $\tilde{f}_b \leftarrow \text{FHE.Enc}(\text{pk}, f_b)$. Upon receiving $\tilde{f}_b$, $\mathcal{A}_{\text{FHE}}$ prepares $\text{hk} = (\text{pk}, \tilde{f}_b, \ell_{in}, t)$ and outputs bit $b' \leftarrow \mathcal{A}_{\mathcal{H}}(\text{hk})$. See that $\mathcal{A}_{\text{FHE}}$ faithfully simulates the function hiding game for $\mathcal{A}_{\mathcal{H}}$, and the advantage of $\mathcal{A}_{\text{FHE}}$ equals that of $\mathcal{A}_{\mathcal{H}}$. Thus, the advantage of $\mathcal{A}_{\mathcal{H}}$ is negligible.

We prove statistical binding by the correctness of FHE. Let $\lambda, \ell_{in} \in \mathbb{N}$, and let $f \in \mathcal{M}$ computable in at most t steps on length ℓ_{in} inputs, let $(\text{hk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, f, \ell_{in}, t)$, and let x, x' be such that $h = \text{Hash}(\text{hk}, x) = \text{Hash}(\text{hk}, x') = h'$. By our construction, we have that $h = \text{FHE.Enc}(\text{pk}, f(x)) = \text{FHE.Enc}(\text{pk}, f(x')) = h'$, where pk is the FHE public key in hk . Let sk denote the corresponding FHE secret key in td . By correctness of FHE, $\text{FHE.Dec}(\text{sk}, h) = f(x) = f(x') = \text{FHE.Dec}(\text{sk}, h')$ with probability at least $1 - \text{negl}(\lambda)$, as desired. $\square$

5 Dual-Mode Programmable One-Time Signatures

Definition 4 (Dual-Mode Programmable One-Time Signature). A dual-mode programmable one-time signature DPSig for message space $\mathcal{M}$ is a tuple of algorithms $(\text{DPSig.Setup}, \text{DPSig.Sign}, \text{DPSig.Verify}, \text{DPSig.Prog})$ with the following syntax:

- $\text{DPSig.Setup}(1^\lambda) \mapsto (\text{sk}, \text{vk})$: On input the security parameter λ , generate signing key sk and verification key vk .
- $\text{DPSig.Sign}(\text{sk}, m) \mapsto \sigma$: On input a signing key sk and message m , generate signature σ .
- $\text{DPSig.Verify}(\text{vk}, m, \sigma) \mapsto b$: On input a verification key vk , message m , and signature σ , deterministically output a bit b representing accept/reject.
- $\text{DPSig.Prog}(1^\lambda, m) \mapsto (\text{sk}^*, \text{vk}^*)$: On input the security parameter λ and a message $m \in \mathcal{M}$, generate a signing key sk^* and a programmed verification key vk^* .

and satisfying the following properties:

- **Correctness:** For any $\lambda \in \mathbb{N}$ and for all $m \in \mathcal{M}$,

$$\Pr \left[\text{DPSig.Verify}(\text{vk}, m, \sigma) = 1 \mid \begin{array}{l} (\text{sk}, \text{vk}) \leftarrow \text{DPSig.Setup}(1^\lambda) \\ \sigma \leftarrow \text{DPSig.Sign}(\text{sk}, m) \end{array} \right] = 1$$

- **One-Time Unforgeability:** The scheme is unforgeable if the advantage of any PPT algorithm $\mathcal{A}$ in the following (standard) game is negligible in λ :

- $(\text{sk}, \text{vk}) \leftarrow \text{DPSig.Setup}(1^\lambda)$
- $m \leftarrow \mathcal{A}(\text{vk})$ // $\mathcal{A}$ submits a message to one-time signing oracle
- $(m^*, \sigma^*) \leftarrow \mathcal{A}(\text{vk}, \text{DPSig.Sign}(\text{sk}, m))$
- $\mathcal{A}$ wins if $\text{DPSig.Verify}(\text{vk}, m^*, \sigma^*) = 1$ and $m^* \neq m$

- **Mode Indistinguishability:** The following distribution ensembles (indexed by $m \in \mathcal{M}$ and $\lambda \in \mathbb{N}$) are computationally indistinguishable:

$$\{\text{vk}^*, \sigma^*\}_{m, \lambda} \stackrel{c}{\approx} \{\text{vk}, \sigma\}_{m, \lambda},$$

where $(\text{sk}, \text{vk}) \leftarrow \text{DPSig.Setup}(1^\lambda)$, $(\text{sk}^*, \text{vk}^*) \leftarrow \text{DPSig.Prog}(1^\lambda, m)$, $\sigma \leftarrow \text{DPSig.Sign}(\text{sk}, m)$, and $\sigma^* \leftarrow \text{DPSig.Sign}(\text{sk}^*, m)$.

- **Functionality of Programmed Verification Key:** If vk^* is programmed on m^* , then only signatures on m^* - and no other message - can be verified by vk^* with overwhelming probability. More formally, there

exists a negligible function negl such that for any $\lambda \in \mathbb{N}$, for all σ , and for all $m, m^* \in \mathcal{M}, m \neq m^*$:

$$\Pr \left[\text{DPSig.Verify}(\text{vk}^*, m^*, \sigma) = 1 \mid \begin{array}{l} (\text{sk}, \text{vk}^*) \leftarrow \text{DPSig.Prog}(1^\lambda, m^*) \\ \sigma \leftarrow \text{DPSig.Sign}(\text{sk}, m^*) \end{array} \right] = 1, \text{ and}$$

$$\Pr [\text{DPSig.Verify}(\text{vk}^*, m, \sigma) = 1 \mid (-, \text{vk}^*) \leftarrow \text{DPSig.Prog}(1^\lambda, m^*)] \leq \text{negl}(\lambda)$$

We provide a construction of dual-mode programmable one-time signatures for messages of length ℓ . Our construction only assumes the existence of a pseudorandom generator $\text{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^p$, where $p = p(\lambda) \geq 2\lambda$ is polynomial in λ . Let $\text{DPSig} = (\text{Setup}, \text{Sign}, \text{Verify}, \text{Prog})$ be as follows:

- $\text{Setup}(1^\lambda) \rightarrow (\text{sk}, \text{vk})$:
 - Sample $\{x_{i,j}\} \in (\{0, 1\}^\lambda)^{2 \times \ell}$, where each $x_{i,j} \xleftarrow{\$} \{0, 1\}^\lambda$ for $i \in \{0, 1\}, 1 \leq j \leq \ell$.
 - Compute $\{y_{i,j}\} \in (\{0, 1\}^p)^{2 \times \ell}$, where each $y_{i,j} \leftarrow \text{PRG}(x_{i,j})$ for $i \in \{0, 1\}, 1 \leq j \leq \ell$.
 - Output $\text{sk} = \{x_{i,j}\}, \text{vk} = \{y_{i,j}\}$.
- $\text{Sign}(\text{sk}, m) \rightarrow \sigma$:
 - Parse sk as $\{x_{i,j}\} \in (\{0, 1\}^\lambda)^{2 \times \ell}$, and m as $(m_1, \dots, m_\ell) \in \{0, 1\}^\ell$.
 - For each $j \in [\ell]$, let $\sigma_j = x_{0,j}$ if $m_j = 0$, else $\sigma_j = x_{1,j}$.
 - Output $\sigma = (\sigma_1, \dots, \sigma_\ell) \in (\{0, 1\}^\lambda)^\ell$.
- $\text{Verify}(\text{vk}, m \in \{0, 1\}^\ell, \sigma) \rightarrow b$:
 - Parse vk as $\{y_{i,j}\} \in (\{0, 1\}^p)^{2 \times \ell}$, m as $(m_1, \dots, m_\ell) \in \{0, 1\}^\ell$, and σ as $(\sigma_1, \dots, \sigma_\ell) \in (\{0, 1\}^\lambda)^\ell$.
 - If $y_{m_j,j} \neq \text{PRG}(\sigma_j)$ for any $j \in [\ell]$, output 0. Else, output 1.
- $\text{Prog}(1^\lambda, m) \mapsto (\text{sk}, \text{vk}^*)$:
 - Parse m as $(m_1, \dots, m_\ell) \in \{0, 1\}^\ell$.
 - Sample $\{x_{i,j}\} \in (\{0, 1\}^\lambda)^{2 \times \ell}$, where each $x_{i,j} \xleftarrow{\$} \{0, 1\}^\lambda$ for $i \in \{0, 1\}, 1 \leq j \leq \ell$.
 - Compute $\{y_{i,j}\} \in (\{0, 1\}^p)^{2 \times \ell}$, where $y_{m_j,j} \leftarrow \text{PRG}(x_{m_j,j})$; $y_{1-m_j,j} \xleftarrow{\$} \{0, 1\}^p$ for $1 \leq j \leq \ell$.

Proof. Correctness and one-time unforgeability follow from the properties of the Lamport OTS.

Mode indistinguishability. We prove mode indistinguishability with a simple hybrid argument. Let $\lambda \in \mathbb{N}$ and $m \in \{0, 1\}^\ell$ be arbitrary, and consider the sequence of hybrids $H_{\lambda,m}^{(0)}, \dots, H_{\lambda,m}^{(\ell)}$, defined as follows:

$H_{\lambda,m}^{(k)}$:

- Set $x_{m_j,j} \xleftarrow{\$} \{0, 1\}^\lambda$ and $y_{m_j,j} \leftarrow \text{PRG}(x_{m_j,j})$ for $j \in \{1, \dots, \ell\}$.
- Set $y_{1-m_j,j} \xleftarrow{\$} \{0, 1\}^p$ for $j \in \{1, \dots, k\}$.
- Set $y_{1-m_j,j} \leftarrow \text{PRG}(s_j \xleftarrow{\$} \{0, 1\}^\lambda)$ for $j \in \{k+1, \dots, \ell\}$.
- Output $(\text{vk} = \{y_{i,j}\}, \sigma = \{x_{m_j,j}\})$

First, see that $H_{\lambda,m}^{(0)}$ corresponds to the unprogrammed distribution $\{\text{vk}, \sigma\}_{\lambda,m}$, while $H_{\lambda,m}^{(\ell)}$ corresponds to the distribution $\{\text{vk}^*, \sigma\}_{\lambda,m}$, where vk^* is programmed on m . It suffices to prove $H_{\lambda,m}^{(k)} \stackrel{c}{\approx} H_{\lambda,m}^{(k+1)}$ for any $k \in \{0, \dots, \ell-1\}$ by reduction to the security of PRG. Consider an adversary $\mathcal{A}_{\text{PRG}}$ who receives as input 1^λ , and a string $w \in \{0, 1\}^{p(\lambda)}$ that is either uniformly random, or in the image of PRG. $\mathcal{A}_{\text{PRG}}$ prepares (vk, σ) exactly as in the description of $H_{\lambda,m}^{(k)}$ (for any non-uniformly chosen m), except he sets $y_{1-m_k,k} \leftarrow w$. $\mathcal{A}_{\text{PRG}}$ computes $b \leftarrow \mathcal{A}_H(\lambda, m, (\text{vk}, \sigma))$ (where $\mathcal{A}_H$ is any PPT adversary attempting to distinguish between $H_{\lambda,m}^{(k)}$

and $H_{\lambda,m}^{(k+1)}$ and outputs b to the PRG challenger. See that if w is pseudorandom, then (vk, σ) is distributed according to $H_{\lambda,m}^{(k)}$, and if w is uniform, then (vk, σ) is distributed according to $H_{\lambda,m}^{(k+1)}$. Thus, the advantage of $\mathcal{A}_{\text{PRG}}$ in breaking the security of PRG is equal to the advantage of $\mathcal{A}_H$ in distinguishing between $H_{\lambda,m}^{(k)}$ and $H_{\lambda,m}^{(k+1)}$.

Functionality of Programmed Programmed Verification Key. Suppose $(\text{sk} = \{x_{i,j}\}, \text{vk}^* = \{y_{i,j}\}) \leftarrow \text{Prog}(1^\lambda, m^*)$, and $\sigma^* \leftarrow \text{Sign}(\text{sk}, m^*)$. Because $y_{m_j,j} = \text{PRG}(x_{m_j,j})$ for every $j \in \{1, \dots, \ell\}$, it follows that $\Pr [\text{Verify}(\text{vk}^*, \sigma^*) = 1] = 1$. Now let $m \neq m^*$ be arbitrary. We show the probability of vk^* accepting any signature on m is negligible:

$$\begin{aligned} & \Pr [\text{Verify}(\text{vk}^*, m, \sigma) = 1] \\ & \leq \Pr [y_{m_j,j} \in \text{Im}(\text{PRG}) \ \forall j \in \{1, \dots, \ell\}] \\ & \leq \Pr [y_{m_k,j} \in \text{Im}(\text{PRG})] \quad (\exists k \text{ s.t. } m_k \neq m_k^*) \\ & \leq 2^{\lambda - p(\lambda)} \leq 2^{-\lambda} \leq \text{negl}(\lambda) \end{aligned}$$

□

6 An Insecure-To-Secure Protocol Compiler

Ingredients. The description is given in [Prot. 6.1](#), which is based on the following building blocks:

- An insecure protocol $\pi = \langle A, B \rangle$ for evaluation of a deterministic function f with constant-length outputs y_A, y_B , where each party communicates one bit in each of L rounds. We further assume for simplicity that in the last L_{out} rounds of the protocol, A computes and sends y_B to B , while B computes and sends y_A to A . We describe how to remove these restrictions in [Sec. 8](#).
- A sub-exponentially secure input-succinct TM obfuscator iOM ([Def. 2](#)).
- A sub-exponentially secure function-binding hash ([Def. 3](#))⁹.
- A Dual-Mode Programmable Signature scheme DPSig ([Def. 4](#)).
- A sub-exponentially secure puncturable PRF $F_k : \{0, 1\}^* \rightarrow \{0, 1\}$ ([Def. 6](#)).
- A succinct extractable commitment scheme ECom ([Def. 10](#)).
- A statistically binding commitment scheme Com ([Def. 9](#)).
- A somewhere statistically binding hash ([Def. 5](#)).
- A succinct zero-knowledge argument of knowledge ZKAoK ([Def. 11](#)).
- A maliciously secure 2PC protocol ([Def. 15](#)).
- A standard EUF-CMA signature scheme Sig .

Additional Notation. We use a 1-indexed, end-inclusive “slicing” notation to index into a list. That is, let $\text{vec} := (v_1, \dots, v_N)$ be a list of N elements. If $1 \leq a \leq b \leq N$ are positive integers, then we define $\text{vec}[a] := v_a$, and $\text{vec}[a : b] = \text{vec}_{[a:b]} := (v_a, \dots, v_b)$. We may also use the notation $[a : b]$ to refer to the set $\{a, \dots, b\}$. We define $[N] := [1 : N]$. As in [Sec. 2](#), we use $\tilde{\cdot}$ to denote the encrypted versions of “plain” objects. For example, if $\text{tr} := (m_1^A, m_1^B, m_2^A, m_2^B, \dots)$ is the transcript from the “plain” protocol, then the corresponding encrypted transcript in the secure protocol is denoted by $\tilde{\text{tr}} := (\tilde{m}_1^A, \tilde{m}_1^B, \tilde{m}_2^A, \tilde{m}_2^B, \dots)$. We use $\text{prefix}_{\tilde{m}_i^A} := \tilde{\text{tr}}[1 : 2(i-1)]$ to denote the prefix of $\tilde{\text{tr}}$ up to, but not including, $\tilde{m}_i^A$. Similarly, $\text{prefix}_{\tilde{m}_i^B} := \tilde{\text{tr}}[1 : 2(i-1) + 1]$. As we shall see, [Phase 3](#) partitions $\tilde{\text{tr}}$ into N segments of η rounds each, where $N := \lfloor L/\eta \rfloor$, and segment length η is chosen later in [Sec. 6.2](#). For $j \in [N]$, we refer to rounds $(j-1)\eta + 1$ through $j\eta$ as the j -th segment,

⁹ Subexponential security of the function-binding hash can be avoided through a different sequence of hybrids; we adopt it here only to simplify the presentation.

and denote this by $\widetilde{\text{seg}}_j := \widetilde{\text{tr}}[2\eta(j-1) + 1 : 2\eta j]$. Finally, for “two-sided” communicated objects like $\widetilde{\text{tr}}$ and $\widetilde{\text{seg}}$, we use A or B as superscripts to denote their restriction the messages sent by that party. For example, $\widetilde{\text{tr}}^A := (\widetilde{m}_1^A, \widetilde{m}_2^A, \dots)$.

6.1 Our Construction

Inputs. Both parties receive as common input the descriptions of the machines $\langle A, B \rangle$, the round bound L , the segment length η , and two security parameters λ, λ_2 . Party P_A receives x_A as its private input, and party P_B receives x_B as its private input.

1. Preamble Phase

(a) Succinctly Commit to Inputs:

- P_A and P_B engage in an execution of **ECom**, where P_A commits to x_A with randomness r_A^{ext} . We use ecom_A to refer to (the transcript of) this **ECom**. Symmetrically, P_B commits to x_B , and similarly we use ecom_B to denote the (the transcript of) this **ECom**.

(b) Exchange Signing Material:

- i. P_A samples a SSB hash key H_{sig}^A and sends it to P_B . H_{sig}^A compresses η bits to ℓ_{sig} bits and statistically binds on a single input bit. Symmetrically, P_B sends H_{sig}^B to P_A .
- ii. P_A runs $\text{DPSig.Setup}(1^\lambda)$ for message space $\{0, 1\}^{\ell_{\text{sig}} N}$ times to generate key pairs $(\text{sk}_i^A, \text{vk}_i^A)_{i \in [N]}$. It then computes $h_{\text{vk}}^A := H_{\text{vk}}^A(\{\text{vk}_i^A\}_{i \in [N]})$, where H_{vk}^A is a SSB hash that statistically binds on a single vk_i^A . Finally, P_A sends $\{\text{vk}_i^A\}_{i \in [N]}$, h_{vk}^A , and H_{vk}^A to P_B . Symmetrically, P_B generates $(\text{sk}_i^B, \text{vk}_i^B)_{i \in [N]}$ and h_{vk}^B , then sends $\{\text{vk}_i^B\}_{i \in [N]}$, h_{vk}^B and H_{vk}^B to P_A .

(c) Hash the Inputs with CRS:

- i. (**Sample H_{crs}**) P_A and P_B use 2PC to jointly sample SSB hash H_{crs} .
- ii. P_A and P_B locally compute $h_{\text{crs}} := H_{\text{crs}}(\text{crs})$.
- iii. P_A generates a function-binding hash key H_{NM}^A by running $H_{\text{NM}}^A \leftarrow \text{Gen}(1^\lambda, \mathcal{F}_B, |x_B|, T)$, where T is a bound on the runtime of $\mathcal{F}_B$ on inputs of length $|x_B|$, and $\mathcal{F}_B$ is defined in [Algo. 6.1](#).

Algorithm 6.1: $\mathcal{F}_B$ // “next-message binding” function

Constants: Machines B , PPRF; key k_{crs} ; index $\ell_{\text{NM}} \in [L]$

Inputs: x , crs

Decrypt $m_i^A \leftarrow \text{crs}[i] \oplus F_{k_{\text{crs}}}(i)$ for $i \in \{1, \dots, \ell_{\text{NM}} - 1\}$

Output $m_{\ell_{\text{NM}}} \leftarrow B(x, m_1^A, \dots, m_{\ell_{\text{NM}}-1}^A)$

Note that P_A computes $k_{\text{crs}} \leftarrow \text{KGen}(1^{\lambda_2})$ and $\ell_{\text{NM}} \leftarrow 1$ during an honest execution (normal mode), and we refer to H_{NM}^A as the next-message binding hash throughout the rest of this work. P_A sends H_{NM}^A to P_B . Symmetrically, P_B computes and sends H_{NM}^B to P_A .

- iv. P_A computes $h_A := H_{\text{NM}}^B(x_A, \text{crs})$ and sends a commitment of h_A , denoted $\text{com}_A := \text{Com}(h_A; r_A^{\text{com}})$. Symmetrically, P_B computes $h_B := H_{\text{NM}}^A(x_B, \text{crs})$ and sends $\text{com}_B := \text{Com}(h_B; r_B^{\text{com}})$.
- v. (**Proof of Consistent Input**) P_A proves to P_B , via ZKAoK, its knowledge of x_A and the corresponding randomness such that x_A is the value committed in ecom_A , and com_A is a valid commitment to $h_A = H_{\text{NM}}^B(x_A, \text{crs})$. Formally, P_A proves that $\text{com}_A \in \mathcal{L}_{\text{Com}}$, where

$$\mathcal{L}_{\text{Com}} := \left\{ \text{com}_A \left| \begin{array}{l} \exists (x_A, r_A^{\text{ext}}, r_A^{\text{com}}) : \text{ecom}_A = \text{ECom}(x_A; r_A^{\text{ext}}) \wedge \\ \text{com}_A = \text{Com}(h_A; r_A^{\text{com}}) \wedge \\ h_A = H_{\text{NM}}^B(x_A, \text{crs}) \end{array} \right. \right\} \quad (1)$$

Similarly, P_B proves to P_A that com_B is generated correctly.

2. Obfuscation Generation Phase

The preamble τ_0 of this protocol is defined to be:

$$(\text{ecom}_A, \text{ecom}_B, H_{\text{sig}}^A, H_{\text{sig}}^B, h_{\text{vk}}^A, h_{\text{vk}}^B, H_{\text{vk}}^A, H_{\text{vk}}^B, H_{\text{NM}}^A, H_{\text{NM}}^B, \text{com}_A, \text{com}_B, H_{\text{crs}}, h_{\text{crs}})$$

(a) Parties run 2PC protocol Π_{setup} for functionality f_{setup} defined as follows:

- i. Sample PPRF key $\text{k}_{\text{msg}} \leftarrow \text{KGen}(1^{\lambda^2})$, and secret share k_{msg} as $\text{k}_{\text{msg}} = [\text{k}_{\text{msg}}]_A \oplus [\text{k}_{\text{msg}}]_B$.
- ii. Sample a key pair $(\text{sk}_{\text{msg}}, \text{vk}_{\text{msg}}) \leftarrow \text{Sig.Gen}(1^\lambda)$, and sign $[\text{k}_{\text{msg}}]_A, [\text{k}_{\text{msg}}]_B$ with sk_{msg} to obtain $\sigma_{[\text{k}_{\text{msg}}]_A}$ and $\sigma_{[\text{k}_{\text{msg}}]_B}$, respectively.
- iii. Sample a SSB hash key H_{tr} that has the following two properties: (1) statistically binds on the input length; (2) statistically binds on η bits of the input.
- iv. Get $\text{obNext}^A \leftarrow \text{iOM}(1^\lambda, \text{Next}^A)$ and $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}^B)$, where $\text{Next}^A, \text{Next}^B$ are defined as in [Algo. 6.2](#) (and padded to the maximum size of any modified versions in the security proof).
- v. Get $\text{com}_{\text{obf}}^A \leftarrow \text{Com}(\text{obNext}^A; r_A^{\text{obf}})$ and $\text{com}_{\text{obf}}^B \leftarrow \text{Com}(\text{obNext}^B; r_B^{\text{obf}})$
- vi. Output $([\text{k}_{\text{msg}}]_A, \sigma_{[\text{k}_{\text{msg}}]_A}, \text{vk}_{\text{msg}}, H_{\text{tr}}, \text{obNext}^A, \text{com}_{\text{obf}}^B, r_A^{\text{obf}})$ to P_A ,
and $([\text{k}_{\text{msg}}]_B, \sigma_{[\text{k}_{\text{msg}}]_B}, \text{vk}_{\text{msg}}, H_{\text{tr}}, \text{obNext}^B, \text{com}_{\text{obf}}^A, r_B^{\text{obf}})$ to P_B .

3. Encrypted Protocol Phase

Parties run “encrypted” $\langle A, B \rangle$ round-by-round, signing after every η rounds.

(a) For every round $i \in L$, $P_A \tilde{m}_i^A \leftarrow \text{obNext}^A(\text{inp}_i^A)$ where

$$\text{inp}_i^A := (x_A, r_A^{\text{com}}, \text{crs}, \tilde{\text{tr}}_{[1:i-1]}^B, \{\text{vk}_i^B\}_{i \in [N]}, \{\sigma_i^B\}_{i \in [n]})$$

Recall that r_A^{com} is the randomness P_A uses to commit to h_A , and n is defined in [Algo. 6.2](#). P_B responds with $\tilde{m}_i^B$, which is computed similarly.

(b) Messages sent by P_A (denoted as $\tilde{\text{tr}}^A$) are partitioned into segments of length η rounds each. After each segment:

- i. (**Proof of Honest Behavior**) P_A and P_B engage in an execution of ZKAoK, where P_A proves the messages it sends in the j -th segment $\widetilde{\text{seg}}_j^A$ is generated honestly.
Formally, P_A proves that $\text{com}_{\text{obf}}^A \in \mathcal{L}_{\text{seg}_j}$, where

$$\mathcal{L}_{\text{seg}_j} := \left\{ \text{com}_{\text{obf}}^A \mid \begin{array}{l} \text{ecom}_A = \text{ECom}(x_A; r_A^{\text{ext}}) \wedge \\ \exists w : \text{com}_{\text{obf}}^A = \text{Com}(\text{obNext}^A; r_A^{\text{obf}}) \wedge \\ \forall i \in [\eta(j-1)+1 : \eta j], \tilde{m}_i^A = \text{obNext}^A(\text{inp}_i^A) \end{array} \right\} \quad (2)$$

with $w := (x_A, r_A^{\text{ext}}, \text{obNext}^A, r_A^{\text{com}}, r_{\text{obf}}^A)$.

Similarly, P_B proves to P_A that it generates $\widetilde{\text{seg}}_j^B$ honestly.

- ii. P_A signs a hash of the j -th segment $\sigma_j^A := \text{DPSig.Sign}(\text{sk}_j^A, H_{\text{sig}}^A(\widetilde{\text{seg}}_j^A))$, and sends σ_j^A . Symmetrically, P_B sends σ_j^B .

4. Output Decryption Phase

(a) Let $L_{\text{out}} = |y_A| = |y_B|$ be the length of each output. Wlog assume the (encrypted) output for P_A is the last L_{out} messages (bits) sent by P_B , denoted as $\tilde{\text{tr}}_{\text{out}}^B$. Similarly, the encrypted output for P_B is $\tilde{\text{tr}}_{\text{out}}^A$.

- (b) P_A and P_B run a 2PC protocol Π_{dec} for functionality $f_{\text{dec}}(\text{in}^A, \text{in}^B)$, where f_{dec} takes common input τ_{dec} , P_A 's input $\text{in}^A := ([\text{k}_{\text{msg}}]_A, \sigma_{[\text{k}_{\text{msg}}]_A}, \text{vk}_{\text{msg}})$ and P_B 's input $\text{in}^B := ([\text{k}_{\text{msg}}]_B, \sigma_{[\text{k}_{\text{msg}}]_B}, \text{vk}_{\text{msg}})$, where τ_{dec} is defined as:

$$(\tilde{\text{tr}}_{\text{out}}^A, \tilde{\text{tr}}_{\text{out}}^B, \{h_{\text{out},i}^A, h_{\text{out},i}^B\}_{i \in [L-L_{\text{out}}]})$$

for $h_{\text{out},i}^A := \tau_0 \| \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})$ and $h_{\text{out},i}^B := \tau_0 \| \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})$ ¹⁰.

Additionally, $f_{\text{dec}}(\text{in}^A, \text{in}^B)$ is defined as follows:

- Verify signatures $\sigma_{[\text{k}_{\text{msg}}]_A}, \sigma_{[\text{k}_{\text{msg}}]_B}$ with vk_{msg} .
- Reconstruct $\text{k}_{\text{msg}} = [\text{k}_{\text{msg}}]_A \oplus [\text{k}_{\text{msg}}]_B$
- $\forall i \in [L - L_{\text{out}}]$, get $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{k}_{\text{msg}}}(h_{\text{out},i}^A)$, $m_i^B \leftarrow \tilde{m}_i^B \oplus F_{\text{k}_{\text{msg}}}(h_{\text{out},i}^B)$.
- Output $\{m_i^B\}_{i \in [L-L_{\text{out}}]}$ to P_A and $\{m_i^A\}_{i \in [L-L_{\text{out}}]}$ to P_B .

Abort Condition. If any verifications fail, parties abort and output $\perp$.

Algorithm 6.2: Description of Next^B	
Constants:	
– The machine description B; – Lengths L, η (encoded in binary); – τ_0; – An SSB hash key H_{tr}; – A PPRF key k_{msg}.	
Inputs:	
– crs; – x_B; // P_B's private input – $\tilde{\text{tr}}^A$; // Messages sent by P_A so far. – $\{\text{vk}_i^A\}_{i \in [N]}$; // Verification keys – $\{\sigma_i^A\}_{i \in [n]}$; // Signatures – r_B^{com}. // Randomness used by P_B to commit to h_B.	
Let $\ell := \tilde{\text{tr}}^A$ and parse $\tilde{\text{tr}}^A$ as $(\tilde{m}_1^A, \dots, \tilde{m}_\ell^A)$. If (i) $\ell > L$, (ii) $N \neq \lfloor L/\eta \rfloor$ or (iii) $n \neq \lfloor (\ell - 1)/\eta \rfloor$, then output $\perp$.	
Step I: Verify Inputs & Signatures.	
Output $\perp$ if any of the following checks fail:	
– $h_{\text{crs}} = \text{H}_{\text{crs}}(\text{crs})$; – $h_{\text{vk}}^A = \text{H}_{\text{vk}}^A(\{\text{vk}_i^A\}_{i \in [N]})$; – Let $h_B := \text{H}_{\text{NM}}^A(x_B, \text{crs})$. Check that $\text{com}_B = \text{Com}(h_B; r_B^{\text{com}})$; – For every $j \in [n]$, verify that	
$\text{DPSig.Verify}(\text{vk}_j^A, \text{H}_{\text{sig}}^A(\tilde{\text{seg}}_j^A), \sigma_j^A) = \top.$	
Step II: Decrypt & Compute.	

¹⁰ Intuitively, $\{h_{\text{out},i}^A, h_{\text{out},i}^B\}_{i \in [L-L_{\text{out}}]}$ are hash values needed to decrypt $\tilde{\text{tr}}_{\text{out}}^A, \tilde{\text{tr}}_{\text{out}}^B$. Additionally, note that H_{tr} is part of the output for both P_A and P_B , therefore both P_A and P_B can compute $\{h_{\text{out},i}^A, h_{\text{out},i}^B\}_{i \in [L-L_{\text{out}}]}$

For every $i \in [\ell]$, compute the following:

$$\begin{aligned} m_i^A &:= \tilde{m}_i^A \oplus F_{\text{msg}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})). \\ m_i^B &:= B(x_B, \text{tr}_{[1:i]}^A). \quad // \text{Recall that } \text{tr}_{[1:i]}^A = m_1^A \| \dots \| m_i^A \\ \tilde{m}_i^B &:= m_i^B \oplus F_{\text{msg}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})). \end{aligned}$$

Output: return $\tilde{m}_\ell^B$.

Theorem 4. (Constant Rate 2PC Compiler) *Prot. 6.1* compiles an insecure protocol $\pi = \langle A, B \rangle$ for deterministic f with constant-length outputs and single-bit messages with communication complexity cc , into a two-party protocol $\Pi = \langle P_A, P_B \rangle$ that securely evaluates f with abort in the presence of static malicious adversaries, with communication complexity $O(\text{cc})$.

6.2 Parameters and Rate

As alluded to earlier, to get constant rate, we will have two security parameters, λ and λ_2 , where λ is the smaller of the two. We set λ_2 as the security parameter for the following primitives are assumed to be subexponentially secure: (a) the input-succinct Turing machine obfuscator iOM ; (b) the puncturable PRF PPRF ; (c) the function binding hash. Therefore, for any attacker running in time polynomial in λ_2 , there is a constant $\varepsilon \in (0, 1)$ such that the distinguishing advantage for any of these primitives is at most $2^{-\lambda_2^\varepsilon}$. All other primitives use parameter λ .

Recall that our construction uses signatures and succinct ZK proofs of honest behavior in [Phase 3](#) after every segment of length η . The size of each signature, each verification key, and communication in each succinct proof are bounded by some fixed polynomial in the *shorter* parameter, say $\text{poly}_1(\lambda)$, and there are $N = O(\text{cc}/\eta)$ such segments. We set:

$$\eta = \lambda \cdot \text{poly}_1(\lambda), \quad \lambda_2 = \eta^{2/\varepsilon}.$$

Observe that the total overhead due to verification keys, signatures, and succinct proofs is at most $N \cdot O(\text{poly}_1(\lambda))$, total communication in the encrypted protocol ([Phase 3](#)) is exactly cc , total communication in all phases outside [Phase 3](#) is bounded by some fixed polynomial in the larger security parameter, say $\text{poly}_2(\lambda_2)$, since we have assumed that the protocol has short outputs. Therefore, total communication of the compiled protocol is

$$N \cdot \text{poly}_1(\lambda) + \text{cc} + \text{poly}_2(\lambda_2) = O(\text{cc}/\lambda) + \text{cc} + \text{poly}_2(\lambda_2).$$

This simplifies to $O(\text{cc})$. For outputs of length ℓ_f , as we discuss later, we have to send $O(\ell_f)$ extra bits, so total communication remains $O(\text{opt}_\pi)$ as desired.

7 Proof of Security

Lemma 1. *Prot. 6.1 is secure against static malicious adversaries.*

Our proof assumes a malicious PPT adversary P_B^* in [Prot. 6.1](#). The proof for a malicious P_A^* is symmetric. Consider the following sequence of hybrids.

$\mathcal{G}_0$ (REAL): This is the real experiment where simulator Sim follows [Prot. 6.1](#) with input x_A . This experiment returns the outputs of P_A and P_B^* .

$\mathcal{G}_1$: This hybrid is identical to $\mathcal{G}_0$ except that it invokes the extractor $\mathcal{E}$ for ECom to extract the value x_B^* and the corresponding randomness $r_B^{\text{ext}*}$ from ecom_B . If the extraction fails, it outputs $\perp$; otherwise, it sends x_B^* to the ideal functionality for f and receives y_B^* . Note that this experiment obtains x_B^* and y_B^* but does

not yet use them. By the extractability of ECom, the probability that this hybrid outputs $\perp$ is negligible. Thus, $\mathcal{G}_1 \stackrel{s}{\approx} \mathcal{G}_0$.

$\mathcal{G}_2$: This hybrid is identical to $\mathcal{G}_1$ except that it invokes the simulator for all occurrences of ZKAoK sent by P_A . In particular, this includes:

- The **Proof of Consistent Input** in Step 1(c)v;
- The **Proof of Honest Behavior** in Step 3(b)i.

By the zero-knowledge property of ZKAoK, we have $\mathcal{G}_2 \stackrel{c}{\approx} \mathcal{G}_1$.

$\mathcal{G}_3$: Identical to $\mathcal{G}_2$ except that it invokes the simulator for the 2PC that samples H_{crs} in Step 1(c)i. By the malicious security of 2PC, we have $\mathcal{G}_3 \stackrel{c}{\approx} \mathcal{G}_2$.

$\mathcal{G}_4$: This hybrid is identical to $\mathcal{G}_3$ except that it invokes the simulator for the 2PC protocol Π_{setup} in Phase 2. By the malicious security of 2PC, we have $\mathcal{G}_4 \stackrel{c}{\approx} \mathcal{G}_3$.

Claim 1. *In hybrid $\mathcal{G}_4$, the output delivered to P_B^* by Π_{dec} is exactly y_B^* .*

Proof. To prove the claim, we first show that the value x_B^* extracted by Sim from ecom_B is the same value that P_B^* uses to compute $\tilde{\text{tr}}^B$ and generate ZKAoK for the **Proof of Honest Behavior** in Step 3(b)i.

Suppose, for the sake of contradiction, that P_B^* uses a value $x'_B \neq x_B^*$ as part of the witness in one of the ZKAoK with non-negligible probability. Then we can construct an adversary $\mathcal{A}^*$ that breaks the computational binding property of ECom, as shown in Algo. 7.1.

Algorithm 7.1: $\mathcal{A}^*$ against computational binding of ECom
$\mathcal{A}^*$ behaves as in $\mathcal{G}_4$, except that for every ZKAoK sent by P_B^* (in Algo. 7.1), it proceeds as follows: 1. It invokes the extractor $\mathcal{E}$ to extract a witness $(x'_B, r_B^{\text{ext}'}, \text{obNext}^B, r^B, r_{\text{obf}}^B)$; 2. If $x'_B \neq x_B^*$, then $\mathcal{A}^*$ outputs both $(x'_B, r_B^{\text{ext}'})$ and $(x_B^*, r_B^{\text{ext}*})$.

Clearly, $\mathcal{A}^*$ runs in polynomial time, and by assumption it outputs two different openings of ecom_B with non-negligible probability, contradicting the computational binding property of ECom. Hence, $x'_B = x_B^*$ with all but negligible probability.

Since P_B^* is bound to the same input x_B^* throughout the protocol, the unforgeability of Sig together with the correctness of Prot. 6.1 implies that the output delivered to P_B^* by Π_{dec} is exactly y_B^* . This proves Claim 1.

$\mathcal{G}_5$: This hybrid is identical to $\mathcal{G}_4$ except that it invokes the simulator for the 2PC protocol Π_{dec} in Phase 4, using y_B^* as the prescribed output. Since we proved that in $\mathcal{G}_4$ the output of Π_{dec} received by P_B^* is y_B^* , the malicious security of the 2PC implies that $\mathcal{G}_5 \stackrel{c}{\approx} \mathcal{G}_4$.

7.1 Sequence of Hybrids

Next, we define a sequence of hybrids Hyb_j for every $j \in [N]$ (recall that N is the total number of segments). Hybrid Hyb_0 is identical to $\mathcal{G}_5$. This sequence of gradually removes Sim's dependence on x_A and on P_B^* 's messages, while simultaneously transforming the obfuscated program obNext^B into a dummy machine, segment by segment.

Description of Hyb_j : At a high level, the first j segments sent by Sim in Phase 3 are no longer computed according to the honest protocol, but are replaced with dummy values $F_{k_{\text{path}}}(i)$, where i denotes the round number (depicted as the red path in Fig. 1). Correspondingly, the obfuscated program obNext^B is modified

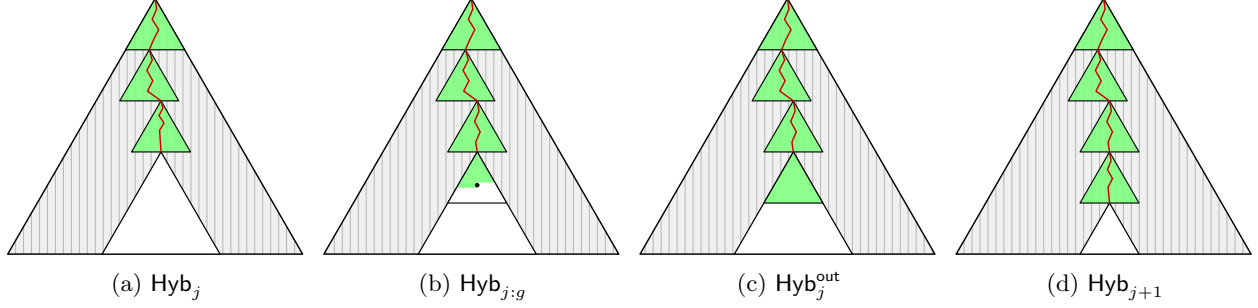


Fig. 1: Evolution of the hybrid argument. Green regions represent the space of inputs for which the adversary can learn a simulated response from the program, white region represents the remaining real transcript space, red line represents the simulated transcript sent by the simulator to the adversary.

so that the space of acceptable inputs $\tilde{\text{tr}}^A$ is restricted (illustrated by the green and white regions in Fig. 1): it accepts only those inputs $\tilde{\text{tr}}^A$ whose prefix, up to the largest completed segment among the first j segments, matches the F_{kpath} -generated prefix fixed by Sim.

For every acceptable input $\tilde{\text{tr}}^A$, the program outputs dummy values on the first j segments (shown as the green region in Fig. 1). Formally, the dummy output at round i is computed as $\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$. Beyond the j -th segment, both Sim and obNext^B continue to behave honestly. In more detail, this hybrid is identical to Hyb_{j-1} except for the following changes:

1. In [Step 1\(b\)ii](#), the verification key vk_j^A for the j th segment is generated differently. Specifically, Sim generates

$$\text{DPSig.Prog}(1^\lambda, h_j^{\text{sig}}) \mapsto (\text{sk}_j, \text{vk}_j^A), \text{ where } h_j^{\text{sig}} := H_{\text{sig}}^A(F_{\text{kpath}}(\eta(j-1) + 1 : \eta j))^{11}.$$

2. In [Phase 3](#), during the j th segment, Sim sends pseudorandom bits to P_B^* instead of computing its messages honestly. Formally, for every round $i \in [\eta(j-1) + 1 : \eta j]$, Sim sends $F_{\text{kpath}}(i)$ to P_B^* .
3. Still in [Phase 3](#), to decrypt P_B^* 's messages in the j th segment (denoted $\widetilde{\text{seg}}_j^B$), the simulator decrypts each $\tilde{m}_i^B$ as

$$m_i^B := \tilde{m}_i^B \oplus F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})) \oplus B(x_B^*, \text{Dec}_{\text{kcrs}}(\text{crs}_{[1:i]})),$$

where $i \in [\eta(j-1) + 1 : \eta j]$, x_B^* is the value extracted by Sim from ecom_B , and $\text{Dec}_{\text{kcrs}}(\text{crs}_{[1:i]})$ is the decryption of the first i bits of crs under kcrs . Formally, $\text{Dec}_{\text{kcrs}}(\text{crs}[i]) := F_{\text{kcrs}}(i) \oplus \text{crs}[i]$.

Observe that in Hyb_j , Sim's messages in the j th segment, namely $\widetilde{\text{seg}}_j^A$ (equivalently $\tilde{m}_i^A$ for $i \in [\eta(j-1)+1 : \eta j]$), no longer depend on $\widetilde{\text{seg}}_j^B$. However, from the $(j+1)$ th segment onward, the simulator behaves honestly when computing its messages:

$$\tilde{m}_i^A := F_{\text{kmsg}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})) \oplus A(x_A, \text{tr}_{[1:i-1]}^B),$$

for all $i \in [\eta j + 1 : L]$. Note that $\text{tr}_{[1:i-1]}^B$ still depends on $\widetilde{\text{seg}}_j^B$, and therefore decryption of $\widetilde{\text{seg}}_j^B$ remains necessary.

4. Finally, Sim sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_j}^B)$, where $\text{Next}_{\text{Hyb}_j}^B$ is defined in [Algo. 7.2](#). At a high level, in [Algo. 7.2](#), for any input $\tilde{\text{tr}}^A$, if there exists $t \in [j]$ such that $t\eta < |\tilde{\text{tr}}^A| \leq (t+1)\eta$. The program accepts only if the prefix $\tilde{\text{tr}}^A[1 : t\eta]$ equals $F_{\text{kpath}}(1 : t\eta)$. Moreover, for such accepting inputs, the program sets the plaintext messages m_i^A for all $i \leq t\eta$ by decrypting the corresponding entries of crs , rather than by decrypting the corresponding components of $\tilde{\text{tr}}^A$.

¹¹ With a slight abuse of notation, $F_{\text{kpath}}(\eta(j-1) + 1 : \eta j)$ denotes the concatenation of $F_{\text{kpath}}(\eta(j-1) + i)$ for $i \in [\eta]$.

Algorithm 7.2: Description of $\text{Next}_{\text{Hyb}_j}^B$ // identical to Next^B , except for the modifications listed below in blue**Constants.** // same as before, with the following additional hard-coded values

- ...
- PPRF keys $k_{\text{msg}}, k_{\text{dummy}}, k_{\text{path}}$;
- k_{crs} , the decryption key for crs ;
- ...

Step I: Verify Input & Signatures. // same as in before, with the following additional check:

- ...
- Let $t = \min(n, j)$. Verify that

$$\tilde{\text{tr}}^A[1 : t\eta] \stackrel{?}{=} F_{k_{\text{path}}}(1 : t\eta).$$

If any check fails, output $\perp$.**Step II: Decryption & Compute.**

- If $\ell \leq j \cdot \eta$:
 - For all $i \in [\ell]$, compute

$$\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- Otherwise:

- For $i \in [j \cdot \eta]$:

* Compute

$$m_i^A \leftarrow \text{Dec}_{k_{\text{crs}}}(\text{crs}[i]).$$

* Compute

$$\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- For $i \in [j \cdot \eta + 1 : \ell]$:

* Compute

$$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})).$$

* Compute

$$m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A).$$

* Compute

$$\tilde{m}_i^B \leftarrow m_i^B \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- Output $\tilde{m}_\ell^B$.

Claim 2. For every $j \in [0 : N - 1]$, we have $\text{Hyb}_j \stackrel{c}{\approx} \text{Hyb}_{j+1}$.**Proof.** Fix $j \in [0 : N - 1]$. To prove $\text{Hyb}_j \stackrel{c}{\approx} \text{Hyb}_{j+1}$, we introduce an intermediate hybrid $\text{Hyb}_j^{\text{out}}$ and show

$$\text{Hyb}_j \stackrel{c}{\approx} \text{Hyb}_j^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j+1}.$$

$\text{Hyb}_j^{\text{out}}$: At a high level, [Algo. 7.3](#) is a modification of [Algo. 7.2](#) such that for any acceptable $\tilde{\text{tr}}^A$ with $j\eta < |\tilde{\text{tr}}^A| \leq (j+1)\eta$, it outputs a dummy value (cf. [Fig. 1](#)). Additionally, Sim updates its decryption procedure to preserve consistency with the P_B^* 's view.

Formally, this hybrid is identical to Hyb_j except for the following changes:

- In [Phase 3](#) of [Prot. 6.1](#), to decrypt P_B^* 's messages in the $(j+1)$ th segment (denoted $\widetilde{\text{seg}}_{j+1}^B$), the simulator decrypts each $\tilde{m}_i^B$ as

$$m_i^B := \tilde{m}_i^B \oplus F_{k_{\text{dummy}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})) \oplus B(x_B^*, \text{Dec}_{k_{\text{crs}}}(\text{crs}_{[1:i]})),$$

where $i \in [\eta j + 1 : \eta(j+1)]$, x_B^* is the value extracted by `Sim` from `ecomB`, and $\text{Dec}_{k_{\text{crs}}}(\text{crs}_{[1:i]})$ is the decryption of the first i bits of `crs` under k_{crs} .

- `Sim` sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_j^{\text{out}}}^B)$ where $\text{Next}_{\text{Hyb}_j^{\text{out}}}^B$ is shown in [Algo. 7.3](#).

Algorithm 7.3: Description of $\text{Next}_{\text{Hyb}_j^B}$

// identical to $\text{Next}_{\text{Hyb}_j}^B$, except for the modifications listed below in [blue](#)

$\vdots$

Step I: Verify Input & Signatures. // added in $\text{Next}_{\text{Hyb}_j}^B$

- ...
- Let $t = \min(n, j)$. Verify that

$$\text{tr}^A[1 : t \cdot \eta] \stackrel{?}{=} F_{k_{\text{path}}}(1 : t \cdot \eta).$$

If any check fails, output $\perp$.

Step II: Decryption & Compute.

- If $\ell \leq (j+1) \cdot \eta$:

- * For all $i \in [\ell]$, compute

$$\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- Otherwise:

- * For $i \in [j \cdot \eta]$:

- Compute

$$m_i^A \leftarrow \text{Dec}_{k_{\text{crs}}}(\text{crs}[i]).$$

- Compute

$$\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- * [For \$i \in \[j\eta + 1 : \(j+1\)\eta\]\$: // The \$\(j+1\)\$ -th Segment](#)

- [Compute](#)

$$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$$

- [Compute](#)

$$\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$$

- * For $i \in [(j+1) \cdot \eta + 1 : \ell]$:

- Compute

$$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})).$$

- Compute

$$m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A).$$

- Compute

$$\tilde{m}_i^B \leftarrow m_i^B \oplus F_{k_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- Output $\tilde{m}_\ell^B$.

7.1.1 Proof of $\text{Hyb}_j^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_j$

Claim 3. For every $j \in [0 : N - 1]$, $\text{Hyb}_j^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_j$

Proof. Recall that for any input $\tilde{\text{tr}}^A$ to obNext^B satisfying $j\eta < |\tilde{\text{tr}}^A| \leq (j+1)\eta$, the program accepts only if $\tilde{\text{tr}}^A[1 : j\eta] = F_{\text{k}_{\text{path}}}(1 : j\eta)$. Define hybrids $\text{Hyb}_{j:g}$ for $g \in [0 : 2^{\eta+1} - 2]$, where $\text{Hyb}_{j:0} := \text{Hyb}_j$ and $\text{Hyb}_{j:2^{\eta+1}-2} := \text{Hyb}_j^{\text{out}}$.

These hybrids enumerate all acceptable inputs $\tilde{\text{tr}}^A$ to obNext^B with length $|\tilde{\text{tr}}^A| > j\eta$ in shortlex order (see [Sec. 3](#)). Moving from $\text{Hyb}_{j:g}$ to $\text{Hyb}_{j:g+1}$, we first hardwire obNext^B 's behavior on the $(g+1)$ -th *acceptable* input (in shortlex order) and then replace the corresponding output with a dummy value. Consequently, by the final hybrid, obNext^B 's behavior on the entire $(j+1)$ -th segment is fully simulated, rather than computed according to the real protocol.

For $0 \leq g \leq 2^{\eta+1} - 2$, let str_g denote the g -th string in shortlex order. In particular, str_0 is empty, $\text{str}_1 = 0$, $\text{str}_2 = 1$, $\text{str}_3 = 00$, $\dots$, and $\text{str}_{2^{\eta+1}-2} = 1^\eta$, and we set str_0 to be the empty string.

$\text{Hyb}_{j:g}$: Identical to $\text{Hyb}_{j:g-1}$ except that $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g}}^B$ is defined in [Algo. 7.4](#). At a high level, [Algo. 7.4](#) outputs dummy values for the first g acceptable inputs in shortlex order.

Algorithm 7.4: Description of $\text{Next}_{\text{Hyb}_{j:g}}^B$ // identical to $\text{Next}_{\text{Hyb}_j}^B$, except for the modifications listed below in blue
Constants. // same as before, with the following additional hard-coded values – $\dots$ – str_g, and let $\ell_g := \text{str}_g$. $\dots$ Step II: Decryption & Compute. – If $\ell < \eta j + \ell_g$ or ($\ell = \eta j + \ell_g$ and $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_g] \leq \text{str}_g$) : // all inputs that are smaller than str_g in shortlex order • For all $i \in [\ell]$, compute $\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$. – Otherwise: • For $i \in [j \cdot \eta]$, * Compute $m_i^A \leftarrow \text{Dec}_{\text{k}_{\text{crs}}}(\text{crs}[i])$ * Compute $\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ • For $i \in [j\eta + 1 : j\eta + \ell_g - 1]$, * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{k}_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$ * Compute $\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ • For $i = j\eta + \ell_g$, * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{k}_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$ * If $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_g] \leq \text{str}_g$, • Compute $\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ * else, • Compute $m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A)$

- Compute $\tilde{m}_i^B \leftarrow m_i^B \oplus F_{\text{kmsg}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
- For $i \in [j \cdot \eta + \ell_g + 1 : \ell]$,
 - * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$
 - * Compute $m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A)$
 - * Compute $\tilde{m}_i^B \leftarrow m_i^B \oplus F_{\text{kmsg}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
- Output $\tilde{m}_\ell^B$

Claim 4. For every $j \in [0 : N - 1]$ and $g \in [0 : 2^{\eta+1} - 3]$, we have that $\text{Hyb}_{j:g}$ and $\text{Hyb}_{j:g+1}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable.

Proof. As discussed above, to move from $\text{Hyb}_{j:g}$ to $\text{Hyb}_{j:g+1}$, we focus on modifying the behavior of obNext^B on the $(g+1)$ -th *acceptable* input $\tilde{\text{tr}}^A$ (in shortlex order) whose length exceeds $j\eta$. Concretely, an acceptable input satisfies the following condition:

$$|\tilde{\text{tr}}^A| = j\eta + \ell_{g+1} \quad \wedge \quad \tilde{\text{tr}}_{[1:j\eta]}^A = F_{\text{kpath}}(1 : j\eta) \quad \wedge \quad \tilde{\text{tr}}_{[\eta j+1:\eta j+\ell_{g+1}]}^A = \text{str}_{g+1} \quad (3)$$

That is, we focus on $\tilde{\text{tr}}^A$ whose $(j+1)$ -th segment prefix equals str_{g+1} . We now consider the following sequence of hybrids.

$\text{Hyb}_{j:g}^{(1)}$: This hybrid is identical to $\text{Hyb}_{j:g}$ except that the first ℓ_{g+1} bits in the $(j+1)$ -th segment of crs are changed to encryptions (under k_{crs}) of decryptions (under k_{msg}) of str_{g+1} . Formally, for $i \in [\eta j + 1 : \eta j + \ell_{g+1}]$,

$$\text{crs}[i] = F_{\text{k}_{\text{crs}}}(i) \oplus m_i^A,$$

where the m_i^A s are computed as follows.

1. Parse str_{g+1} as $\tilde{\text{tr}}_{[\eta j+1:\eta j+\ell_{g+1}]}^A = \{\tilde{m}_{\eta j+1}^A, \tilde{m}_{\eta j+2}^A, \dots, \tilde{m}_{\eta j+\ell_{g+1}}^A\}$
2. For $i \in [\eta j + 1 : \eta j + \ell_{g+1}]$, compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$

By the pseudorandomness of PPRF at the punctured points, we have that $\text{Hyb}_{j:g}^{(1)}$ and $\text{Hyb}_{j:g}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable.

Remark 1. The observant reader may notice that $\text{Hyb}_{j:g}^{(1)}$ should formally be replaced with a sequence of hybrids for every $i \in [\eta j + 1 : \eta j + \ell_{g+1}]$ that (1) punctures k_{crs} at i and hardcodes the punctured key in obfuscation; (2) changes $\text{crs}[i]$ as in $\text{Hyb}_{j:g}^{(1)}$; and (3) unpunctures k_{crs} . Changes (1) and (3) follow from security of iOM since functionality of obfuscation is unchanged, while change (2) follows from pseudorandomness of PPRF at punctured points. We omitted these as formal hybrids for brevity.

$\text{Hyb}_{j:g}^{(2)}$: This is identical to $\text{Hyb}_{j:g}^{(1)}$ except that Sim sets $\ell_{\text{NM}} = j\eta + \ell_{g+1}$ when computing H_{NM}^A in [Algo. 6.1](#). By the function hiding property of function-binding hash, we have that $\text{Hyb}_{j:g}^{(2)}$ and $\text{Hyb}_{j:g}^{(1)}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable¹².

$\text{Hyb}_{j:g}^{(3)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(2)}$ except for the following changes:

¹² If $\ell_{g+1} = \ell_g$, then hybrid $\text{Hyb}_{j:g}^{(2)}$ is unnecessary. That is, we only need to appeal to the security of the function binding hash once per layer. Consequently, subexponential security of the function-binding hash is not inherently required; we assume it here only to simplify the exposition.

1. The simulator defines

$$m^* := F_{\text{kmsg}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B})) \oplus B(x_B^*, \text{Dec}_{\text{kcrs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]})$$

Note that to compute $\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B}$, we need to compute $\{\tilde{m}_i^A\}_{i \in [\eta j + \ell_{g+1}]}$ and $\{\tilde{m}_i^B\}_{i \in [\eta j + \ell_{g+1} - 1]}$ which are computed as

- For $i \in [1 : \eta j + \ell_{g+1}]$, $\tilde{m}_i^A = \tilde{\text{tr}}^A[i]$ and sets $\tilde{\text{tr}}^A$ as the $(g+1)$ -th *acceptable* input ([Condition 3](#)).
- For $i \in [1 : \eta j + \ell_{g+1} - 1]$, $\tilde{m}_i^B = F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$.

2. Sim sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B)$ where $\text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B$ is shown in [Algo. 7.5](#). At a high level, if $\tilde{\text{tr}}^A$ is the $(g+1)$ -th *acceptable* input ([Condition 3](#)) to obfuscated program, then [Algo. 7.5](#) directly outputs the hard-coded value m^* instead of computing the value according to the honest protocol logic.

Algorithm 7.5: Description of $\text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B$ // identical to $\text{Next}_{\text{Hyb}_{j:g}^B}$, except for the modifications listed below in blue Constants. // same as before, with the following additional hard-coded values – str_{g+1}, and let $\ell_{g+1} := \text{str}_{g+1}$. – m^* Hard-coded Output: – If $\ell = \ell_{g+1} + j\eta$ and $\tilde{\text{tr}}_{[\eta j + 1 : \eta j + \ell_{g+1}]}^A = \text{str}_{g+1}$, output m^* and HALT. Step II: Decryption & Compute. – If $\ell < \eta j + \ell_{g+1}$ or ($\ell = \eta j + \ell_{g+1}$ and $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_{g+1}] < \text{str}_{g+1}$) : // all inputs that are smaller than str_{g+1} in shortlex order • For all $i \in [\ell]$, compute $\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$. – Otherwise: • For $i \in [j \cdot \eta]$, * Compute $m_i^A \leftarrow \text{Dec}_{\text{kcrs}}(\text{crs}[i])$ * Compute $\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ • For $i \in [j\eta + 1 : j\eta + \ell_{g+1} - 1]$, * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$ * Compute $\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ • For $i = j\eta + \ell_{g+1}$, * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$ * If $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_{g+1}] < \text{str}_{g+1}$, • Compute $\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ * If $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_{g+1}] = \text{str}_{g+1}$, • $\tilde{m}_i^B = m^*$ * else, • Compute $m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A)$ • Compute $\tilde{m}_i^B \leftarrow m_i^B \oplus F_{\text{kmsg}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$ • For $i \in [j \cdot \eta + \ell_{g+1} + 1 : \ell]$, * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$
--

- * Compute $m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A)$
- * Compute $\tilde{m}_i^B \leftarrow m_i^B \oplus F_{\text{kmsg}}(\tau_0 \parallel \text{Htr}(\text{prefix}_{\tilde{m}_i^B}))$
- Output $\tilde{m}_\ell^B$

Claim 5. $\text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B$ and $\text{Next}_{\text{Hyb}_{j:g}^{(2)}}^B$ have the same functionality.

Proof. It suffices to consider the behavior of the obfuscated programs on the $(g+1)$ -th *acceptable* input $\tilde{\text{tr}}^A$ in shortlex order whose length exceeds $j\eta$ (See [Condition 3](#)), since this is the only input on which the programs' behavior is modified.

Behavior of the two programs.

- $\text{Next}_{\text{Hyb}_{j:g}^{(2)}}^B$ outputs

$$F_{\text{kmsg}}(\tau_0 \parallel \text{Htr}(\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B})) \oplus B(x_B, \text{tr}_{[1:\eta j + \ell_{g+1}]}^A), \quad (4)$$

where x_B is an arbitrary input provided by P_B^* .

- $\text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B$ outputs the hardcoded value

$$m^* = F_{\text{kmsg}}(\tau_0 \parallel \text{Htr}(\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B})) \oplus B(x_B^*, \text{Dec}_{\text{kcrs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]}) , \quad (5)$$

where x_B^* is extracted from ecom_B .

Equality of the PRF values. Observe that in $\text{Hyb}_{j:g}^{(3)}$, the value $\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B}$ is fixed in advance and is defined consistently with input $\tilde{\text{tr}}^A$ that satisfies [Condition 3](#)). Consequently, $\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B}$ is identical in both [Eq. \(4\)](#) and [Eq. \(5\)](#). Hence, the PRF is evaluated on the same input in both expressions, and therefore the resulting PRF values are equal.

Equality of the $B(\cdot, \cdot)$ terms. It remains to show

$$B(x_B, \text{tr}_{[1:\eta j + \ell_{g+1}]}^A) = B(x_B^*, \text{Dec}_{\text{kcrs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]}) , \quad (6)$$

First, recall that in $\text{Hyb}_{j:g}^{(1)}$, the first ℓ_{g+1} bits in the $(j+1)$ segment of crs are changed to encryptions of decryptions of str_{g+1} . Therefore, if $\tilde{\text{tr}}^A$ satisfies [Condition 3](#), we have

$$\text{tr}_{[j\eta + 1:j\eta + \ell_{g+1}]}^A \equiv \text{Dec}_{\text{kcrs}}(\text{crs}_{[j\eta + 1:j\eta + \ell_{g+1}]}).$$

Additionally, recall that in $\text{Next}_{\text{Hyb}_{j:g}^{(2)}}^B$, the first j segments of tr^A are obtained by decrypting the corresponding positions of the input crs . Therefore, we have

$$\text{tr}_{[1:j\eta + \ell_{g+1}]}^A \equiv \text{Dec}_{\text{kcrs}}(\text{crs}_{[1:j\eta + \ell_{g+1}]}).$$

However, since com_B is not statistically binding with respect to x_B and crs , it is possible that the adversary P_B^* provides alternative values $x'_B \neq x_B^*$ and $\text{crs}' \neq \text{crs}$ to $\text{Next}_{\text{Hyb}_{j:g}^{(2)}}^B$ such that

$$\text{com}_B = \text{Com}(\text{H}_{\text{NM}}^A(x'_B, \text{crs}'); r_B^{\text{com}}).$$

In this case, the equality in [Eq. \(6\)](#) is no longer guaranteed to hold.

Claim 6. For all x'_B and crs' , if $\tilde{\text{tr}}^A$ satisfies [Condition 3](#) and

$$\text{Com}(\text{H}_{\text{NM}}^A(x'_B, \text{crs}'); r_B^{\text{com}}) = \text{com}_B,$$

then

$$B(x'_B, \text{tr}_{[1:\eta j + \ell_{g+1}]}^A) = B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]}))$$

except with negligible probability.

Proof. Suppose for contradiction that P_B^* commits to a value $x'_B \neq x_B^*$ in com_B and produces an accepting ZKAoK in [Step 1\(c\)v \(Proof of Consistent Input\)](#), using x'_B as part of the witness for [Language \(1\)](#). Then we can build an adversary $\mathcal{A}^*$ that breaks the computational binding of ECom similar to [Algo. 7.1](#). Hence, $x'_B = x_B^*$ with all but negligible probability. Additionally, by the soundness of ZKAoK, com_B is generated honestly with respect to the correct crs , so we have

$$\text{com}_B = \text{Com}(\text{H}_{\text{NM}}^A(x_B^*, \text{crs}); r_B^{\text{com}}).$$

Finally, since Sim sets $\ell_{\text{NM}} = j\eta + \ell_{g+1}$ when computing H_{NM}^A in [Algo. 6.1](#), the statistical binding property of function-binding hash ensures that for all inputs to the obfuscation program x'_B and crs'

$$B(x'_B, \text{tr}_{[1:\eta j + \ell_{g+1}]}^A) = B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]})) .$$

□

Therefore, for all acceptable inputs x'_B and crs' , the two programs will produce identical outputs on every input, and hence

$$\text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B \equiv \text{Next}_{\text{Hyb}_{j:g}^{(2)}}^B .$$

□

By [Claim 5](#) and the the sub-exponential security of the indistinguishability obfuscator iOM , we have that $\text{Hyb}_{j:g}^{(3)}$ and $\text{Hyb}_{j:g}^{(2)}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable.

$\text{Hyb}_{j:g}^{(4)}$: This is identical to $\text{Hyb}_{j:g}^{(3)}$ except that H_{tr} is now statistically binds on $\widetilde{\text{seg}}_{j+1}^A$, the $(j+1)$ th segment of $\tilde{\text{tr}}^A$. By the mode indistinguishability of H_{tr} , it follows that $\text{Hyb}_{j:g}^{(4)}$ and $\text{Hyb}_{j:g}^{(3)}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable.

$\text{Hyb}_{j:g}^{(5)}$: This is identical to $\text{Hyb}_{j:g}^{(4)}$ except for the following changes:

1. Define $\text{p}_{g+1} := \tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}}^B)$
2. Sim computes $r^* := \text{F}_{\text{k}_{\text{msg}}}(\text{p}_{g+1})$
3. The value m^* is set as $m^* := r^* \oplus B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]}))$
4. Sim computes $\text{k}_{\text{msg}}^* \leftarrow \text{Punc}(\text{k}, \text{p}_{g+1})$, where $\text{k} \leftarrow \text{KGen}(1^\lambda)$.
5. Finally, Sim sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$ is defined in [Algo. 7.6](#). The only difference between [Algo. 7.6](#) and the previous obfuscated program is that k_{msg}^* is hardcoded in place of k_{msg} .

Algorithm 7.6: Description of $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$	
// identical to $\text{Next}_{\text{Hyb}_{j:g}^{(3)}}^B$, except for the modifications listed below in blue	
Constants. // same as before, with the following additional hard-coded values	
—	...

– PPRF keys $k_{\text{msg}}, k_{\text{msg}}^*, k_{\text{dummy}}, k_{\text{path}};$ – ...

Claim 7. $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$ and $\text{Next}_{\text{Hyb}_{j:g}^{(4)}}^B$ have identical functionality.

Proof. Note that the value of m^* remains unchanged between the two hybrids. It therefore suffices to show that puncturing the key to obtain k_{msg}^* does not affect the functionality of the obfuscation program, namely, its ability to correctly encrypt and decrypt on any input transcript $\tilde{\text{tr}}^A$.

First we observe that

- To decrypt $\tilde{m}_i^A$, $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$ (and $\text{Next}_{\text{Hyb}_{j:g}^{(4)}}^B$) needs to compute

$$H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}), \quad \text{where} \quad \text{prefix}_{\tilde{m}_i^A} := \tilde{\text{tr}}_{[1:2 \cdot (i-1)]},$$

So, $\text{prefix}_{\tilde{m}_i^A}$ always has even length.

- To encrypt $\tilde{m}_i^B$, $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$ (and $\text{Next}_{\text{Hyb}_{j:g}^{(4)}}^B$) needs to compute

$$H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}), \quad \text{where} \quad \text{prefix}_{\tilde{m}_i^B} := \tilde{\text{tr}}_{[1:2 \cdot (i-1)+1]},$$

So, $\text{prefix}_{\tilde{m}_i^B}$ always has odd length.

Since H_{tr} is statistically binding with respect to the input length, and $\mathbf{p}_{g+1} \neq \tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})$ for any $i \in [L]$, the punctured point $\mathbf{p}_{g+1}$ is never queried during decryption procedure. Therefore, puncturing at $\mathbf{p}_{g+1}$ does not affect the decryption functionality.

Similarly, for any input with $|\tilde{\text{tr}}^A| \neq \eta j + \ell_{g+1}$, the encryption functionality remains unaffected. Hence, it suffices to consider the case where

$$|\tilde{\text{tr}}^A| = \eta j + \ell_{g+1}. \quad (7)$$

Observe that in obNext^B , all accepting inputs $\tilde{\text{tr}}^A$ that satisfy Eq. (7) share the same prefix—namely, the first j segments are identical. Thus, any two distinct accepting inputs must differ within the $(j+1)$ -th segment. Recall that in $\text{Hyb}_{j:g}^{(4)}$, the hash function H_{tr} is already changed to be statistically binding on $\widetilde{\text{seg}}_{j+1}^A$; these differing transcripts necessarily yield different hash values. Consequently, the punctured point $\mathbf{p}_{g+1}$ is never queried by any other accepting input, and puncturing at $\mathbf{p}_{g+1}$ does not affect the encryption functionality. Consequently, the puncturing at $\mathbf{p}_{g+1}$ does not affect the encryption or decryption behavior on any accepting transcript. Combining the above, $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$ and $\text{Next}_{\text{Hyb}_{j:g}^{(4)}}^B$ have the same functionality. $\square$

By Claim 7 and the sub-exponential security of iOM , we have that $\text{Hyb}_{j:g}^{(5)}$ and $\text{Hyb}_{j:g}^{(4)}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable.

$\text{Hyb}_{j:g}^{(6)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(5)}$ except for the following change. In Phase 3, if Sim sends str_{g+1} as the first ℓ_{g+1} bits of the $(j+1)$ th segment to P_B^* , then it will use r^* to decrypt the message $\tilde{m}_{\eta j + \ell_{g+1}}^B$ sent by P_B^* in round $\eta j + \ell_{g+1}$. Observe that this is merely a change of variables, and the two hybrids are functionally identical, and therefore $\text{Hyb}_{j:g}^{(6)} \equiv \text{Hyb}_{j:g}^{(5)}$.

$\text{Hyb}_{j:g}^{(7)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(6)}$ except that r^* is now sampled uniformly at random, rather than being computed as $F_{k_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B}))$. By the pseudorandomness of PPRF at the punctured point $\mathbf{p}_{g+1}$, it follows that $\text{Hyb}_{j:g}^{(7)}$ and $\text{Hyb}_{j:g}^{(6)}$ are $2^{-\lambda_2^\epsilon}$ -indistinguishable.

$\text{Hyb}_{j:g}^{(8)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(7)}$ except for the following change:

– Instead of sampling r^* uniformly at random, Sim now samples m^* uniformly at random and sets

$$r^* := m^* \oplus B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]})) .$$

Observe that this is merely a change of variables: the joint distribution of (r^*, m^*) remains identical to that in $\text{Hyb}_{j:g}^{(7)}$. Therefore, $\text{Hyb}_{j:g}^{(8)} \equiv \text{Hyb}_{j:g}^{(7)}$.

$\text{Hyb}_{j:g}^{(9)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(8)}$ except that when Sim sends str_{g+1} as the first ℓ_{g+1} bits of the $(j+1)$ th segment to P_B^* in [Phase 3](#), it decrypts $\tilde{m}_{\eta j + \ell_{g+1}}^B$ using a different but equivalent expression.

In $\text{Hyb}_{j:g}^{(9)}$, Sim computes

$$m_{\eta j + \ell_{g+1}}^B := \tilde{m}_{\eta j + \ell_{g+1}}^B \oplus m^* \oplus B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]})) .$$

Recall that in $\text{Hyb}_{j:g}^{(8)}$, Sim instead computes

$$m_{\eta j + \ell_{g+1}}^B = r^* \oplus \tilde{m}_{\eta j + \ell_{g+1}}^B, \quad \text{where} \quad r^* = m^* \oplus B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]})) .$$

Substituting the definition of r^* shows that the two expressions are identical. Therefore, $\text{Hyb}_{j:g}^{(9)} \equiv \text{Hyb}_{j:g}^{(8)}$.

$\text{Hyb}_{j:g}^{(10)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(9)}$ except for the following changes:

1. Define $\mathbf{p}_{g+1} := \tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_{\eta j + \ell_{g+1}}^B})$.
2. Sim computes $\mathbf{k}_{\text{dummy}}^* \leftarrow \text{Punc}(\mathbf{k}_{\text{dummy}}, \mathbf{p}_{g+1})$, where $\mathbf{k}_{\text{dummy}} \leftarrow \text{KGen}(1^\lambda)$.
3. Finally, Sim sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g}^{(10)}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g}^{(10)}}^B$ is defined in [Algo. 7.7](#). The only difference between [Algo. 7.7](#) and the previous obfuscated program is that $\mathbf{k}_{\text{dummy}}^*$ is hardcoded in place of $\mathbf{k}_{\text{dummy}}$.

Algorithm 7.7: Description of $\text{Next}_{\text{Hyb}_{j:g}^{(10)}}^B$ // identical to $\text{Next}_{\text{Hyb}_{j:g}^{(5)}}^B$, except for the modifications listed below in blue Constants. // same as before, with the following additional hard-coded values – ... – PPRF keys $\mathbf{k}_{\text{msg}}$, $\mathbf{k}_{\text{dummy}}$, $\mathbf{k}_{\text{dummy}}^*$, $\mathbf{k}_{\text{path}}$; – ...
--

By an argument similar to that in [Claim 7](#), [Algo. 7.7](#) does not change the functionality. Therefore, by the sub-exponential security of iOM , we have that $\text{Hyb}_{j:g}^{(10)}$ and $\text{Hyb}_{j:g}^{(9)}$ are $2^{-\lambda^\epsilon}$ -indistinguishable.

$\text{Hyb}_{j:g}^{(11)}$: This hybrid is identical to $\text{Hyb}_{j:g}^{(10)}$ except that $m^* := \mathbf{F}_{\mathbf{k}_{\text{dummy}}}(\mathbf{p}_{g+1})$. By the pseudorandomness of PPRF at the punctured point $\mathbf{p}_{g+1}$, it follows that $\text{Hyb}_{j:g}^{(11)}$ and $\text{Hyb}_{j:g}^{(10)}$ are $2^{-\lambda^\epsilon}$ -indistinguishable.

Recall that when Sim sends str_{g+1} as the first ℓ_{g+1} bits of the $(j+1)$ -th segment in [Phase 3](#), it uses the hardwired value m^* to decrypt $\tilde{m}_{\eta j + \ell_{g+1}}^B$ received from P_B^* . Since m^* is updated in this hybrid, Sim now decrypts $\tilde{m}_{\eta j + \ell_{g+1}}^B$ as

$$m_{\eta j + \ell_{g+1}}^B = \tilde{m}_{\eta j + \ell_{g+1}}^B \oplus \mathbf{F}_{\mathbf{k}_{\text{dummy}}}(\mathbf{p}_{g+1}) \oplus B(x_B^*, \text{Dec}_{\text{crs}}(\text{crs}_{[1:\eta j + \ell_{g+1}]}))$$

$\text{Hyb}_{j:g}^{(12)}$: Identical to $\text{Hyb}_{j:g}^{(11)}$ except that Sim sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g}^{(12)}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g}^{(12)}}^B$ is defined in [Algo. 7.8](#). At a high level, [Algo. 7.8](#) unpunctures $\mathbf{k}_{\text{dummy}}$ and no longer hardcodes m^* . Instead, the corresponding value is generated directly using $\mathbf{k}_{\text{dummy}}$.

Algorithm 7.8: Description of $\text{Next}_{\text{Hyb}_{j:g}}^B$

// identical to $\text{Next}_{\text{Hyb}_{j:g}}^B$, except for the modifications listed below in blue

Constants. // same as before, with the following additional hard-coded values

- PPRF keys $k_{\text{msg}}, k_{\text{dummy}}^*, k_{\text{dummy}}, k_{\text{path}};$
- m^*

Hard-coded Output:

- If $\ell = \ell_{g+1} + j\eta$ and $\tilde{\text{tr}}_{[\eta j + 1 : \eta j + \ell_{g+1}]}^A = \text{str}_{g+1}$, output m^* and HALT.

Step II: Decryption & Compute.

- If $\ell < \eta j + \ell_{g+1}$ or ($\ell = \eta j + \ell_{g+1}$ and $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_{g+1}] \leq \text{str}_{g+1}$) :
// all inputs that are smaller than or equal to str_{g+1} in shortlex order
 - For all $i \in [\ell]$, compute $\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$.
- Otherwise:
 - For $i \in [j \cdot \eta]$,
 - * Compute $m_i^A \leftarrow \text{Dec}_{k_{\text{crs}}}(\text{crs}[i])$
 - * Compute $\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
 - For $i \in [j\eta + 1 : j\eta + \ell_{g+1} - 1]$,
 - * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$
 - * Compute $\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
 - For $i = j\eta + \ell_{g+1}$,
 - * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$
 - * If $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_{g+1}] \leq \text{str}_{g+1}$,
 - Compute $\tilde{m}_i^B \leftarrow F_{k_{\text{dummy}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
 - * If $\tilde{\text{tr}}^A[\eta j + 1 : \eta j + \ell_{g+1}] = \text{str}_{g+1}$,
 - $\tilde{m}_i^B = m^*$
 - * else,
 - Compute $m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A)$
 - Compute $\tilde{m}_i^B \leftarrow m_i^B \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
 - For $i \in [j \cdot \eta + \ell_{g+1} + 1 : \ell]$,
 - * Compute $m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$
 - * Compute $m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A)$
 - * Compute $\tilde{m}_i^B \leftarrow m_i^B \oplus F_{k_{\text{msg}}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
- Output $\tilde{m}_\ell^B$

By an argument similar to that used in Claim 5, Algo. 7.8 does not change the functionality. Therefore, by the sub-exponential security of iOM, we have that $\text{Hyb}_{j:g}^{(12)}$ and $\text{Hyb}_{j:g}^{(11)}$ are $2^{-\lambda^\epsilon}$ -indistinguishable.

Observe that $\text{Hyb}_{j:g}^{(12)}$ and $\text{Hyb}_{j:g+1}$ are identical, so this completes the proof of Claim 4. Further observe that the sum of the distinguishing advantages from $\text{Hyb}_{j:0} \equiv \text{Hyb}_j$ to $\text{Hyb}_{j:2\eta+1-2} \equiv \text{Hyb}_j^{\text{out}}$ remains bounded by $2^{-\eta^2} = \text{negl}(\lambda)$, thus completing the proof of Claim 3.

7.1.2 Proof of $\text{Hyb}_j^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j+1}$

Claim 8. For every $j \in [0 : N - 1]$, $\text{Hyb}_j^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j+1}$

Proof. Consider the following sequence of hybrids $\text{Hyb}_{j:g}^{\text{out}}$ for $g \in [0 : \eta]$, where $\text{Hyb}_{j:0}^{\text{out}} := \text{Hyb}_j^{\text{out}}$ and $\text{Hyb}_{j:\eta}^{\text{out}} := \text{Hyb}_{j+1}$.

$\text{Hyb}_{j:g}^{\text{out}}$: For each $g \geq 1$, this hybrid is identical to $\text{Hyb}_{j:g-1}^{\text{out}}$ except for the following changes:

- In round $j\eta + g$ of [Phase 3](#), Sim sends $\tilde{m}_{j\eta+g}^A := F_{\text{kpath}}(j\eta + g)$ to P_B^* , instead of computing $\tilde{m}_{j\eta+g}^A$ honestly.
- Sim updates the obfuscated program by setting $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g}^{\text{out}}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g}^{\text{out}}}^B$ is defined in [Algo. 7.9](#). At a high level, in [Algo. 7.9](#), for every acceptable input $\tilde{\text{tr}}^A$ with $|\tilde{\text{tr}}^A| > (j+1)\eta$, the first g bits of the seg_{j+1}^A are taken from the decryption of crs rather than from the decryption of input transcript $\tilde{\text{tr}}^A$ itself.

Algorithm 7.9: Description of $\text{Next}_{\text{Hyb}_{j:g}^{\text{out}}}^B$

// identical to $\text{Next}_{\text{Hyb}_j^{\text{out}}}^B$, except for the modifications listed below in [blue](#)

...

Step I: Verify Input & Signatures. // same as in before, with the following additional check:

– ...

– Let $t = \min(n, j+1)$. Verify that

$$\tilde{\text{tr}}^A[1 : t\eta + g] \stackrel{?}{=} F_{\text{kpath}}(1 : t\eta + g).$$

If any check fails, output $\perp$.

Step II: Decryption & Compute.

– If $\ell \leq (j+1) \cdot \eta$:

- For all $i \in [\ell]$, compute

$$\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

– Otherwise:

- For $i \in [j\eta + g]$:

* Compute

$$m_i^A \leftarrow \text{Dec}_{\text{kcrs}}(\text{crs}[i]).$$

* Compute

$$\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

- For $i \in [j\eta + 1 + g : (j+1)\eta]$:

* Compute

$$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$$

* Compute

$$\tilde{m}_i^B \leftarrow F_{\text{kdummy}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$$

- For $i \in [(j+1) \cdot \eta + 1 : \ell]$:

* Compute

$$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{kmsg}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})).$$

* Compute

$$m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A).$$

* Compute

$$\tilde{m}_i^B \leftarrow m_i^B \oplus F_{\text{kmsg}}(\tau_0 \parallel \text{H}_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$$

– Output $\tilde{m}_\ell^B$.

Claim 9. For every $j \in [0 : N - 1]$ and $g \in [0 : \eta - 1]$, $\text{Hyb}_{j:g+1}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g}^{\text{out}}$.

Proof. We prove the claim via the following intermediate hybrids.

$\text{Hyb}_{j:g:0}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g}^{\text{out}}$ except that the $(g+1)$ th bit in $(j+1)$ th segment of crs is changed to

$$\text{crs}_{[j\eta+g+1]} = F_{\text{k}_{\text{crs}}}(j\eta + g + 1) \oplus m_{j\eta+g+1}^A$$

where the message $m_{j\eta+g+1}^A$ is computed as follows:

$$m_{j\eta+g+1}^A \leftarrow A(x_A, \text{tr}_{[1:j\eta+g]}^B) \quad (8)$$

where x_A is P_A 's private input and $\text{tr}_{[1:j\eta+g]}^B$ is computed as follows:

– For $i \in [1 : j\eta + g]$:

- Compute $\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$.
- Compute $m_i^B \leftarrow \tilde{m}_i^B \oplus F_{\text{k}_{\text{dummy}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})) \oplus B(x_A^*, \text{Dec}_{\text{k}_{\text{crs}}}(\text{crs}_{[1:i]}))$.

Note that to compute $\text{prefix}_{\tilde{m}_i^B}$, we also need to compute $\{\tilde{m}_j^A\}_{j \in [i]}$, which is computed as $\tilde{m}_j^A \leftarrow F_{\text{k}_{\text{path}}}(j)$.

By the pseudorandomness of PPRF at the punctured point, we conclude that $\text{Hyb}_{j:g:0}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g}^{\text{out}}$. Again, as mentioned in [Rmk. 1](#), to be more formal we should sandwich $\text{Hyb}_{j:g:0}^{\text{out}}$ between two hybrids for puncturing/unpuncturing k_{crs} .

$\text{Hyb}_{j:g:1}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:0}^{\text{out}}$ except that Sim switches H_{crs} to statistically binding on the $(j\eta + g + 1)$ th bit of crs . By the mode indistinguishability property of H_{crs} , it follows that $\text{Hyb}_{j:g:1}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:0}^{\text{out}}$.

$\text{Hyb}_{j:g:2}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:1}^{\text{out}}$ except that Sim switches H_{vk}^A to statistically binding on vk_{j+1}^A . By the mode indistinguishability property of H_{vk}^A , we obtain $\text{Hyb}_{j:g:2}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:1}^{\text{out}}$.

$\text{Hyb}_{j:g:3}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:2}^{\text{out}}$ except that Sim switches H_{sig}^A to statistically binding on the $(g+1)$ th bit of its input (out of the η bits). By the mode indistinguishability property of H_{sig}^A , we obtain $\text{Hyb}_{j:g:3}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:2}^{\text{out}}$.

$\text{Hyb}_{j:g:4}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:3}^{\text{out}}$ except that the verification key vk_{j+1}^A is programmed to accept only signatures on the value

$$h_{j+1}^{\text{sig}} := H_{\text{sig}}^A(\widetilde{\text{seg}}_{j+1}^A) \quad (9)$$

Formally, the simulator generates

$$\text{DPSig.Prog}(1^\lambda, h_{j+1}^{\text{sig}}) \mapsto (\text{sk}_{j+1}, \text{vk}_{j+1}^A). \quad (10)$$

where $\widetilde{\text{seg}}_{j+1}^A$ is computed as follows:

– For $i \in [j\eta + 1 : j\eta + g]$,

$$\tilde{m}_i^A \leftarrow F_{\text{k}_{\text{path}}}(i)$$

– For $i \in [j\eta + g + 1 : (j+1)\eta]$,

$$\tilde{m}_i^A \leftarrow m_i^A \oplus F_{\text{k}_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})) \quad \text{where} \quad m_i^A \leftarrow A(x_A, \text{tr}_{[1:i-1]}^B)$$

where the $\{\tilde{m}_{i^*}^B\}_{i^* \in [i-1]}$ contained in $\text{prefix}_{\tilde{m}_i^A}$, and the $\{m_{i^*}^B\}_{i^* \in [i-1]}$ contained in $\text{tr}_{[1:i-1]}^B$ are computed in the same way as $\text{Hyb}_{j+1:g:0}^{\text{out}}$.

By the mode indistinguishability property of DPSig , it follows that $\text{Hyb}_{j:g:4}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:3}^{\text{out}}$

Remark 2. If P_B^* does not send the honest output of obNext^B to Sim during **Phase 3**, then Sim may be unable to generate a valid signature for the $(j+1)$ -th segment $\widetilde{\text{seg}}_{j+1}^A$ since the verification key is programmed to accept only $H_{\text{sig}}^A(\widetilde{\text{seg}}_{j+1}^A)$ for a fixed $\widetilde{\text{seg}}_{j+1}^A$, while $\widetilde{\text{tr}}_{[j\eta+g+1:(j+1)\eta]}^A$ may still depend on the adversarial segment $\widetilde{\text{seg}}_{j+1}^B$, which need not correspond to an honest execution.

However, Sim is required to produce σ_{j+1}^A only after P_B^* provides a valid ZKAoK proof of honest behavior in **Step 3(b)i**. Therefore, this situation does not affect the correctness of the simulation.

$\text{Hyb}_{j:g:5}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:4}^{\text{out}}$ except for the following changes:

- Sim computes

$$m^* := m_{j\eta+g+1}^A \oplus F_{k_{\text{msg}}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_{j\eta+g+1}^A})) \quad (11)$$

where $m_{j\eta+g+1}^A$ is the value computed in [Eq. \(8\)](#) of $\text{Hyb}_{j:g:0}^{\text{out}}$.

- The programmed verification key vk_{j+1}^A only accepts signatures on $h_{j+1}^{\text{sig}} = H_{\text{sig}}^A(\widetilde{\text{seg}}_{j+1}^A)$, where $\tilde{m}_{j\eta+g+1}^A$ is replaced with m^* .
- If P_B^* does not abort and provides the honest output of obNext^B to Sim up to round $j\eta+g+1$ in **Phase 3**, then at round $j\eta+g+1$, Sim sends m^* to P_B^* .
- Sim sets $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ is defined in [Algo. 7.10](#). At a high level, [Algo. 7.10](#) hardcodes the value m^* . When it receives $\tilde{\text{tr}}^A$ as input and $|\tilde{\text{tr}}^A| > \eta \cdot (j+1)$, it checks whether $\tilde{m}_{j\eta+g+1}^A \stackrel{?}{=} m^*$ and aborts otherwise.

Algorithm 7.10: Description of $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ // identical to $\text{Next}_{\text{Hyb}_{j:g}^B}$, except for the modifications listed below in blue Constants. // same as before, with the following additional hard-coded values – ... – m^* ... Step I: Verify Input & Signatures. // same as in before, with the following additional check: – ... – If $\tilde{\text{tr}}^A > \eta(j+1)$, verify that $\tilde{\text{tr}}_{[j\eta+g+1]}^A \stackrel{?}{=} m^*$. If any check fails, output $\perp$. ...
--

Claim 10. $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ and $\text{Next}_{\text{Hyb}_{j:g}^B}$ have identical functionality.

Proof. Since H_{vk}^A is switched to a statistically binds on the $(j+1)$ th verification key, for any input $\{\text{vk}_i^A\}_{i \in [N]}$ to obNext^B , if vk_{j+1}^A differs from the value computed in [Eq. \(10\)](#), then

$$H_{\text{vk}}^A(\{\text{vk}_i^A\}_{i \in [N]}) \neq h_{\text{vk}}^A$$

Hence, for every accepting $\{\text{vk}_i^A\}_{i \in [N]}$, vk_{j+1}^A must equal the value computed in [Eq. \(10\)](#).

Furthermore, recall that in $\text{Hyb}_{j:g:4}^{\text{out}}$, the key vk_{j+1}^A is programmed to accept signatures only on the value h_{j+1}^{sig} (defined in Eq. (9)). In addition, H_{sig}^A is switched to a statistically binds on the $(g+1)$ th bit of its input. Therefore, for any input $\tilde{\text{tr}}^A$ satisfying

$$\tilde{\text{tr}}_{[j\eta+g+1]}^A \neq m^* \quad \text{and} \quad |\tilde{\text{tr}}^A| > \eta(j+1)$$

we must have

$$H_{\text{sig}}^A(\tilde{\text{tr}}_{[j\eta+1:(j+1)\eta]}^A) \neq h_{j+1}^{\text{sig}}$$

Consequently, no valid signature exist for such a transcript, and $\text{Next}_{\text{Hyb}_{j:g:4}^{\text{out}}}^B$ would also output $\perp$. Thus, the additional check in $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ does not change the set of accepting inputs, and the two programs $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ and $\text{Next}_{\text{Hyb}_{j:g:4}^{\text{out}}}^B$ have identical functionality. $\square$

Note that the behavior of Sim in Phase 3 and vk_{j+1}^A remains unchanged, since the value m^* is the same and this modification is merely a change of variables. Combined with Claim 10, the security of iOM implies that $\text{Hyb}_{j:g:5}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:4}^{\text{out}}$.

$\text{Hyb}_{j:g:6}^{\text{out}}$: Identical to $\text{Hyb}_{j:g:5}^{\text{out}}$ except that $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g:6}^{\text{out}}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g:6}^{\text{out}}}^B$ is defined in Algo. 7.11. That is, in Algo. 7.11, for any acceptable inputs $\tilde{\text{tr}}^A$ satisfying $|\tilde{\text{tr}}^A| > \eta(j+1)$, obfuscation derives $\text{tr}_{[j\eta+g+1]}^A$ as:

$$\text{tr}_{[j\eta+g+1]}^A = F_{\text{k}_{\text{crs}}}(j\eta + g) \oplus \text{crs}_{[j\eta+g]}$$

rather than

$$\text{tr}_{[j\eta+g+1]}^A \leftarrow \tilde{\text{tr}}_{[j\eta+g+1]}^A \oplus F_{\text{k}_{\text{msg}}}(\tau_0 \| H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$$

Algorithm 7.11: Description of $\text{Next}_{\text{Hyb}_{j:g:6}^{\text{out}}}^B$ <i>// identical to $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$, except for the modifications listed below in blue</i>	
...	
Step II: Decryption & Compute.	
– If $\ell \leq (j+1) \cdot \eta$:	• For all $i \in [\ell]$, compute
	$\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \ H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
– Otherwise:	
• For $i \in [j\eta + g+1]$:	
* Compute	
	$m_i^A \leftarrow \text{Dec}_{\text{k}_{\text{crs}}}(\text{crs}[i])$
* Compute	
	$\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \ H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
• For $i \in [j\eta + g+2 : (j+1)\eta]$:	
* Compute	
	$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{\text{k}_{\text{msg}}}(\tau_0 \ H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A}))$
* Compute	
	$\tilde{m}_i^B \leftarrow F_{\text{k}_{\text{dummy}}}(\tau_0 \ H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B}))$
• For $i \in [(j+1) \cdot \eta + 1 : \ell]$:	

* Compute	$m_i^A \leftarrow \tilde{m}_i^A \oplus F_{k_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^A})).$
* Compute	$m_i^B \leftarrow B(x_B, \text{tr}_{[1:i]}^A).$
* Compute	$\tilde{m}_i^B \leftarrow m_i^B \oplus F_{k_{\text{msg}}}(\tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_i^B})).$
– Output $\tilde{m}_\ell^B$.	

Claim 11. $\text{Next}_{\text{Hyb}_{j:g:6}^{\text{out}}}^B$ and $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ have the same functionality.

Proof. We distinguish two cases.

- If $\ell \leq (j+1)\eta$, both programs output dummy values by construction. Hence $\text{Next}_{\text{Hyb}_{j:g:6}^{\text{out}}}^B$ and $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ behave identically.
- If $\ell > (j+1)\eta$, consider any accepting input $\tilde{\text{tr}}^A$ to the obfuscated program. First observe that if

$$\tilde{\text{tr}}_{[j\eta+g+1]}^A \neq m^*,$$

then $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ outputs $\perp$. Hence, it suffices to consider inputs satisfying

$$\tilde{\text{tr}}_{[j\eta+g+1]}^A = m^*.$$

Recall that $\text{crs}_{[j\eta+g+1]}$ is programmed to encrypt $m_{j\eta+g+1}^A$ (see Eq. (8)), and m^* also encrypts $m_{j\eta+g+1}^A$ (see Eq. (11)). Moreover, H_{crs} is switched to be statistically binding on the $(j\eta + g + 1)$ -th bit of crs . Consider any alternative crs' given as input to the obfuscated program. If

$$\text{crs}'_{[j\eta+g+1]} \neq \text{crs}_{[j\eta+g+1]},$$

then by statistical binding we would have

$$H_{\text{crs}}(\text{crs}') \neq H_{\text{crs}}(\text{crs}),$$

and thus the input would be rejected. Therefore, for every accepting input crs' , it must hold that

$$\text{crs}'_{[j\eta+g+1]} = \text{crs}_{[j\eta+g+1]}.$$

Consequently, for all accepting inputs $(\tilde{\text{tr}}^A, \text{crs}')$, both $\tilde{\text{tr}}_{[j\eta+g+1]}^A$ and $\text{crs}'_{[j\eta+g+1]}$ encrypt the same plaintext value. Hence decrypting from either source yields the same result. It follows that $\text{Next}_{\text{Hyb}_{j:g:6}^{\text{out}}}^B$ and $\text{Next}_{\text{Hyb}_{j:g:5}^{\text{out}}}^B$ produce identical outputs on all accepting inputs.

Thus, the two programs have identical functionality. $\square$

By Claim 11 and the security of $i\text{OM}$, we have $\text{Hyb}_{j:g:6}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:5}^{\text{out}}$.

$\text{Hyb}_{j:g:7}^{\text{out}}$: Identical to $\text{Hyb}_{j:g:6}^{\text{out}}$ except for the following changes:

1. Let $\text{p}_{g+1} := \tau_0 \parallel H_{\text{tr}}(\text{prefix}_{\tilde{m}_{j\eta+g+1}^A})$;
2. Sim computes $r^* := F_{k_{\text{msg}}}(\text{p}_{g+1})$;
3. Sim computes $m^* \leftarrow m_{j\eta+g+1}^A \oplus r^*$, where $m_{j\eta+g+1}^A$ is defined in Eq. (8).
4. Sim computes $k_{\text{msg}}^* \leftarrow \text{Punc}(k_{\text{msg}}, \text{p}_{g+1})$, where $k_{\text{msg}} \leftarrow \text{KGen}(1^\lambda)$;
5. $\text{obNext}^B \leftarrow i\text{OM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g:7}^{\text{out}}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g:7}^{\text{out}}}^B$ is given in Algo. 7.12. The only difference between Algo. 7.12 and the previous obfuscated program is that k_{msg}^* is hardcoded in place of k_{msg} .

Algorithm 7.12: Description of $\text{Next}_{\text{Hyb}_{j:g:7}}^B$ // identical to $\text{Next}_{\text{Hyb}_{j:g:6}}^B$, except for the modifications listed below in blue**Constants.** // same as before, with the following additional hard-coded values

- ...
- PPRF keys k_{msg} , k_{msg}^* , k_{dummy} , k_{path} ;
- ...

By an argument similar to that in Claim 7, Algo. 7.12 does not change the functionality. Therefore, by the security of iOM , we have $\text{Hyb}_{j:g:7}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:6}^{\text{out}}$.

$\text{Hyb}_{j:g:8}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:7}^{\text{out}}$ except r^* is sampled uniformly at random, by the pseudorandomness at punctured points of PPRF, we have $\text{Hyb}_{j:g:8}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:7}^{\text{out}}$.

$\text{Hyb}_{j:g:9}^{\text{out}}$: This hybrid is identical to $\text{Hyb}_{j:g:8}^{\text{out}}$ except that the hardcoded value m^* is sampled uniformly at random and $r^* := m^* \oplus m_{j\eta+g+1}^A$, where $m_{j\eta+g+1}^A$ is defined in Eq. (8). Observe that this is merely a change of variables: the joint distribution (r^*, m^*) remains identical, we have $\text{Hyb}_{j:g:9}^{\text{out}} \equiv \text{Hyb}_{j:g:8}^{\text{out}}$.

$\text{Hyb}_{j:g:10}^{\text{out}}$: Identical to $\text{Hyb}_{j:g:9}^{\text{out}}$ except that $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g:10}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g:10}}^B$ is defined in Algo. 7.13. In Algo. 7.13, Sim unpunctures k_{msg} and punctures k_{path} at position $j\eta + g + 1$. By an argument similar to that in Claim 7, Algo. 7.13 does not change the functionality. Therefore, by the security of iOM , we have $\text{Hyb}_{j:g:10}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:9}^{\text{out}}$.

Algorithm 7.13: Description of $\text{Next}_{\text{Hyb}_{j:g:10}}^B$ // identical to $\text{Next}_{\text{Hyb}_{j:g:7}}^B$, except for the modifications listed below in blue**Constants.** // same as before, with the following additional hard-coded values

- ...
- PPRF keys k_{msg} , k_{msg}^* , k_{dummy} , k_{path} , k_{path}^* ;
- ...

$\text{Hyb}_{j:g:11}^{\text{out}}$: Identical to $\text{Hyb}_{j:g:10}^{\text{out}}$ except for the following changes:

- Sim sets $m^* := F_{k_{\text{path}}}(j\eta + g + 1)$ instead of sampling uniformly at random.
- $\text{obNext}^B \leftarrow \text{iOM}(1^\lambda, \text{Next}_{\text{Hyb}_{j:g:11}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g:11}}^B$ is given in Algo. 7.14.

Algorithm 7.14: Description of $\text{Next}_{\text{Hyb}_{j:g:11}}^B$ // identical to $\text{Next}_{\text{Hyb}_{j:g:10}}^B$, except for the modifications listed below in blue**Constants.** // same as before, with the following additional hard-coded values

- ...
- m^* , $F_{k_{\text{path}}}(j\eta + g + 1)$
- ...

Step I: Verify Input & Signatures.

- ...

– If $|\tilde{\text{tr}}^A| > \eta(j+1)$, verify that $\tilde{\text{tr}}_{[j\eta+g+1]}^A \stackrel{?}{=} F_{k_{\text{path}}}(j\eta+g+1)$.
 If any check fails, output $\perp$.
 ...

By pseudorandomness at punctured points of PPRF, we have $\text{Hyb}_{j:g:11}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:10}^{\text{out}}$.

Hyb_{j:g:12}^{out}: Identical to $\text{Hyb}_{j:g:11}^{\text{out}}$ except that $\text{obNext}^B \leftarrow \text{iOM}(\text{Next}_{\text{Hyb}_{j:g:12}^{\text{out}}}^B)$, where $\text{Next}_{\text{Hyb}_{j:g:12}^{\text{out}}}^B$ is given in [Algo. 7.15](#). At a high level, the obfuscated program unpunctures k_{path} and it no longer hardcodes the value m^* ; instead, the corresponding value is generated directly using k_{path} .

Algorithm 7.15: Description of $\text{Next}_{\text{Hyb}_{j:g:12}^{\text{out}}}^B$
 // identical to $\text{Next}_{\text{Hyb}_{j:g:11}^{\text{out}}}^B$, except for the modifications listed below in blue

Constants. // same as before, with the following additional hard-coded values

- ...
- PPRF keys $k_{\text{msg}}, k_{\text{dummy}}, k_{\text{path}}, k_{\text{path}}^*$;
- $F_{k_{\text{path}}}(j\eta+g+1)$.

...

Step I: Verify Input & Signatures.

- ...
- Let $t = \min(n, j+1)$. Verify that

$$\tilde{\text{tr}}^A[1 : t\eta + g+1] \stackrel{?}{=} F_{k_{\text{path}}}(1 : t\eta + g+1).$$

- ~~If $|\tilde{\text{tr}}^A| > \eta(j+1)$, verify that $\tilde{\text{tr}}_{[j\eta+g+1]}^A \stackrel{?}{=} F_{k_{\text{path}}}(j\eta+g+1)$.~~

If any check fails, output $\perp$.
 ...

It is easy to see that this does not change functionality. By the security of iOM , we have $\text{Hyb}_{j:g:12}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g:11}^{\text{out}}$.

Observe that $\text{Hyb}_{j:g:12}^{\text{out}}$ is identical to $\text{Hyb}_{j:g+1}^{\text{out}}$. By a hybrid argument over all intermediate hybrids, we obtain $\text{Hyb}_{j:g}^{\text{out}} \stackrel{c}{\approx} \text{Hyb}_{j:g+1}^{\text{out}}$, which completes the proof of [Claim 9](#). Since [Claim 9](#) implies [Claim 8](#), and [Claim 8](#) together with [Claim 3](#) implies [Claim 2](#), we conclude that $\text{REAL} \stackrel{c}{\approx} \text{Hyb}_N$, where N is the total number of segments. Finally, since Hyb_N is independent of x_A , it follows that [Prot. 6.1](#) is secure against static malicious adversaries. This completes the proof of [Lem. 1](#).

8 Generalizations

For sake of clarity, we chose to present our compiler for a protocol where parties are deterministic, they alternate by sending a *single* bit in each round, outputs are short (constant), and a short CRS is available. We now show how to remove each of these assumptions to obtain our main results.

Randomized protocols. To obtain a compiler for a randomized protocol $\langle A, B \rangle$ we use a PRG to derandomize it as follows. The parties run an augmented coin tossing protocol [[Gol01](#)] where P_A receives (s_A, d_A, C_B) and P_B receives (s_B, d_B, C_A) where s_A, s_B are short PRG seeds, C_A, C_B are their respective (binding) commitments, and d_A, d_B are corresponding decommitments information. The parties now run the protocol for a modified deterministic protocol $\langle A', B' \rangle$ where input A' is $x_A \| s_A$ and A' simply runs A with input x_A and randomness $r_A = \text{PRG}(s_A)$; B' is defined similarly. Note that the proof of “honest behavior” is now defined

for the algorithm A', B' and thus must prove that they are indeed using correct randomness (e.g., r_A for A which is done by including C_A in the statement and (s_A, d_A) in the witness).

Longer messages in each round. If the protocol consists of sending longer messages in each round (whose size may differ by round and by party, but can be determined given the size of each parties input), we simply adopt the following natural interpretation of segments and signatures: segment i of the transcript for party A is defined as the sequences of messages $(m_{i_1}, \dots, m_{i_k})$ sent by A where m_{i_1} is the first message sent by A after the last message of segment $i - 1$, along with corresponding proof of honest behavior and a signature, has been received from B ; and m_{i_k} is the last message such that the total length of all messages sent from i_1 to i_k is at most η . Therefore, if $m_{i_1} \geq \eta$, segment i simply consists of m_{i_1} ; otherwise it continues to include future consecutive messages of A until the length is up to η . Now, as before, A continues to prove for every segment that it behaved honestly and provides a signature to B before B will provide the corresponding response. Party B behaves symmetrically.

With this convention for defining the segments, we observe that the proof remains unmodified. The triangles in our proof will not be binary trees but may have many nodes at each level (2^{α_i} in particular where α_i is the length of A 's messages in round i when proving security for A). However, the size of the tree—in particular the number of nodes representing B 's responses that can be obtained from the obfuscated program run by B —is still at most 2^η . The proof thus proceeds the same way, modifying one node at a time within these trees. If parties have longer messages, they now just have to sign earlier than η rounds. This effectively robs the adversary from seeing more responses from obfuscation compared to the case where messages are single bits.

Longer Outputs. Recall that in [Prot. 6.1](#), P_A and P_B send each other their encrypted outputs, and parties execute Π_{dec} to receive their outputs. The communication incurred in Π_{dec} may be polynomial in the output length, so our rate claims do not generalize to longer outputs with this approach.

Our solution is to first modify the protocol in [Phase 3](#) so that P_A and P_B do not send each other encrypted outputs. Instead, after the L -th round, parties use their obfuscations to locally obtain their own outputs in encrypted form under a new derived key. More concretely, we modify Next^B so that, upon receiving a complete encrypted transcript $\tilde{\text{tr}}^A$ and verifying signatures on all segments (assume a signature is needed after the last message), it computes the output $y_B \leftarrow B(x_B, \text{tr}^A)$, derives key $k_B \leftarrow F_{k_{\text{msg}}}(B \| h_{\text{out}, L}^B)$, prepares a “pad” $\text{pad}_B := (F_{k_B}(i))_{i \in [|y_B|]}$, and outputs $\tilde{y}_B = y_B \oplus \text{pad}_B$. Next^A is modified analogously. [Phase 4](#) is completely replaced with the following. Parties execute a secure protocol for a function taking $(([k_{\text{msg}}]_A, \sigma_{[k_{\text{msg}}]_A}), ([k_{\text{msg}}]_B, \sigma_{[k_{\text{msg}}]_B}))$, shares of k_{msg} and corresponding signatures and verification keys as input, and outputting $(k_B, \text{Com}(k_A))$ to P_A , and $(k_A, \text{Com}(k_B))$ to P_B . P_A then sends pad_B along with a ZKAoK that pad_B was computed correctly. P_B acts symmetrically. After verifying proofs, parties decrypt their outputs with the received pads. Communication in this phase is $|y_A| + |y_B| + \text{poly}(\lambda)$, yielding constant rate.

In the security proof, we no longer have a simulator for Π_{dec} with which we can force the output (so we must delete hybrid $\mathcal{G}_5$). Our new strategy is to modify Next^B and “cheat” with the ZKAoK simulator in order to “swap” the values of pad_B and $\tilde{y}_B$. This way, P_B^* receives a dummy $\tilde{y}_B$ from obNext^B , but it still obtains y_B after decrypting with pad_B . In slightly more detail, after Hyb_N , invoke simulators for the 2PC protocol and ZKAoK mentioned just above. To swap the i -th bit of the output, first set H_{NM}^A to bind on $y_B[i]$. Then puncture¹³ $k_B^* \leftarrow \text{Punc}(k_B, i)$, define $r^* := F_{k_B}(i)$ and hardwire $\tilde{y}_B[i] = y_B[i] \oplus r^*$ in Next^B , which uses the hardwired $\tilde{y}_B[i]$ for the i -th bit of the output (also notice $\text{pad}_B[i] = r^*$). Switch r^* to random. Observe $(r^*, y_B \oplus r^*) \equiv (y_B \oplus r^*, r^*)$, so we can swap: $\text{pad}_B[i] = y_B[i] \oplus r^*$ and $\tilde{y}_B = r^*$. Finally, unpuncture k_B^* . Note again that we use ZKAoK's simulator (to prove pad_B is “correct”) before swapping with $\tilde{y}_B$.

Construction in the Plain Model. Lifting our construction to the plain model in order to achieve our main result is quite simple. At any point in [Phase 4](#) prior to [Step 1c](#), insert one round of communication where P_A sends a uniform string crs^A (this is used to protect A , so coin tossing is not required to generate

¹³ Technically, we have to first make k_B truly random by puncturing k_{msg} at point $B \| h_{\text{out}, L}^B$ using standard hybrids.

them uniformly). Likewise, P_B sends uniform crs^B . The length of crs^A is equal to the number of bits party B sends in the insecure protocol, and $|\text{crs}^A| + |\text{crs}^B| = \text{cc}$. Each party uses its received string as it would use crs in [Prot. 6.1](#), and our security proof against P_B^* uses crs^A as it would crs . Observe that removing the CRS only added cc bits to the compiled protocol, so rate remains constant.

Construction in the Random Oracle Model. As a final note, we can also achieve constant rate in the (programmable) random oracle model, under standard and falsifiable assumptions. This is done by using programmable random oracles RO_1 and RO_2 to instantiate crs and crs_{iOM} , respectively, where crs is the short CRS used in our construction, and crs_{iOM} is the long CRS used by the input-succinct TM obfuscator iOM of Badrinarayanan *et al.* [\[BFK⁺19\]](#). These CRS instantiations can be done at any point in [Phase 4](#) prior to [Step 1c](#) by coin tossing for a short random value ρ , and setting $\text{crs} \leftarrow \text{RO}_1(\rho)$, $\text{crs}_{\text{iOM}} \leftarrow \text{RO}_2(\rho)$. The iOM of Badrinarayanan *et al.* requires obfuscation keys generated by a setup algorithm taking crs_{iOM} and an input length bound in unary. Parties can generate these keys locally, and then enter Π_{setup} with these keys as private input. The resulting obfuscated programs take crs_{iOM} as an additional input; the rest of the protocol remains unmodified. The security proof also remains unmodified, modulo changes to $\text{crs}, \text{crs}_{\text{iOM}}$ are done by programming RO_1 and RO_2 .

References

- ABG⁺13. Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *Cryptology ePrint Archive*, 2013. [3](#)
- BBK⁺23. Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. Snargs for monotone policy batch np. In *Annual International Cryptology Conference*, pages 252–283. Springer, 2023. [10](#)
- BCP14. Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theory of cryptography conference*, pages 52–73. Springer, 2014. [3](#)
- BFK⁺19. Saikrishna Badrinarayanan, Rex Fernando, Venkata Koppula, Amit Sahai, and Brent Waters. Output compression, mpc, and io for turing machines. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 342–370. Springer, 2019. [3](#), [6](#), [38](#)
- BG01. Boaz Barak and Oded Goldreich. Universal arguments and their applications. *Cryptology ePrint Archive*, Paper 2001/105, 2001. [42](#)
- BGI⁺01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001. [1](#), [3](#)
- BK12. Zvika Brakerski and Yael Tauman Kalai. Efficient interactive coding against adversarial noise. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 160–166. IEEE, 2012. [2](#)
- BP04. Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 273–289. Springer, Berlin, Heidelberg, August 2004. [2](#)
- BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. [2](#)
- BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *International conference on the theory and application of cryptography and information security*, pages 280–300. Springer, 2013. [6](#)
- CCLY22. Nai-Hui Chia, Kai-Min Chung, Xiao Liang, and Takashi Yamakawa. Post-quantum simulatable extraction with minimal assumptions: Black-box and constant-round. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 533–563. Springer, Cham, August 2022. [41](#)
- 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, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Cham, August 2017. [3](#)
- CDG⁺24. Alessandro Chiesa, Marcel Dall’Agnol, Ziyi Guan, Nicholas Spooner, and Eylon Yogev. Untangling the security of kilian’s protocol: Upper and lower bounds. In *Theory of Cryptography Conference*, pages 158–188. Springer, 2024. [8](#)

- CHS18. Gil Cohen, Bernhard Haeupler, and Leonard J Schulman. Explicit binary tree codes with polylogarithmic size alphabet. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 535–544, 2018. [2](#)
- CMOS25. Michele Ciampi, Ankit Kumar Misra, Rafail Ostrovsky, and Akash Shah. Black-box constant-round secure 2pc with succinct communication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–389. Springer, 2025. [3](#)
- Coo75. Stephen A Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the seventh annual ACM symposium on Theory of computing*, pages 83–97, 1975. [3](#)
- Dam92. Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO’91*, volume 576 of *LNCS*, pages 445–456. Springer, Berlin, Heidelberg, August 1992. [2](#)
- DGMR22. Nico Döttling, Vipul Goyal, Giulio Malavolta, and Justin Raizes. Interaction-Preserving Compilers for Secure Computation. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:18, 2022. [2](#)
- FWW23. Cody Freitag, Brent Waters, and David J Wu. How to use (plain) witness encryption: registered abe, flexible broadcast, and more. In *Annual International Cryptology Conference*, pages 498–531. Springer, 2023. [10](#)
- Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009. [1](#), [3](#)
- GGH⁺13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. [1](#)
- GGHW17. Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. *Algorithmica*, 79(4):1353–1373, 2017. [3](#)
- GMS11. Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 768–777. IEEE, 2011. [2](#)
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. [8](#)
- Gol01. Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*, volume 2. Cambridge university press, 2001. [36](#), [42](#)
- GS18. Sanjam Garg and Akshayaram Srinivasan. A simple construction of io for turing machines. In *Theory of Cryptography*, volume 11240 of *Theory of Cryptography*, pages 425–454. Springer, 2018. [41](#)
- HW15. Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS ’15, page 163–172, New York, NY, USA, 2015. Association for Computing Machinery. [1](#), [2](#), [5](#), [40](#)
- IPS15. Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *Theory of Cryptography Conference*, pages 668–697. Springer, 2015. [3](#)
- JJ22. Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1023–1034. IEEE, 2022. [3](#)
- JJMP25. Abhishek Jain, Zhengzhong Jin, Surya Mathialagan, and Omar Paneth. On succinct obfuscation via propositional proofs. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1706–1743. IEEE, 2025. [3](#), [9](#)
- JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 60–73, New York, NY, USA, 2021. Association for Computing Machinery. [1](#), [3](#), [9](#)
- Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992. [8](#), [42](#), [43](#)
- Kil95. Joe Kilian. Improved efficient arguments: Preliminary version. In *Annual International Cryptology Conference*, pages 311–324. Springer, 1995. [8](#), [43](#)
- KLW15. Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC ’15, page 419–428, New York, NY, USA, 2015. Association for Computing Machinery. [6](#)

- KN96. Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1996. [1](#)
- Lin01. Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 171–189. Springer, Berlin, Heidelberg, August 2001. [42](#)
- Lin17. Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017. [43](#)
- MDS25. Yaohua Ma, Chenxin Dai, and Elaine Shi. Quasi-linear indistinguishability obfuscation via mathematical proofs of equivalence and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 157–186. Springer, 2025. [3](#)
- Nao03. Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Berlin, Heidelberg, August 2003. [2](#)
- NN01. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 590–599, 2001. [1](#), [8](#), [42](#)
- PF79. Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, April 1979. [10](#)
- PW09. Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 403–418. Springer, Berlin, Heidelberg, March 2009. [42](#)
- QWW18. Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 859–870. IEEE, 2018. [3](#)
- Sch93. Leonard J Schulman. Deterministic coding for interactive communication. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 747–756, 1993. [2](#)
- Sch96. L.J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. [2](#)
- SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014. [41](#)
- WW24. Brent Waters and David J Wu. Adaptively-sound succinct arguments for np from indistinguishability obfuscation. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 387–398, 2024. [9](#)
- Yan88. Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 223–228, 1988. [1](#)

A Additional Preliminaries

A.1 Somewhere Statistically Binding Hash

Definition 5 (Somewhere Statistically Binding (SSB) Hash [HW15]). A SSB hash consists of PPT algorithms $(\text{Gen}, H, \text{Open}, \text{Verify})$ along with a block alphabet $\Sigma = \{0, 1\}^{\ell_{\text{blk}}}$, output size ℓ_{hash} and opening size ℓ_{opn} , where $\ell_{\text{blk}}(\lambda), \ell_{\text{hash}}(\lambda), \ell_{\text{opn}}(\lambda)$ are some fixed polynomials in the security parameter. The algorithms have the following syntax:

- $\text{hk} \leftarrow \text{Gen}(1^\lambda, L, i)$ takes as input an integer $L \leq 2^\lambda$ and index $i \in \{0, \dots, L-1\}$ (both in binary) and outputs a public hashing key hk .
- $H_{\text{hk}} : \Sigma^L \rightarrow \{0, 1\}^{\ell_{\text{hash}}}$ is a deterministic polynomial time algorithm that takes as input $x = (x[0], \dots, x[L-1]) \in \Sigma^L$ and outputs $H_{\text{hk}}(x) \in \{0, 1\}^{\ell_{\text{hash}}}$.
- $\pi \leftarrow \text{Open}(\text{hk}, x, j)$: Given the hash key hk , $x \in \Sigma^L$, and an index $j \in \{0, \dots, L-1\}$, creates an opening $\pi \in \{0, 1\}^{\ell_{\text{opn}}}$.
- $\text{Verify}(\text{hk}, y, j, u, \pi)$: Given a hash key hk and $y \in \{0, 1\}^{\ell_{\text{hash}}}$, an integer index $j \in \{0, \dots, L-1\}$, a value $u \in \Sigma$, and an opening $\pi \in \{0, 1\}^{\ell_{\text{opn}}}$, outputs a decision $\in \{\text{accept}, \text{reject}\}$. This is intended to verify that a pre-image x of $y = H_{\text{hk}}(x)$ has $x[j] = u$.

We require the following properties:

- **Correctness:** for any integers $L \leq 2^\lambda$ and $i, j \in \{0, \dots, L-1\}$, any $\text{hk} \leftarrow \text{Gen}(1^\lambda, L, i)$, $x \in \Sigma^L$, $\pi \leftarrow \text{Open}(\text{hk}, x, j)$: we have $\text{Verify}(\text{hk}, H_{\text{hk}}(x), j, x[j], \pi) = \text{accept}$.
- **Index Hiding:** We consider the following game between an attacker $\mathcal{A}$ and a challenger:
 - $\mathcal{A}(1^\lambda)$ chooses an integer L and two indices $i_0, i_1 \in \{0, \dots, L-1\}$.
 - The challenger chooses a bit $b \leftarrow \{0, 1\}$ and sets $\text{hk} \leftarrow \text{Gen}(1^\lambda, L, i_b)$.
 - $\mathcal{A}(\text{hk})$ outputs a bit b' .

We require that for any PPT attacker $\mathcal{A}$ we have $\left| \Pr [b = b'] - \frac{1}{2} \right| \leq \text{negl}(\lambda)$ in the above game.

- **Somewhere Statistically Binding:** We say that hk is statistically binding for an index i if there do not exist any values $y, u \neq u', \pi, \pi'$ such that $\text{Verify}(\text{hk}, y, i, u, \pi) = \text{Verify}(\text{hk}, y, i, u', \pi') = \text{accept}$. We require that for any integers $L \leq 2^\lambda, i \in \{0, \dots, L-1\}$ the key $\text{hk} \leftarrow \text{Gen}(1^\lambda, L, i)$ is statistically binding for i .

A.2 Puncturable Pseudorandom Function

Definition 6 (Puncturable Pseudorandom Function (PPRF) [SW14, GS18]). A puncturable pseudorandom function PPRF is a tuple of PPT algorithms $(\text{KGen}, \text{F}, \text{Punc})$ with the following properties:

- **Efficiently Computable:** For all $\lambda \in \mathbb{N}$ and for all $\text{k} \leftarrow \text{KGen}(1^\lambda)$, $\text{F}_{\text{k}} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ is a polynomial time computable.
- **Functionality is preserved under puncturing:** For all $\lambda \in \mathbb{N}$, for all $y \in \{0, 1\}^\lambda$, and $\forall x \neq y$,

$$\Pr [\text{F}_{\text{k}\{y\}}(x) = \text{F}_{\text{k}}(x)] = 1$$

- **Pseudorandomness at punctured points:** For all $\lambda \in \mathbb{N}$, for all $y \in \{0, 1\}^\lambda$, and for all poly-sized adversaries $\mathcal{A}$,

$$\left| \Pr [\mathcal{A}(\text{F}_{\text{k}}(y), \text{k}\{y\}) = 1] - \Pr [\mathcal{A}(U_\lambda, \text{k}\{y\})] \right| \leq \text{negl}(\lambda)$$

where $\text{k} \leftarrow \text{KGen}(1^\lambda)$, $\text{k}\{y\} \leftarrow \text{Punc}(\text{k}, y)$ and U_λ denotes the uniform distribution over $\{0, 1\}^\lambda$.

A.3 Commitment Schemes

The following definitions are based on those in [CCLY22].

Definition 7 (Commitment). A commitment scheme is a pair of PPT ITMs $\langle C, R \rangle$ that satisfies the following. Let $m \in \{0, 1\}^{\ell(\lambda)}$ (where $\ell(\cdot)$ is some polynomial) is a message that C wants to commit to. The protocol consists of the following stages:

- **Commit Stage:** $C(m)$ and R interact with each other to generate a transcript τ , a decommitment **decom** as C 's private output, and a decision value $b_{\text{com}} \in \{\top, \perp\}$ as R 's private output. We denote this execution by $(\tau, \text{decom}, b_{\text{com}}) \leftarrow \langle C(m), R \rangle(1^\lambda)$.
- **Decommit Stage:** C sends the committed message m and the decommitment **decom** to R , and R outputs decision $b_{\text{dec}} \in \{\top, \perp\}$. We denote this execution by $b_{\text{dec}} \leftarrow \text{Verify}(\tau, m, \text{decom})$.¹⁴ W.l.o.g., we assume that whenever R rejects at the end of the commit stage (i.e., $b_{\text{com}} = \perp$), R will reject at the end of decommit stage (i.e., $b_{\text{dec}} = \perp$). (Note that w.l.o.g., one can think that b_{com} is included in τ , because we can always modify the protocol to ask R to send b_{com} as the last-round message of the commit stage.)

The scheme satisfies the following completeness requirement:

¹⁴ We assume that there is no state information for R kept from the commit stage.

1. **Completeness.** For any polynomial $\ell : \mathbb{N} \rightarrow \mathbb{N}$ and any $m \in \{0, 1\}^{\ell(\lambda)}$, it holds that

$$\Pr \left[b_{\text{com}} = b_{\text{dec}} = \top : \begin{array}{l} (\tau, \text{decom}, b_{\text{com}}) \leftarrow \langle C(m), R \rangle(1^\lambda) \\ b_{\text{dec}} \leftarrow \text{Verify}(\text{com}, m, \text{decom}) \end{array} \right] = 1.$$

Definition 8 (Computationally Hiding). A commitment scheme $\langle C, R \rangle$ is computationally hiding if for any non-uniform PPT receiver R^* and any polynomial $\ell : \mathbb{N} \rightarrow \mathbb{N}$, the following holds for all $\lambda \in \mathbb{N}$, $m_0, m_1 \in \{0, 1\}^{\ell(\lambda)}$:

$$\{\text{OUT}_{R^*} \langle C(m_0), R^* \rangle(1^\lambda)\}_{\lambda, m_0, m_1} \stackrel{c}{\approx} \{\text{OUT}_{R^*} \langle C(m_1), R^* \rangle(1^\lambda)\}_{\lambda, m_0, m_1},$$

where $\text{OUT}_{R^*} \langle C(m_b), R^* \rangle(1^\lambda)$ ($b \in \{0, 1\}$) denotes the output of R^* at the end of the commit stage.

Definition 9 (Statistically Binding). A commitment scheme $\langle C, R \rangle$ is statistically binding if for any unbounded-time committer C^* , the following probability is negligible in λ :

$$\Pr \left[\begin{array}{l} \exists m_0 m_1, \text{decom}_0, \text{decom}_1, \text{ s.t. } m_0 \neq m_1 \wedge \\ \text{Verify}(\tau, m_0, \text{decom}_0) = \top \wedge \\ \text{Verify}(\tau, m_1, \text{decom}_1) = \top \end{array} : (\tau, \text{decom}, b_{\text{com}}) \leftarrow \langle C^*, R \rangle(1^\lambda) \right]$$

A.4 Extractable Commitments

The following is based on [PW09].

Definition 10 (Succinct Extractable Commitment). A commitment $\text{ECom} = \langle C, R \rangle$ is extractable if there exists an expected polynomial-time probabilistic oracle machine (the extractor) $\mathcal{E}$ that given oracle access to any PPT cheating committer C^* outputs a pair (τ, σ^*) such that:

- **Simulation:** τ is identically distributed to the view of C^* at the end of interacting with an honest receiver R in commitment phase.
- **Extraction:** the probability that τ is accepting and $\sigma^* = \perp$ is negligible.
- **Binding:** if $\sigma^* \neq \perp$, then it is computationally infeasible to open τ to any value other than σ^* .

Additionally, we call ECom a succinct extractable commitment scheme if it satisfies the following property

- **Succinctness:** the total communication incurred during a protocol execution $\langle C(m), R \rangle$ is at most $\text{poly}(\lambda, \log(|m|))$ for security parameter λ and message m .

Constructions for succinct extractable commitments are known from [Kil92, NN01, BG01].

A.5 Interactive Arguments

The following definition is adapted from [Lin01, Gol01].

Definition 11 (Succinct Zero-Knowledge Interactive Arguments of Knowledge). An interactive argument for an NP language $\mathcal{L}$ with corresponding relation $\mathcal{R}$ consists of an interactive protocol between a prover P and a verifier V . We use $\langle P(w), V \rangle(x) = b$ to denote that V outputs bit b when parties enter the protocol with common input x , and P with private input w . We require interactive arguments to satisfy completeness and soundness properties defined as follows.

- **Completeness.** For every $(x, w) \in \mathcal{R}$ it holds that

$$\Pr [\langle P(w), V \rangle(x) = 1] = 1$$

- **Soundness.** For any non-uniform PPT prover P^* , there exists a negligible function negl such that for any $x, z \in \{0, 1\}^*$ (where z represents auxiliary input and random coins for P^*), it holds that

$$\Pr [\langle P_z^*, V \rangle(x) = 1 \mid x \notin \mathcal{L}] \leq \text{negl}(|x|)$$

We call $\langle P, V \rangle$ a succinct zero knowledge interactive argument of knowledge if it satisfies the following three additional properties:

- **Knowledge Soundness.** There exists a probabilistic oracle machine $\mathcal{E}$ and a polynomial $q(\cdot)$ such that, for every non-uniform probabilistic expected polynomial-time machine P^* , and every $x, z \in \{0, 1\}^*$, then $\mathcal{E}^{P_z^*(\cdot)}(x) \in \mathcal{R}(x)$, and the expected runtime of $\mathcal{E}^{P_z^*(\cdot)}(x)$ is

$$\frac{q(|x|)}{p(x, z) - \kappa(|x|)}$$

where $p(x, z) := \Pr [\langle P_z^*, V \rangle(x) = 1] > \kappa(|x|)$, and $\kappa(\cdot)$ is the knowledge soundness error.

- **Zero Knowledge.** For every PPT V^* there exists a PPT Sim such that

$$\{\text{view}_{V^*}(x)\}_{x \in \mathcal{L}} \stackrel{c}{\approx} \{\text{Sim}(x)\}_{x \in \mathcal{L}}$$

where $\text{view}_{V^*}(x)$ is the distribution of the random coins and messages received by V^* during execution of $\langle P, V^* \rangle(x)$.

- **Succinctness.** The total communication incurred during a protocol execution $\langle P(w), V \rangle(x)$ is at most $\text{poly}(\lambda, \log(|w|))$ for security parameter λ .

Constructions for succinct ZKAoK are known from [Kil92, Kil95].

A.6 Fully Homomorphic Encryption

Definition 12 (Fully Homomorphic Encryption). A fully homomorphic encryption scheme consists of a tuple of PPT algorithms $\text{FHE} = (\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$ with the following syntax:

- $\text{Gen}(1^\lambda) \rightarrow (\text{sk}, \text{pk})$: On input security parameter 1^λ , output a key pair (sk, pk) .
- $\text{Enc}(\text{pk}, m) \rightarrow c$: On input public key pk and message m , output a ciphertext c .
- $\text{Eval}(\text{pk}, f, c) \rightarrow c^*$: On input a public key pk , function description f , and ciphertext c , output a ciphertext c^* .
- $\text{Dec}(\text{sk}, c) \rightarrow m$: On input secret key sk and ciphertext c , output a plaintext message m .

satisfying the following properties:

- **Encryption Correctness.** For any $(\text{sk}, \text{pk}) \leftarrow \text{Gen}(1^\lambda)$ and any message m , it holds that $\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, m)) = m$.
- **Evaluation Correctness.** For any $(\text{sk}, \text{pk}) \leftarrow \text{Gen}(1^\lambda)$ and any ciphertext c , it holds that $\text{Dec}(\text{sk}, \text{Eval}(\text{pk}, f, c)) = f(\text{Dec}(\text{sk}, c))$.
- **Semantic Security.** For any $\lambda \in \mathbb{N}$, for all pairs of messages m_0, m_1 , and for all poly-sized $\mathcal{A}$, there exists a negligible function negl such that:

$$\Pr \left[\mathcal{A}(\text{pk}, \text{Enc}(\text{pk}, m_b)) = b \mid \begin{array}{l} (\text{sk}, \text{pk}) \leftarrow \text{Gen}(1^\lambda) \\ b \xleftarrow{\$} \{0, 1\} \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

A.7 Secure Two-Party Computation

The following definitions for secure two-party computation are all adapted from Lindell [Lin17] as we only consider the standalone setting.

Definition 13 (The Ideal Execution Model). The ideal execution $\text{IDEAL}_{f, \mathcal{A}(z), i}(x_A, x_B, \lambda)$ for a two-party functionality $f = (f_A, f_B)$, with a non-uniform PPT adversary $\mathcal{A}$ accepting auxiliary input z and corrupting party $P_i \in \{P_A, P_B\}$, on input x_A for party P_A , input x_B for party P_B , and security parameter 1^λ common to all entities, is defined as the output pair of the honest party and $\mathcal{A}$ from the following execution:

- P_A and P_B send inputs (x'_A, x'_B) to a trusted party. Depending on z and the input x_i of the corrupted party P_i , $\mathcal{A}$ may choose to send x'_i as an abort message, the prescribed input x_i , or some other string. If the trusted party receives an abort message, it forwards an abort message to the honest party P_j and the execution terminates. The honest party always sends its prescribed input.
- The trusted party computes $f(x'_A, x'_B) = (f_A(x'_A), f_B(x'_B))$ and sends $f_i(x'_A, x'_B)$ to corrupted party P_i .
- $\mathcal{A}$ sends either a continue or an abort message to the trusted party. If the trusted party receives an abort message, it forwards an abort message to the honest party P_j . Else, the trusted party sends $f_j(x'_A, x'_B)$ to P_j .
- The honest party outputs the value received from the trusted party. The adversary $\mathcal{A}$ outputs any arbitrary PPT computable function of $x_i, z, f_i(x'_A, x'_B)$.

Definition 14 (The Real Execution Model). The real execution $\text{REAL}_{\Pi, \mathcal{A}(z), i}(x_A, x_B, \lambda)$ of a protocol Π for a two-party functionality f , with a non-uniform PPT adversary $\mathcal{A}$ accepting auxiliary input z and corrupting party $P_i \in \{P_A, P_B\}$ is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the real execution of Π , where P_A has input x_A , P_B has input x_B , and all entities share common security parameter 1^λ . In the case where both parties are honest, this output pair is $(f_A(x_A, x_B), f_B(x_A, x_B))$.

Definition 15 (Maliciously Secure Two-Party Computation). Let f be a two-party functionality and let Π be a two-party protocol that computes f . Protocol Π is said to securely compute f with abort in the presence of static malicious adversaries if for every non-uniform PPT adversary $\mathcal{A}$ for the real model, there exists a non-uniform PPT adversary Sim for the ideal model, such that for every $i \in \{A, B\}$, $x_A, x_B \in \{0, 1\}^*$ with $|x_A| = |x_B|$, $z \in \{0, 1\}^*$, and $\lambda \in \mathbb{N}$,

$$\{\text{IDEAL}_{f, \text{Sim}(z), i}(x_A, x_B, \lambda)\}_{x_A, x_B, z, \lambda} \stackrel{c}{\approx} \{\text{REAL}_{\Pi, \mathcal{A}(z), i}(x_A, x_B, \lambda)\}_{x_A, x_B, z, \lambda}$$