Incremental Program Obfuscation[*]

Sanjam Garg
University of California, Berkeley
sanjamg@berkeley.edu

Omkant Pandey
University of California, Berkeley
omkant@berkeley.edu

Abstract

Recent advances in program obfuscation suggest that it is possible to create software that can provably safeguard secret information. However, software systems usually contain large executable code that is updated multiple times and sometimes very frequently. Freshly obfuscating the program for every small update will lead to a considerable efficiency loss. Thus, an extremely desirable property for obfuscation algorithms is *incrementality*: small changes to the underlying program translate into small changes to the corresponding obfuscated program.

We initiate a thorough investigation of *incremental program obfuscation*. We show that the strong simulation-based notions of program obfuscation, such as "virtual black-box" and "virtual grey-box" obfuscation, cannot be incremental even for very simple functions such as point functions. We then turn to the indistinguishability-based notions, and present two security definitions of varying strength. To understand the overall strength of our definitions, we formulate the notion of *incremental best-possible obfuscation* and show that it is equivalent to our (stronger) indistinguishability-based notion.

Finally, we present constructions for incremental program obfuscation satisfying both our security notions. We first give a construction achieving the weaker security notion based on the existence of general purpose indistinguishability obfuscation. Next, we present a generic transformation using *oblivious RAM* to amplify security from weaker to stronger, while maintaining the incrementality property.

1 Introduction

Program obfuscation is the process of transforming a computer program into an "unintelligible" one while preserving its functionality. Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI+12] formulated several notions for program obfuscation, and demonstrated that the strongest form of obfuscation, called *virtual black-box* (VBB) obfuscation, is impossible in general. In contrast, a recent work of Garg, Gentry, Halevi, Raykova, Sahai and Waters [GGH+13] presents an obfuscation mechanism for general programs that achieves the notion of *indistinguishability obfuscation*. Indistinguishability obfuscation, or IO, is a weaker form of obfuscation than VBB; nevertheless, it results in best possible obfuscation [GR07].

The feasibility of general purpose obfuscation, in principle, allows the creation of software that can provably safeguard secret information, e.g., cryptographic keys, proprietary algorithms, and so on. A typical software, however, can be quite complex with millions of lines of executable code

[*]Research supported in part from a DARPA/ARL SAFEWARE award, AFOSR Award FA9550-15-1-0274, and NSF CRII Award 1464397. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

[IB]. Once installed, it may go through frequent updates, e.g., when new features are added or security vulnerabilities are discovered. If cryptographic obfuscation is used to protect the software, it must be possible to quickly update the *obfuscated* software when new updates become available. In particular, if the original program is large, the obfuscating it from scratch for every new update would be prohibitive. Furthermore, in various client-server settings, where the obfuscated software resides on a networked machine, transmitting the entire (updated) software would be a bottleneck. Ideally, the effort to update the obfuscated program should only be proportional to the changes made to the the original unobfuscated program.

These issues are not unique to program obfuscation and analogous problems have been considered in several other settings. Bellare, Goldreich and Goldwasser [BGG94] introduced and developed the notion of incremental cryptography, first in the context of hashing and digital signatures. The idea is that, once we have signed a document D, signing new versions of D should be rather quick. For example, if we only flip a single bit of D, we should be able to update the signature in time polynomial in $\log |D|$ (instead of $|D|$) and the security parameter λ. Incrementality is an attractive feature to have for many cryptographic primitive such as encryption, signatures, hash functions, and so on [BGG95, Mic97, Fis97a, BM97, BKY01, MPRS12].

If we want to obfuscate large programs that are frequently updated, then it is crucial to have *incremental* program obfuscation. Current program obfuscation methods are prohibitively slow for application to large programs. However as obfuscation becomes more efficient, it is clear that incrementality will be essential for deployment. This line of investigation is particularly important because of its relevance to the deployment (as opposed to security) of obfuscated software.

1.1 Our Contributions

In this work, we initiate the study of *incremental* program obfuscation. Here we ask what is the right security definition for incremental program obfuscation and if it can be realized. We show that enhancing obfuscation with incrementality must come at the cost of some degradation in security. On the positive side, we realize the *best-possible* incremental obfuscation. More specifically, our incremental obfuscation method hides as much about the original program as any other incremental obfuscation of a given size. Our results are generic in the sense that starting with a general purpose obfuscation method for circuit, Turing Machines or RAM programs, we obtain general purpose incremental obfuscation for the same class of programs, with only a polynomial loss in the security reduction. Next we provide more details.

Modeling incremental obfuscation. We model an incremental obfusctaion method by an obfuscation procedure and an Update procedure. In this setting, a large program P is obfuscated and placed on a remote machine in the obfuscated form $\widetilde{P}$. Update can query any bit of the obfuscated program $\widetilde{P}$. The input to Update is a set S of indices, indicating which bits of P should be "updated." For every index $i \in S$, there is an associated update *operation* $f_i : \{0,1\} \to \{0,1\}$ to be performed on the ith bit of P; the set of these operations is denoted by $F_S = \{f_i\}_{i \in S}$. The output of Update consists of a set $\widetilde{S}$ indicating the bits of $\widetilde{P}$ to be updated; for every $i \in \widetilde{S}$ there is an associated bit b_i indicating the updated value.

The *incrementality* of an obfuscation scheme, denoted Δ, is defined to be the *running time* of Update in the worst case with respect to *singleton* sets S. This is a robust measure since it tells us how bad Update can be when we just want to update one bit.[1]

[1]Another natural way to define incrementality is the running time of Update for S, divided by $|S|$ in the worst case, taken over all (S, F_S). The single bit measure implies this definition because for larger sets S, one can simply

An important observation is that the Update cannot be a public algorithm, since otherwise, one can "tamper" with the $\widetilde{P}$ and potentially recover the entire P. Therefore, Update must require a *secret key*, generated at the time of obfuscation of P, to be able to make updates.

In this work, we require that the size of the updated obfuscated program should be the same as the fresh obfuscation of the new (updated) program. Further strengthening this requirement: we require that an updated obfuscation of a program should look *computationally indistinguishable* from a freshly generated obfuscation of the new program. We actually achieve statistical indistinguishability, and refer to this property as *pristine updates.* These requirements are crucial for various applications.

Lower bound for incremental VBB and VGB obfuscation. We show that incrementality significantly interferes with the security an obfuscation method can offer. More specifically, we argue that for the family of point functions, every incremental VBB obfuscation scheme must have incrementality $\Delta \in \Omega(n)$ where n is the size of the point function program. In fact, our result holds also for the weaker notion of *virtual grey box* (VGB) obfuscation, introduced by Bitanksy and Canetti [BC10]. VGB obfuscation, like VBB, is a simulation-based notion of security but it allows the simulator to be unbounded; it only requires that the number of queries made by the simulator to the black-box program should be polynomial. This further strengthens our result.

Our lower bound proceeds by proving that every incremental VGB (and hence VBB) scheme must "leak" the Hamming distance between the obfuscated and updated programs. Therefore, no generic compiler can preserve obfuscation quality and provide low incrementality at the same time.

Interestingly, our negative result holds even if only one update is made to the obfuscated point function. This is somewhat surprising since point-functions are inherently related to deterministic encryption [BS16] for which incremental schemes are known for single updates [MPRS12].

Positive results for indistinguishability obfuscation. Our lower bound motivates us to look more deeply into the definitions for incremental indistinguishability obfuscation, or IIO. We aim to obtain high flexibility in addition to as strong security as possible.

With this goal, we consider an obfuscated program $\widetilde{P}$ which is continually updated over time, by applying the Update algorithm to the previous obfuscation. This results in a sequence of obfuscated programs $\widetilde{P}, \widetilde{P}^1, \ldots, \widetilde{P}^t$ where $\widetilde{P}^i$ is obtained from $\widetilde{P}^{i-1}$ using Update. Since the adversary can view the entire sequence of obfuscated programs, this suggests the following natural definition: for every pair of programs (P_0, P_1) and every sequence of increments (or updates) $I = (S_1, \ldots, S_t)$ of arbitrary polynomial size sets S_i and for arbitrary polynomial t, the distributions $X_0 := (\widetilde{P}_0, \widetilde{P}_0^1, \ldots, \widetilde{P}_0^t)$ and $X_1 := (\widetilde{P}_1, \widetilde{P}_0^1, \ldots, \widetilde{P}_1^t)$ should be computationally indistinguishable provided that P_0^i and P_1^i are functionally equivalent for every i where P_b^i is the program corresponding to $\widetilde{P}_b^i$ for every $b \in \{0,1\}$.

We use this definition as the default definition of IIO. We present an IIO scheme for the class of all circuits with incrementality $\mathsf{poly}(\lambda, \log |P|)$, assuming the existence of IO for the same class.

Increment-privacy and "best possible" incremental obfuscation. The IIO definition does not necessarily hide the sequence I of increments. Indeed, by looking at the sequence of updated programs, an adversary might be able to recover which bits were updated at each time step. For many application this could be a serious issue. For example, if $\widetilde{P}$ is updated to fix a security vulnerability in P, by looking at which bits were updated, a hacker might be able to discover the vulnerability; it can then exploit other machines where the program is still awaiting update.

apply Update one by one for each index in S, and still achieve same incrementality.

Even more importantly, since IIO may leak the sequence I, it is unlikely to achieve our desired "best possible" incremental obfuscation. We therefore consider a strengthened definition which hides the sequence I, and call it *increment-private* IIO. More specifically, we consider two sequences I_0, I_1 where I_0 is applied to $\widetilde{P}_0$ and I_1 to $\widetilde{P}_1$. As before, we require that the resulting obfuscation sequences should look indistinguishable provided that the underlying programs are functionally equivalent at every time step.

With the goal of realizing obfuscation satisfying this security notion, we show a transformation which converts any IIO scheme into increment-private IIO scheme without affecting its incrementality too much. Our transformation is obtained by combining IIO with *oblivious RAM* (ORAM) programs [GO96].

Finally, in an effort to better understand what is the strongest possible incremental obfuscation, we define the notion of *incremental best-possible obfuscation*, or IBPO. For example, our lower bound for VGB/VBB obfuscations demonstrates that any incremental scheme must leak the size of incremental updates that take place. It is important to understand if this is all that is leaked. Our IBPO notion essentially captures all the information that can leak by an incremental obfuscation. Interestingly, we are able to show that increment-private IIO and IBPO are equivalent! That is, hiding the sequence suffices for best possible obfuscation. This is further evidence that increment-private IIO is the right notion to target.

In all of our constructions we work with the *non-adaptive* model of updates — i.e., the updates are not adversarially chosen based on the previously provided obfuscations. In most settings for software updates, the updates are not adversarial since they come from the the software provider. The non-adaptive definition suffices for such applications. However, the adaptive definition might be desirable in other settings, for example if the adversary can influence the choice of updates, e.g., through insider attacks. We do not consider this notion in this work. Additionally, we restrict to the *sequential* model of updates — i.e., changes to the program are made sequentially. Our results, however, easily extend to non-sequential setting which will be discussed towards the end.

Other applications. Besides being interesting in its own right, incremental obfuscation can be seen as a natural tool to enhance applications of obfuscation with incrementality properties. For example, one can very easily enhance known functional encryption schemes [GGH$^+$13] in a way such that secret keys have incrementality properties. In functional encryption an encrypter can encrypt a message m in such a way that a a secret key sk_C, parameterized by the circuit C, can be used to learn $C(m)$. Using our incremental obfuscation we can realize incremental functional encryption, where given a secret key for sk_C one can quickly provide new secret keys for incremental changes in C.

1.2 An Overview of Our Approach

In this section we provide an overview of our constructions. We begin with a systematic exploration, and show how to achieve the basic IIO definition first. We then show how to compile this construction with ORAM to obtain increment-privacy without sacrificing incrementality. For concreteness, we view the program P as a boolean circuit C.

A common approach for obfuscating a general circuit C relies on the Naor-Yung "two-key paradigm" [NY90]: C is encrypted twice to get ciphertexts (e_1, e_2) which are then "hardwired" into a low-depth circuit for which candidate obfuscation is known.

Suppose that we are given IO for the class of all circuits. We will follow the two-key paradigm to obtain IIO. Specifically, we will obfuscate a special circuit C^* using IO which will internally evaluate C. However, we cannot hardwire either C or the value (e_1, e_2) in C^* since even if the values (e_1, e_2) support incrementality, the IO may not.

One option is to provide $e = (e_1, e_2)$ as an explicit input to C^*. C^* can obtain C by decrypting, say e_1, and evaluate it as required. If e_1, e_2 are bitwise encryptions of C, the scheme is also incremental: changing any bit only requires changing the corresponding ciphertexts in e.

In order to make sure that the scheme is secure, C^* must only accept "authorized" values of e. This can be accomplished, for example, by signing e *entirely* after every update. C^* will verify the signature on the updated e before performing the computation. This is indeed a promising approach, however, observe that now the signature must also be incremental since it would be a part of the obfuscation. Unfortunately, this is a big problem since incremental signatures are usually not injective. Consequently, there can exist multiples messages corresponding to the same signature; the existence of such messages is not quite compatible with IO techniques.

Nevertheless, it is a promising approach and it is possible to make it work based on the existence of *(public-coin) differing-inputs obfuscation* (pc-dIO) [ABG$^+$13, BCP14, IPS15]. Our goal is to obtain a feasibility result based on the existence of IO alone, since pc-dIO, due to its extractability flavor, is viewed as a very strong assumption [GGHW14, BP15b].

SSB hash and Merkle trees. A powerful technique to avoid the pc-dIO in many settings is the *somewhere statistically binding* (SSB) hash technique based on Merkle trees. It was introduced by Hubaceck and Wichs [HW15], and is very similar in spirit to the *positional accumulators* of Koppula, Lewko, and Waters [KLW15].

At a high level, the approach considers hash functions which can be generated to be statistically binding at a chosen location. These functions are collisions-resistant, and functions generated for different locations are computationally indistinguishable. Such functions can be constructed from a variety of assumptions, including IO [HW15, KLW15, OPWW15].

Coming back to our construction, we will use the SSB hash functions based on Merkle trees presented in [HW15]. This hash has incrementality property, and consists of two values (h, T) where T is the Merkle tree and h is the root of T. Unfortunately, the SSB hash technique cannot be applied in a black-box manner to all settings. It has to be properly adapted to every setting.

We show how to apply this technique in our setting. More specifically, we take the following high level steps:

1. We encrypt the circuit twice, bit-by-bit, as before to obtain $e = (e_1, e_2)$. We then apply a quickly updatable SSB Hash to e to obtain (h, T). The value h is then "signed."
2. In fact, ordinary signatures are too rigid for our needs. Instead, we use non-interactive zero-knowledge (NIZK) proofs to give proofs that h does not have some special structure.
3. The full obfuscation will include IO of a circuit C^* which has a secret-key sk_1 hardwired for decrypting e_1 and evaluating the resulting circuit on inputs of interest if all NIZK proofs verify.
4. As new updates arrive, values e, h, T are easily updated, as well as the proofs since they only depend on h and not e, T.
5. We then rely on the combined power of SSB-hash functions and NIZK proofs to design a sequence of hybrids such that the signatures exist only for the specified sequence in question. We further use NIZK proofs, SSB hash, and the power of IO to slowly reach a point in hybrid experiments where there is exactly one value e^* that will be accepted by the obfuscated program. At this point, we will able to encrypt the circuits from the other sequence—again, this step is performed one-by-one for each location in the sequence.

We note that executing this strategy is rather involved, and we heavily rely on manipulations via NIZK proofs to complete the security proof.

Increment-privacy via ORAM. To amplify security to increment-private IIO, we rely on two techniques: ORAM programs [Gol87, Ost90, GO96] and the two-key paradigm [NY90]. In more detail, to obfuscate a circuit C, we first apply a statistically-secure ORAM scheme to C twice, to obtain two independent encodings of C, say C_1^*, C_2^*. Next we generate a circuit P that has (C_1^*, C_2^*), and secret information to decode one of them hardcoded in it. The program P, on input x, decodes one of the encoded circuits and outputs the evaluation of the decoded circuit on input x. Having defined P, we now obfuscate P using any IIO scheme (which only satisfies the weaker definition). The resulting obfuscation is our *increment-private* IIO.

At a high level, this works because incremental changes in C can be reduced to corresponding changes in the ORAM encodings of C, and hence the program P. However, since P is encoded using our IIO scheme, this preserves incrementality up to logarithmic factors. At the same time, the use of ORAM ensures the privacy of incremental updates.

We remark that, proving security of our construction raises some additional challenges. In particular, to be able to successfully rely on the security of IIO, the obfuscated program needs a little more "guidance" to perform its task so that the hybrids will go through.

1.3 Related Work

Incremental Cryptography. The concept of incremental cryptography was put forward by Bellare, Goldreich, and Goldwasser [BGG94], as a different type of efficiency measure for cryptographic schemes. They considered the case of hashing and signing, and presented discrete-logarithm based constructions for incremental collision-resistant hash functions and signatures, that support block replacement operation. Soon after, Bellare, Goldreich and Goldwasser [BGG95] also developed constructions for block insertion and deletion, and further issues such as tamper-proof updates, privacy of updates, and incrementality in symmetric encryption were also considered.

Subsequently, Fischlin [Fis97a] presented an incremental signature schemes supporting insertion/deletion of blocks, and tamper-proof updates, and proved a $\Omega(\sqrt{n})$ lower bound in [Fis97b] on the signature size of schemes that support substitution and replacement operations (the bound can be improved to $\Omega(n)$ in certain special cases). The case of hashing was revisited by Bellare and Micciancio [BM97] who provided new constructions for the same based on discrete logarithms and lattices. Buonanno, Katz, and Yung [BKY01] considered the issue of incrementality in symmetric unforgeable encryption and suggested three modes of operations for AES achieving this notion.

Mironov, Pandey, Reingold, and Segev [MPRS12] study incrementality in the context of deterministic public-key encryption. They prove a similar lower bound on the incrementality of such schemes, and present constructions with optimal incrementality.

The task of constructing cryptographic primitives in the complexity class NC^0 can be viewed as a dual of incremental cryptography where the focus is on *output locality* instead of input locality. Applebaum, Ishai, and Kushilevitz [AIK06] resolved this question in the affirmative for public-key encryption, and argue impossibility of the same for constant *input* locality [AIK06, Section C.1].

Obfuscation. Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang [BGI+12] formulated the notion of program obfuscation and proved strong negative results for VBB obfuscation. The connection between zero-knowledge and code obfuscation was first observed and studied by Hada [Had00]. VBB obfuscation for special classes of functions such as point functions were first considered by Wee [Wee05] (and Canetti [Can97]); subsequently, constructions for more functions were achieved such as proxy re-encryption, encrypted signatures, hyperplanes, conjunctions, and so on [LPS04, HRSV07, Had10, CRV10, BR13]. Goldwasser and Kalai extended the negative results for VBB obfuscation in the presence of auxiliary inputs [GK05] which were further extended by Bitansky, Canetti, Cohn, Goldwassser, Kalai, Paneth and Rosen [BCC$^+$14].

The notion of *best possible obfuscation* was put forward by Goldwasser and Rothblum [GR07], who also prove its equivalence to indistinguishability obfuscation (for efficient obfusactors). Bitansky and Canetti [BC10] formulated the notion of *virtual grey box* (VGB) obfuscation which is a simulation based notion; it was further explored in [BCKP14].

The first positive result for indistinguishability obfuscation was first achieved in the breakthrough work of [GGH$^+$13], and further improved in [BGK$^+$14]. In the idealized "generic encodings" model VBB-obfuscation for all circuits were presented in [CV13, BR14, BGK$^+$14, AB15]. These results often involve a "bootstrapping step" which was improved by Applebaum [App14]. Further complexity-theoretic results appear in [BBC$^+$14] and [KMN$^+$14]. Obfuscation in alternative models such as the hardware token model were considered in [GIS$^+$10, BCG$^+$11].

Sahai and Waters [SW14] developed powerful techniques for using indistinguishability obfuscation to construct several (old and new) cryptographic primitives. Since then, IO has been successfully applied to achieve several new results, e.g., [HSW14, BZ14, MO14, PPS15, CLP15, HW15, BP15a, BPW16, AS16]. The equivalence of IO and functional encryption [BSW11, O'N10] was recently established by Ananth and Jain [AJ15] and Bitanksy and Vaikuntanathan [BV15].

Differing input obfuscation (diO) was studied by Ananth, Boneh, Garg, Sahai and Zhandry [ABG$^+$13] and Boyle, Chung and Pass [BCP14], and subsequently used to construct obfuscation for Turing machines. After various implausibility results [GGHW14, BP15b], Ishai, Pandey, and Sahai put forward the improved notion of public-coin differing-inputs obfuscation and recovered several original applications of diO. Indistinguishability obfuscation for bounded-input Turing machines and RAM programs were presented in [BGL$^+$15, CHJV15] and for bounded-space programs in [KLW15].

Oblivious RAM. The notion of oblivious RAM programs was put forward by Goldreich and Ostrovsky [Gol87, Ost90, GO96]. Several improved constructions and variations of ORAM programs are now known [SCSL11, SvDS$^+$13, LO13, CLP14, BCP16].

2 Definitions and Preliminaries

In this section we recall the definitions of some standard cryptographic schemes which will be used in our constructions. For concreteness, we adopt the family of polynomial-sized circuits are model of computation for describing programs. Our definitions, constructions, and results apply to Turing machines and RAM programs as well. We will bring up these models when appropriate.

From here on, unless specified otherwise, λ always denotes the security parameter. We assume familiarity with standard cryptographic primitives, specifically *public-key encryption* (PKE), *non-interactive zero-knowledge proofs* (NIZK, recalled in Section 2.4), *perfectly binding commitments*, and *computational indistinguishability.*

2.1 Indistinguishability Obfuscators

Definition 2.1 (Indistinguishability Obfuscator (IO))**.** *A uniform PPT machine $\mathcal{O}$ is called an* indistinguishability obfuscator *for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:*

- Correctness: *For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that*

$$\Pr[\widetilde{C}(x) = C(x) : \widetilde{C} \leftarrow \mathcal{O}(\lambda, C)] = 1.$$

- Indistinguishability: *For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then*

$$\left| \Pr\left[D(\mathcal{O}(\lambda, C_0)) = 1\right] - \Pr\left[D(\mathcal{O}(\lambda, C_1)) = 1\right] \right| \leq \alpha(\lambda).$$

2.2 Somewhere Statistically Binding Hash

We recall the definition of somewhere statistically binding (SSB) hash from [HW15, OPWW15].

Definition 2.2 (SSB Hash). *A somewhere statistically binding (SSB) hash consists of PPT algorithms* (Gen, H) *and a polynomial* $\ell(\cdot,\cdot)$ *denoting the output length.*

- $hk \leftarrow \mathsf{Gen}(1^\lambda, 1^s, L, i)$*: Takes as input a security parameter λ, a block-length s, an input-length $L \leq 2^\lambda$ and an index $i \in [L]$ (in binary) and outputs a public hashing key hk. We let $\Sigma = \{0,1\}^s$ denote the block alphabet. The output size is $\ell = \ell(\lambda, s)$ and is independent of the input-length L.*
- $H_{hk} : \Sigma^L \to \{0,1\}^\ell$*: A deterministic poly-time algorithm that takes as input $x = (x[0], \ldots, x[L-1]) \in \Sigma^L$ and outputs $H_{hk}(x) \in \{0,1\}^\ell$.*

We require the following properties:

Index Hiding: We consider the following game between an attacker $\mathcal{A}$ and a challenger:

- The attacker $\mathcal{A}(1^\lambda)$ chooses parameters $1^s, L$ and two indices $i_0, i_1 \in [L]$.
- The challenger chooses a bit $b \leftarrow \{0,1\}$ and sets $hk \leftarrow \mathsf{Gen}(1^\lambda, 1^s, L, i)$.
- The attacker $\mathcal{A}$ gets hk and outputs a bit b'. We require that for any PPT attacker $\mathcal{A}$ we have that $|\Pr[b = b'] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$ in the above game.

Somewhere Statistically Binding: We say that hk is *statistically binding for an* index $i \in [L]$ if there do not exist any values $x, x' \in \Sigma^L$ where $x[i] \neq x'[i]$ such that $H_{hk}(x) = H_{hk}(x')$. We require that for any parameters s, L and any integer $i \in [L]$ we have:

$$\Pr[hk \text{ is statistically binding for index } i \ : \ hk \leftarrow \mathsf{Gen}(1^\lambda, 1^s, L, i)] \geq 1 - \mathsf{negl}(\lambda).$$

We say the hash is perfectly binding if the above probability is 1.

Merkle SSB Hash. For concreteness, we work with a specific instantiation of SSB Hash based on Merkle trees (and fully homomorphic encryption) given in [HW15]. The has values from this construction have the form (h, T) where T is the Merkle tree and h is the root of T. This construction has the *incrementality* or "quick update" property: small changes to the underlying value only require $\mathsf{poly}(\lambda, \log n)$ changes to (h, T) where n is the length of the string. Constructions with same properties can be based on a variety of assumptions including IO alone [OPWW15, KLW15].

2.3 Oblivious RAM

We review the notion of ORAM programs from [Gol87, Ost90, GO96]. ORAM can be thought of as a compiler that encodes the memory into a special format such that the sequence of read and write operations into this memory do not reveal the actual access pattern. We recall basic definitions here and refer the reader to [GO96] for more details.

Syntax. A *Oblivious RAM* scheme consists of two procedures $(\mathsf{OData}, \mathsf{OAccess})$ with syntax:

- $(D^*, s) \leftarrow \mathsf{OData}(1^\lambda, D)$: Given a security parameter λ and memory $D \in \{0,1\}^m$ as input, OData outputs the encoded memory D^* and the encoding key s.
- $d \leftarrow \mathsf{OAccess}^{D^*}(1^\lambda, s, \ell, v)$: $\mathsf{OAccess}$ takes as input the security parameter λ and the encoding key s. Additionally, it takes as input a location $\ell \in [m]$ and a value $v \in \{\perp, 0, 1\}$. If $v = \perp$ then this procedure outputs d, the value stored at location ℓ in D. If $v \in \{0,1\}$ the the procedure writes the value v to location ℓ in D. $\mathsf{OAccess}$ has oracle access (read and write) to D^* and changes made to it are preserved from one execution of $\mathsf{OAccess}$ to another.

Efficiency. We require that the run-time of OData should be $m \cdot \mathsf{polylog}(m) \cdot \mathsf{poly}(\lambda)$, and the run-time of $\mathsf{OAccess}$ should be $\mathsf{poly}(\lambda) \cdot \mathsf{polylog}(m)$.

Correctness. Let $\ell_1, \ldots, \ell_t$ be locations accessed on memory D of size m and let $v_1, \ldots, v_t \in \{0, 1, \perp\}$. Then we require that on sequential executions of $d_i = \mathsf{OAccess}^{D^*}(1^\lambda, s, \ell_i, v_i)$ we have that for each $i \in \{1, \ldots t\}$ such that $v_i \neq \perp$ output d_i the correct and the latest value stored in location ℓ_i of D.

Security. For security, we require that for any sequence of access locations $\ell_1, \ldots, \ell_t$ and $\ell'_1, \ldots, \ell'_t$, in $D \in \{0,1\}^m$, we have that:

$$\mathsf{MemAccess} \approx \mathsf{MemAccess}'$$

where $\mathsf{MemAccess}$ and $\mathsf{MemAccess}'$ correspond to the access pattern on D^* during the sequential execution of the $\mathsf{OAccess}^{D^*}$ on input locations $\ell_1, \ldots, \ell_t$ and $\ell'_1, \ldots, \ell'_t$ respectively.

2.4 Non-Interactive Zero-Knowledge Proofs

We recall the definitions of non-interactive zero-knowledge proofs, taken verbatim from [GOS06].

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. Let L be the language consisting of statements in R.

A non-interactive proof system [BFM88, FLS99, GOS06] for a relation R consists of a common reference string generation algorithm K, a prover P and a verifier V. We require that they all be probabilistic polynomial time algorithms, *i.e.*, we are looking at *efficient prover* proofs. The common reference string generation algorithm produces a common reference string σ of length $\Omega(\lambda)$. The prover takes as input (σ, x, w) and produces a proof π. The verifier takes as input (σ, x, π) and outputs 1 if the proof is acceptable and 0 otherwise. We call (K, P, V) a non-interactive proof system for R if it has the completeness and statistical-soundness properties described below.

PERFECT COMPLETENESS. A proof system is complete if an honest prover with a valid witness can convince an honest verifier. Formally we require that for all $(x, w) \in R$, for all $\sigma \leftarrow K(1^\lambda)$ and $\pi \leftarrow P(\sigma, x, w)$ we have that $V(\sigma, x, \pi) = 1$.

Statistical soundness. A proof system is sound if it is infeasible to convince an honest verifier when the statement is false. For all (even unbounded) adversaries $\mathcal{A}$ we have

$$\Pr\left[\sigma \leftarrow K(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma, x, \pi) = 1 : x \notin L\right] = \mathsf{negl}(\lambda).$$

Computational zero-knowledge [FLS99]. A proof system is computational zero-knowledge if the proofs do not reveal any information about the witnesses to a bounded adversary. We say a non-interactive proof (K, P, V) is computational zero-knowledge if there exists a polynomial time simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, such that for all non-uniform polynomial time adversaries $\mathcal{A}$ with oracle access to the prover or the simulator for queries $(x, w) \in R$. Simulator is not provided with the witness w.

$$\Pr\left[\sigma \leftarrow K(1^\lambda) : \mathcal{A}^{P(\sigma,\cdot,\cdot)}(\sigma) = 1\right] \approx \Pr\left[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^\lambda) : \mathcal{A}^{\mathcal{S}_2(\tau,\cdot)}(\sigma) = 1\right].$$

3 Modeling Incremental Obfuscation

In this section, we provide the definitions for incremental program obfuscation. As noted before, for concreteness, we describe our definitions for circuits. The definitions for Turing machines (TM) and RAM programs are obtained by simply replacing circuits with TM/RAMs. We start by providing indistinguishability-based definitions. Let us first set up some notation.

There are four operations we can perform on a bit b: $\mathsf{set}(b) = 1, \mathsf{reset}(b) = 0, \mathsf{flip}(b) = 1 - b$ and $\mathsf{id}(b) = b$; denote by $\mathsf{OP} = \{\mathsf{set}, \mathsf{reset}, \mathsf{flip}, \mathsf{id}\}$ the set of these operations. An *incremental change* to a string x of length n consists of specifying an (ordered) subset $S \subseteq [n]$ of indices of x along with an (ordered) set of corresponding changes $F_S = \{f_i\}_{i \in S}$ (where $f_i \in \mathsf{OP}$, $\forall i \in S$). When we want to be explicit about F_S, we denote the incremental change by (S, F_S).

For a string x, $F_S(x)$ denotes the string x' such that $x'[i] = f_i(x[i])$ for every $i \in S$ and $x'[i] = x[i]$ otherwise. A sequence I of many updates is an ordered sequence of updates $I = ((S_1, F_{S_1}), \ldots, (S_t, F_{S_t}))$ When dealing with a sequence, we write $F_I(x)$ to denote the sequence of *strings* $(x^1, \ldots, x^t)$ where x^j is defined recursively as $x^j := F_{S_j}(x^{j-1})$ for all $j \in [t]$ and $x^0 = x$.

If C is a circuit, represented as a binary string, I is a sequence of t updates to C, and x is an input string, we write $F_I(C)(x)$ to denote the sequence $(C^1(x), \ldots, C^t(x))$ where $(C^1, \ldots, C^t) \stackrel{\text{def}}{=} F_I(C)$

3.1 Incremental Indistinguishability Obfuscation

As discussed earlier, our first definition, called IIO, simply requires that for every sequence of updates I, if I produces two functionally equivalent circuit sequences, then the updated sequence of obfuscated circuits corresponding to I should look indistinguishable. This is the weaker definition.

Definition 3.1 (Incremental Indistinguishability Obfuscator (IIO)). *A pair of uniform PPT machines $(\mathcal{O}, \mathsf{Update})$ is called an* incremental indistinguishability obfuscator *for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:*

- Syntax: *$\mathcal{O}$ takes as input a security parameter λ and a circuit $C \in \mathcal{C}_\lambda$; it outputs an obfuscated circuit $\widetilde{C}$ and a* secret key *sk (for making updates).* Update *takes as input the secret-key sk, an incremental change (S, F_S), and oracle access to $\widetilde{C}$; it outputs an incremental change $(\widetilde{S}, F_{\widetilde{S}})$ for the circuit $\widetilde{C}$.*

- Correctness: *For all security parameters $\lambda \in \mathbb{N}$, for all $C^0 \in \mathcal{C}_\lambda$, for all $t \in \mathbb{N}$, for all sequences of incremental changes $I = (S_1, \ldots, S_t)$ defining the circuit sequence $F_I(C) = (C^1, \ldots, C^t)$, and for all inputs x, we have that*

$$\Pr\left[\bigwedge_{j=0}^{t}\left(\widetilde{C}^j(x)=C^j(x)\right):\begin{array}{l}(\widetilde{C}^0,sk)\leftarrow\mathcal{O}(\lambda,C^0)\\(\widetilde{C}^1,\ldots,\widetilde{C}^t)\leftarrow\mathsf{Update}^{\widetilde{C}^0}(sk,I)\end{array}\right]=1.$$

where $\mathsf{Update}^{\widetilde{C}^0}(sk,I)$ *denotes the (recursively computed) sequence* $(\widetilde{C}^1,\ldots,\widetilde{C}^t)$ *as follows: for every* $j\in[t]$ *define* $(\widetilde{S}_j,F_{\widetilde{S}_j})\leftarrow\mathsf{Update}^{\widetilde{C}^{j-1}}(sk,S_j)$ *and then* $\widetilde{C}^j=F_{\widetilde{S}_j}\left(\widetilde{C}^{j-1}\right)$.

- $\mathsf{Incrementality}$: *There is a fixed polynomial* $\mathsf{poly}(\cdot,\cdot)$, independent of the class $\mathcal{C}_\lambda$*, such that the running time of* $\mathsf{Update}^{\widetilde{C}}\left(sk,\left(\{i\},F_{\{i\}}\right)\right)$ *over all possible values of* $\left(\lambda,\widetilde{C},\left(\{i\},F_{\{i\}}\right),sk\right)$ *is at most* $\Delta(\lambda)=\mathsf{poly}(\lambda,\lg|\widetilde{C}|)$*. The* incrementality *of the scheme is defined to be* $\Delta(\lambda)$*.*

- $\mathsf{Indistinguishability}$: *For any (not necessarily uniform) PPT distinguisher* D*, there exists a negligible function* α *such that the following holds: For all security parameters* $\lambda\in\mathbb{N}$*, for all pairs of circuits* $C_0,C_1\in\mathcal{C}_\lambda$*, for every polynomial* t *and for every sequence* I *of* t *updates we have that if* $C_0(x)=C_1(x)$ *and* $F_I(C_0)(x)=F_I(C_1)(x)$ *for all inputs* x*, then*

$$\left|\Pr\left[D(\mathsf{Expt}(\lambda,C_0,C_1,I,0))=1\right]-\Pr\left[D(\mathsf{Expt}(\lambda,C_0,C_1,I,1))=1\right]\right|\le\alpha(\lambda).$$

where distribution $\mathsf{Expt}(\lambda,C_0,C_1,I,b)$ *outputs as follows: sample* $(\widetilde{C}_b,sk)\leftarrow\mathcal{O}(\lambda,C_b)$*, sample sequence* $(\widetilde{C}_b^1,\ldots,\widetilde{C}_b^t)\leftarrow\mathsf{Update}^{\widetilde{C}_b}(sk,I)$*, and output* $(I,C_0,C_1,\widetilde{C}_b,\widetilde{C}_b^1,\ldots,\widetilde{C}_b^t)$*.*

The above definition does not necessarily hide the sequence I from the adversary. Our next definition, called *increment-private* IIO hides the sequence of updates as well. Informally, it states that if update sequences I_0,I_1 produce functionally equivalent circuit sequences, then the sequence of updated obfuscations hides whether I_0 or I_1 was used for making updates.

Definition 3.2 (Increment-private IIO)**.** *A pair of uniform PPT machines* $(\mathcal{O},\mathsf{Update})$ *is called a* sequence hiding *incremental indistinguishability obfuscator for a circuit class* $\{\mathcal{C}_\lambda\}$ *if it satisfies the* syntax, correctness *and* incrementality *properties (as in definition 3.1) and the following* sequence-hiding indistinguishability *property:*

- Increment-private indistinguishability: *For any (not necessarily uniform) PPT distinguisher* D*, there exists a negligible function* α *such that the following holds: For all security parameters* $\lambda\in\mathbb{N}$*, for all pairs of circuits* $C_0,C_1\in\mathcal{C}_\lambda$*, for every polynomial* t *and for all pairs of update sequences* I_0,I_1 *of length* t *we have that if* $C_0(x)=C_1(x)$ *and* $F_{I_0}(C_0)(x)=F_{I_1}(C_1)(x)$ *for all inputs* x*, then*

$$\left|\Pr\left[D(\mathsf{Expt}(\lambda,C_0,C_1,I_0,I_1,0))=1\right]-\Pr\left[D(\mathsf{Expt}(\lambda,C_0,C_1,I_0,I_1,1))=1\right]\right|\le\alpha(\lambda).$$

where distribution $\mathsf{Expt}(\lambda,C_0,C_1,I_0,I_1,b)$ *outputs as follows: sample* $(\widetilde{C}_b,sk)\leftarrow\mathcal{O}(\lambda,C_b)$*, sample sequence* $(\widetilde{C}_b^1,\ldots,\widetilde{C}_b^t)\leftarrow\mathsf{Update}^{\widetilde{C}_b}(sk,I_b)$*, and output* $(I_0,I_1,C_0,C_1,\widetilde{C}_b,\widetilde{C}_b^1,\ldots,\widetilde{C}_b^t)$*.*

Finally, we define the following *pristine updates* property which is desirable but not implied by above definitions: updated obfuscation of a circuit should be statistically close to its fresh obfuscation.

Definition 3.3 (Pristine updates)**.** *An incremental indistinguishability obfuscator* $(\mathcal{O},\mathsf{Update})$ *for a circuit class* $\{\mathcal{C}_\lambda\}$ *has* pristine updates *if there exists a negligible function* α *such that* $\forall\lambda\in\mathbb{N}$*,* $\forall C\in\mathcal{C}_\lambda$*, and* $\forall(S,F_S)$*, the statistical distance between distributions* $\{\widetilde{C}:(\widetilde{C},sk)\leftarrow\mathcal{O}(\lambda,C)\}$ *and* $\{\widetilde{C}':(\widetilde{C},sk)\leftarrow\mathcal{O}(\lambda,C),(\widetilde{S},F_{\widetilde{S}})\leftarrow\mathsf{Update}^{\widetilde{C}}(sk,(S,F_S)),\widetilde{C}'=F_{\widetilde{S}}(\widetilde{C})$ *is at most* $\alpha(\lambda)$*.*

3.2 Incremental VGB and VBB Obfuscation

The simulation based definitions are defined analogously, but require the existence of a simulator.

Definition 3.4 (Incremental $\mathsf{VGB}/\mathsf{VBB}$ Obfuscator)**.** *A pair of uniform PPT machines* $(\mathcal{O}, \mathsf{Update})$ *is called an* incremental VGB obfuscator *for a circuit class* $\{\mathcal{C}_\lambda\}$ *if it satisfies the* syntax, correctness *and* incrementality *properties (as in definition 3.1) and the following VGB security property:*

For any (not necessarily uniform) PPT (single bit) adversary A, there exists a simulator $\mathcal{S}$, a polynomial q, and a negligible function α, such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all circuits $C \in \mathcal{C}_\lambda$, for every polynomial t and for every sequence I of t updates, we have that,

$$\left| \Pr\left[A(\widetilde{C}, \widetilde{C}^1, \ldots, \widetilde{C}^t) = 1\right] - \Pr\left[\mathcal{S}^{C[q(\lambda)], F_I(C)[q(\lambda)]}(1^\lambda, 1^{|C|}) = 1\right] \right| \leq \alpha(\lambda).$$

where $(\widetilde{C}, sk) \leftarrow \mathcal{O}(\lambda, C)$, $(\widetilde{C}^1, \ldots, \widetilde{C}^t) \leftarrow \mathsf{Update}^{\widetilde{C}}(sk, I)$, and notation$C[q(\lambda)]$ (resp., $F_I(C)[q(\lambda)]$) represents at most q oracle calls to C (resp., every circuit in $(C, F_I(C))$).

If $\mathcal{S}$ is polynomial time in λ, we say that $(\mathcal{O}, \mathsf{Update})$ is incremental VBB *obfuscator for $\{\mathcal{C}_\lambda\}$.*

4 Our Construction

In this section, we present our basic construction which satisfies IIO notion. Let:

- (G, E, D) be a PKE scheme for bits with ciphertext length ℓ_e,
- (Gen, H) be an SSB hash with alphabet $\Sigma = \{0,1\}^{\ell_e}$ and output length ℓ,
- $\mathcal{O}$ be an IO scheme for all circuits,
- (K, P, V) be a NIZK proof system for NP
- com be a non-interactive perfectly binding string commitment scheme

The component algorithms of our IIO scheme $(\mathsf{Inc}\mathcal{O}, \mathsf{Update})$ are described in figures (1, 3).

Theorem 4.1. *Scheme* $(\mathsf{Inc}\mathcal{O}, \mathsf{Update})$ *is an* IIO *scheme for all circuits.*

Proof: It is easy to verify the correctness and incrementality of this scheme. It is also easy to see that the scheme satisfies the pristine updates property.

We show that it satisfies the indistinguishability property. Fix any two circuits (C_0, C_1), and any sequence I of any polynomial many incremental updates, say t updates. We need to show that $\mathsf{Expt}(\lambda, C_0, C_1, I, 0) \equiv_c \mathsf{Expt}(\lambda, C_0, C_1, I, 0)$.

$\mathcal{H}_1$: Same as $\mathsf{Expt}(\lambda, C_0, C_1, I, 0)$. For convenience, let us define $C_0^0 = C_0$ and $C_1^0 = C_1$. Recall that the output of this experiment is $(I, C_0^0, C_1^0, \widetilde{C}_0^0, \widetilde{C}_0^1, \ldots, \widetilde{C}_0^t)$ where $\widetilde{C}_0^i$ is of the form $(e^i, h^i, T^i, r^i, \pi_{h^i}, \widetilde{P}_{hk,\sigma,c_1,c_2,c_3,sk_1})$ and $e^i = (e_1^i, e_2^i)$ represents two (bitwise) encryptions of C_0^i under keys pk_1 and pk_2 respectively.

$\mathcal{H}_2$: This hybrid tests $|r+t| \leq \ell$ (i.e., $r+t \leq 2^\ell - 1$), and if so, it sets $\underline{c_3 \text{ to be a commitment to } t_1 \| t_2}$ for $t_1 = r$ and $t_2 = r + t$, instead of $0^\ell \| 1^\ell$, and continues as $\mathcal{H}_2$. If the test fails, it aborts.

The advantage in distinguishing $\mathcal{H}_2$ and $\mathcal{H}_3$ is at most $\delta_{\mathsf{com}} + t2^{-\ell}$ where δ_{com} is the distinguishing advantage for com. This is because of the following: value $r + t > 2^\ell$ for a randomly

Algorithm $\mathsf{Inc}\mathcal{O}(1^\lambda, C)$ proceeds as follows:

1. Sample $(pk_1, sk_1) \leftarrow K(1^\lambda)$ and $(pk_2, sk_2) \leftarrow G(1^\lambda)$.
2. Generate $e = (e_1, e_2)$ where e_1, e_2 are *bit-wise* encryptions of C, i.e., $e_1 = (e_{1,1}, \dots, e_{1,|C|})$ and $e_2 = (e_{2,1}, \dots, e_{2,|C|})$ where $e_{1,i} \leftarrow E(pk_1, C[i])$, $e_{2,i} \leftarrow E(pk_2, C[i])$, for every $i \in [|C|]$.
3. Sample $hk \leftarrow \mathsf{Gen}(1^\lambda, 1^{\ell_e}, |e|, 0)$ and $\sigma \leftarrow K(1^\lambda)$
4. Generate $c_1 \leftarrow \mathsf{com}(1^\ell; \omega_1), c_2 \leftarrow \mathsf{com}(1^\ell; \omega_2), c_3 \leftarrow \mathsf{com}(0^\ell \| 1^\ell; \omega_3)$
5. Compute $(h, T) = H_{hk}(e)$, where h is the root node of the (Merkle) tree T.
6. Choose a random r and compute the proof $\pi_h \leftarrow P(\sigma, (h, r, c_1, c_2, c_3), w)$ for relation R using witness $w = (1^\ell, 1^\ell, 0^\ell \| 1^\ell, \omega_1, \omega_2, \omega_3)$ where $((h, r, c_1, c_2, c_3), (h', u, t_1 \| t_2, \omega_1, \omega_2, \omega_3)) \in R$ iff:
   (a) $c_1 = \mathsf{com}(h'; \omega_1), c_2 = \mathsf{com}(u; \omega_2), c_3 = \mathsf{com}(t_1 \| t_2; \omega_3), t_1 \le r \le t_2$, AND
   (b) EITHER $\big(h \neq h' \land r \neq u\big)$ OR $\big(h = h' \land r = u\big)$
7. Compute obfuscation $\widetilde{P}_{hk,\sigma,c_1,c_2,c_3,sk_1} \leftarrow \mathcal{O}\left(1^\lambda, P_{hk,\sigma,c_1,c_2,c_3,sk_1}\right)$ where $P_{hk,\sigma,c_1,c_2,c_3,sk_1}$ is described in figure 2. The size of $P_{hk,\sigma,c_1,c_2,c_3,sk_1}$ is padded to a value Q defined later.
8. Let $\widetilde{C} := \left(e, h, T, r, \pi_h, \widetilde{P}_{hk,\sigma,c_1,c_2,c_3,sk_1}\right)$ and $uk := (pk_1, hk, c_1, c_2, c_3, \sigma, w)$; output $(\widetilde{C}, uk)$.

Evaluation of $\widetilde{C}$ on input x is obtained by evaluating $\widetilde{P}_{hk,\sigma,c_1,c_2,c_3,sk_1}$ on input (e, r, π_h, x).

Figure 1: Description of $\mathsf{Inc}\mathcal{O}$

Circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1}$ computes as follows on input (e, v, π, x):

1. Compute $(h, T) = H_{hk}(e)$ and verify that $V(\sigma, (h, v, c_1, c_2, c_3), \pi) = 1$.
   Output $\perp$ if verification fails.
2. Otherwise, parse e as (e_1, e_2), compute $C = D(sk_1, e_1)$, and output $C(x)$.

Figure 2: Description of $P_{hk,\sigma,c_1,c_2,c_3,sk_1}$

Algorithm $\mathsf{Update}^{C,\widetilde{C}}(uk, S)$ computes as follows:

Parse the update key as $uk = (pk_1, hk, c_1, c_2, c_3, \sigma, w)$ where $w = (1^\ell, 1^\ell, 1^\ell, \omega_1, \omega_2, \omega_3)$.
For every index $i \in S$ do the following:

1. Access the bit $C[i]$, and corresponding ciphertexts $e_{1,i}$ and $e_{2,i}$ from e (which is part of $\widetilde{C}$).
2. Generate updated ciphertexts $e'_{1,i} \leftarrow E(pk_1, 1 - C[i])$ and $e'_{2,i} \leftarrow E(pk_2, 1 - C[i])$.
   Let e' denote the string e when $e_{1,i}$ is replaced with $e'_{1,i}$ and $e_{2,i}$ is replaced with $e'_{2,i}$.
3. Define $(h', T') = H_{hk}(e')$, and let S' be the set of indices where e, e' differ.
   Run the (incremental) update algorithm to get $S'' \leftarrow \mathsf{HashUpdate}^{e,h,T}(S')$.
4. Access h and compute h' using the knowledge of S''. Access v and set $v' = v + 1$.
   Compute the new proof $\pi_{h'} \leftarrow P(\sigma, (h', v', c_1, c_2, c_3), w)$.
5. Output the set of indices where $(e', h', T', v', \pi_{h'})$ differ from (e, h, T, v, π_h).

Figure 3: Description of Update.

chosen r if and only if $r \in [2^\ell - t, 2^\ell]$ which happens with probability $t2^{-\ell}$; if the test succeeds, the hybrids differ only in commitment c_3.

This hybrid ensures that there are no valid proofs other than the $t+1$ proofs that are given as part of the obfuscation and the updates.

$\mathcal{H}_3$: This hybrid is identical to $\mathcal{H}_1$ except that it generates the components $\{e_2^i\}$ by (bitwise) encrypting the sequence of circuits corresponding C_1, namely $\{C_1^i\}$ (instead of $\{C_0^i\}$) for every $i \in [t]$. That is, for ever $i \in [t]$, it computes $\widetilde{C}_0^i$ to be of the form $(e^i, h^i, T^i, \pi_{h^i}, \widetilde{P}_{hk,\sigma,c_1,c_2,c_3,sk_1})$ as before where $e^i = (e_1^i, e_2^i)$ and e_1^i is a bitwise encryption of C_0^i under pk_1 obtained via updates but $\underline{e_2^i \text{ is a bitwise encryption of } C_1^i \text{ under } pk_2}$ (also obtained via updates).

The advantage in distinguishing $\mathcal{H}_1$ and $\mathcal{H}_2$ is at most $t \cdot |C_0| \cdot \delta_{\text{pke}}$ where δ_{pke} denotes the advantage in breaking the security of PKE scheme. This claim is straightforward.

$\mathcal{H}_4$: Identical to $\mathcal{H}_3$ except that instead of obfuscating circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1}$, it obfuscates the following circuit $P2_{hk,\sigma,c_1,c_2,c_3,sk_2}$[2], padded to the size Q:

Circuit $P2_{hk,\sigma,c_1,c_2,c_3,sk_2}(e, v, \pi, x)$**:**

1. Compute $(h, T) = H_{hk}(e)$ and verify that $V(\sigma, (h, v, c_1, c_2, c_3), \pi) = 1$.
   Output $\perp$ if verification fails.
2. Otherwise, parse e as (e_1, e_2), $\underline{\text{compute } C = D(sk_2, e_2)}$, and output $C(x)$.

Let δ_4 denote the distinguishing advantage of any polynomial time distinguisher between $\mathcal{H}_4$ and $\mathcal{H}_3$. From lemma 4.1, $\delta_4 \leq 4t^2|C_0| \cdot (\delta_{\text{ssb}} + \delta_{\text{nizk}} + \delta_{\mathsf{com}} + \delta_{\mathsf{IO}})$ where quantities $\delta_{\text{ssb}}, \delta_{\text{nizk}}$, and δ_{IO} denote the distinguishing advantage for SSB hash, NIZK proofs, and IO respectively.

$\mathcal{H}_5$: Same as $\mathcal{H}_4$ except that it generates ciphertexts $\underline{e_1^i \text{ to now encrypt } C_1^i}$ (instead of C_0^i) $\forall i \in [t]$. That is, for every $i \in [t]$, it computes $\widetilde{C}_0^i$ to be of the form $(e^i, h^i, T^i, \pi_{h^i}, \widetilde{P2}_{hk,\sigma,c_1,c_2,c_3,sk_2})$ where $e^i = (e_1^i, e_2^i)$ and e_1^i, e_2^i are (bitwise) encryptions of C_1^i under pk_1, pk_2 respectively. Note that these encryptions are actually obtained via updates.

The distinguishing advantage between $\mathcal{H}_5$ and $\mathcal{H}_4$ is at most $t|C_0|\delta_{\text{pke}}$. This is straightforward.

$\mathcal{H}_6$: Same as $\mathcal{H}_5$ except that it obfuscates circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1}$ (which uses sk_1 and decrypts from e_1, see figure 2) instead of $P2_{hk,\sigma,c_1,c_2,c_3,sk_2}$.

Let δ_6 denote the distinguishing advantage between $\mathcal{H}_6$ and $\mathcal{H}_5$. We claim that $\delta_6 \leq 4t^2|C_0| \cdot (\delta_{\text{ssb}} + \delta_{\text{nizk}} + \delta_{\mathsf{com}} + \delta_{\mathsf{IO}})$. The proof is identical to that of lemma 4.1 and omitted.

$\mathcal{H}_7$: Same as $\mathcal{H}_6$ but now c_3 is switched back to a commitment of $0^\ell \| 1^\ell$ instead of $t_1 \| t_2$. Recall that $t_1 = r, t_2 = r + t$.

The distinguishing advantage between $\mathcal{H}_7$ and $\mathcal{H}_6$ is at most $\delta_{\mathsf{com}} + t2^{-\ell}$ (as argued in $\mathcal{H}_2$).

Observe that $\mathcal{H}_7$ is the same as experiment $\mathsf{Expt}(\lambda, C_0, C_1, I, 1)$. The total distinguishing advantage is thus bounded by $O\left(t^2 \cdot |C_0| \cdot \left(\delta_{\text{pke}} + \delta_{\text{nizk}} + \delta_{\text{ssb}} + \delta_{\mathsf{com}} + 2^{-\ell}\right)\right)$ which is negligible.

$\square$

Lemma 4.1. $\delta_4 \leq 4t^2|C_0| \cdot (\delta_{\text{ssb}} + \delta_{\text{nizk}} + \delta_{\mathsf{com}} + \delta_{\mathsf{IO}})$.

[2]See step 7, figure 1. Note that this obfuscation is generated only once and never changes during the updates.

Circuit $P_{hk,\sigma,c_1,c_2,c_3,r,sk_1,sk_2,j}(e,v,\pi,x)$**:**

1. Compute $(h,T) = H_{hk}(e)$ and verify that $V(\sigma,(h,v,c_1,c_2,c_3),\pi) = 1$.
   Output $\perp$ if verification fails.
2. Otherwise, parse e as (e_1,e_2) and compute C as follows:
   If $r \leq v \leq r+j$, set $C = D(sk_2,e_2)$;
   If $r+j < v \leq r+t$, set $C = D(sk_1,e_1)$.
3. Output $C(x)$.

Figure 4: Description of $P_{hk,\sigma,c_1,c_2,c_3,sk_1,j}$

Proof: The lemma is proven by focusing on one of the t locations in the sequence at a time, and use the properties of SSB hash, to slowly reach a point where there is a unique value of e corresponding to the hash value at this location. All values prior to this location will use sk_2 and e_2, whereas those after it will use sk_1, e_1.

More precisely, we describe t hybrids $\mathcal{G}_1,\ldots,\mathcal{G}_t$ where hybrid $\mathcal{G}_j$ will use sk_2 on inputs with value v if $r \leq v < r+j$, and sk_1 if $r+j \leq v \leq r+t$. Note that v is always in the range $[r, r+t]$ in this hybrid. To prove that $\mathcal{G}_{j-1}$ and $\mathcal{G}_j$ are indistinguishable, we will design another $4+2|C_0|$ hybrids where we will first ensure the uniqueness of (j,h^j) and then perform SSB hash translation to move to $\mathcal{G}_j$.

Formally, define $\mathcal{G}_0$ to be the same as $\mathcal{H}_3$, and for $j \in [t]$ define hybrid $\mathcal{G}_j$ as follows: it is identical to $\mathcal{G}_{j-1}$ except that it obfuscates the circuit $P_{hk,\sigma,c_1,c_2,c_3,r,sk_1,sk_2,j}$ described in figure 4 (instead of $P_{hk,\sigma,c_1,c_2,c_3,r,sk_1,sk_2,j-1}$).[3]
Note that r was chosen uniformly in step 6 of the construction and the size of $P_{hk,\sigma,c_1,c_2,c_3,r,sk_1,sk_2,j}$ is padded to the value Q before obfuscation.

Let ε_j denote the distinguishing advantage between $\mathcal{G}_j$ and $\mathcal{G}_{j-1}$. Observe that $\mathcal{G}_t$ is the same as $\mathcal{H}_4$ and $\delta_4 \leq \sum_j \varepsilon_j$. We now prove that $\varepsilon_j \leq 4t|C_0| \cdot (\delta_{\text{ssb}} + \delta_{\text{nizk}} + \delta_{\mathsf{com}} + \delta_{\mathsf{IO}})$.

Consider the following hybrids:

$\mathcal{G}_{j:1}$: Same as $\mathcal{G}_{j-1}$ except that the CRS σ and all NIZK proofs are obtained from the simulator, i.e., values $(\sigma, \pi_{h^1},\ldots,\pi_{h^t})$ are sampled using the simulator. The distinguishing advantage from $\mathcal{G}_{j-1}$ is at most $t\delta_{\text{nizk}}$.

$\mathcal{G}_{j:2}$: Same as $\mathcal{G}_{j:1}$ except that commitments c_1 and c_2 are changed as follows. Let e^j denote the encryptions corresponding to the j-th update in the sequence, and let $h^j = H_{hk}(e^j)$. Then, c_1 is set to be a commitment to h^j and c_2 a commitment to $r+j$: $c_1 = \mathsf{com}(h^j;\omega_1), c_2 = \mathsf{com}(r+j;\omega_2)$. The distinguishing advantage from $\mathcal{G}_{j:1}$ is at most δ_{com}.

$\mathcal{G}_{j:3}$: Same as $\mathcal{G}_{j:2}$ except that the CRS σ and all NIZK proofs $(\pi_{h^1},\ldots,\pi_{h^t})$ are now generated normally using the appropriate witness (instead of being simulated). Note that the hybrid indeed does have the appropriate witnesses to complete this step normally, and the distinguishing advantage from $\mathcal{G}_{j:2}$ is at most $t\delta_{\text{nizk}}$.

At this point, for location j, there exists only one value of h^j that can produce convincing proofs (from the soundness of NIZK). Next, we will use the SSB hash to reach a point where only the string e^j will be accepted by the obfuscated circuit as the valid input corresponding to hash h^j. We

[3]Note that the only difference between these two circuits are values j and $j-1$, and this obfuscation is performed only *once* throughout the whole sequence.

do this in a series of $2|C_0|$ hybrids corresponding to the $2|C_0|$ encryptions occurring in $e^j = (e_1^j, e_2^j)$.

Formally, for every $m = 1, \ldots, 2|C_0|$, we define two hybrids $\mathcal{G}_{j:3}^{m:1}, \mathcal{G}_{j:3}^{m:2}$ below. Let e_m^* denote the string which identical to e^j in the first m blocks, each of length ℓ_e, and 0 everywhere else. For convention, let $\mathcal{G}_{j:3}^{0:2}$ be the same as $\mathcal{G}_{j:3}$. Then:

$\mathcal{G}_{j:3}^{m:1}$: Same as $\mathcal{G}_{j:3}^{m-1:2}$ except that it makes hk to be binding at m, i.e., $hk \leftarrow \mathsf{Gen}(1^\lambda, 1^{\ell_e}, |e|, m)$.

$\mathcal{G}_{j:3}^{m:2}$: Let us recall that we started with hybrid $\mathcal{G}_{j-1}$ which obfuscates the circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j-1}$. This hybrid proceeds just like $\mathcal{G}_{j:3}^{m:1}$ except that it obfuscates the circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j-1}^{m,e_m^*}$ described below.

Circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j-1}^{m,e_m^*}(e, v, \pi, x)$**:**

1. If the first m blocks of e and e_m^* are not the same, output $\perp$.
2. Otherwise output $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j-1}(e, v, \pi, x)$ (see figure 4).

The distinguishing advantage between $\mathcal{G}_{j:3}^{m-1:2}$ and $\mathcal{G}_{j:3}^{m:1}$ is at most δ_{ssb}, and between $\mathcal{G}_{j:3}^{m:1}$ and $\mathcal{G}_{j:3}^{m:2}$ is at most δ_{IO}.

When $m = 2|C_0|$, the circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j}^{m,e_m^*}$ accepts only $e_{2|C_0|}^* = e^j$ as the input for location j in the sequence and all other inputs are rejected. We can now safely change this program to use sk_2 to decrypt e_2^j at location j (instead of e_1^j). More precisely, we consider the hybrid:

$\mathcal{G}_{j:4}$ Same as $\mathcal{G}_{j:3}^{2|C_0|:2}$ except that it obfuscates a circuit which, in location j, decrypts from e_2^j. More precisely, it obfuscates the following circuit:

Circuit $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j-1}^{e^j}(e, v, \pi, x)$**:**

1. If $e \neq e^j$, output $\perp$.
2. Otherwise output $P_{hk,\sigma,c_1,c_2,c_3,sk_1,\underline{sk_2,j}}(e, v, \pi, x)$ (see figure 4).

The distinguishing advantage from previous hybrid is at most δ_{IO}.

Our goal is now to get rid of the first condition, so that we switch back to only obfuscating $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j}$. This is performed by simply reversing the steps in m hybrids. Furthermore, we also reverse the changes made in the commitment in a sequence of 3 hybrids by considering the reverse of hybrids $\mathcal{G}_{j:3}, \mathcal{G}_{j:2}, \mathcal{G}_{j:1}$. The resulting hybrid would essentially be identical to $\mathcal{G}_j$. We omit these details.

The total distinguishing advantage between $\mathcal{G}_j$ and $\mathcal{G}_{j-1}$ is bounded by the sum of all advantages, which is at most $4t|C_0|\left(\delta_{\mathsf{com}} + \delta_{\text{nizk}} + \delta_{\text{ssb}} + \delta_{\mathsf{IO}}\right)$. This completes the proof.

Size Q: The value of Q is defined to be the size of the program $P_{hk,\sigma,c_1,c_2,c_3,sk_1,sk_2,j-1}^{e^j}$ described above (for any value of j, say $j = 1$). □

Construction for Turing machines and RAM programs. As mentioned earlier, our constructions are quite independent of the underlying model of computation. For example, to obtain construction for the class of bounded-input Turing machines, we use the same construction as in figure 1 except that the obfuscator $\mathcal{O}$ for circuits in (step 7) will now be an obfuscator for the class of bounded-input Turing machines since the program in figure 2 will now be a bounded-input Turing machine. Likewise, we also obtain constructions for RAM programs and unbounded-input Turing machines assuming the existence of IO for the same. This gives us the following theorem.

Theorem 4.2. *If there exists indistinguishability obfuscation for a class of programs $\mathcal{P} = \{P_\lambda\}$ modeled as either boolean circuits, or (bounded/unbounded input) Turing machines, or RAM programs, then there exists* incremental *indistinguishability obfuscation (*IIO*, definition 3.1) for $\mathcal{P}$.*

5 Amplifying Security to Increment-private IIO

In this section, we present our (black-box) transformation which transform an IIO scheme (definition 3.1) into an increment-private IIO scheme (definition 3.2). As before, for concreteness, we present our transformation for circuits, but it works for Turing machines and RAM programs as well. Our transformation preserves the *pristine updates* property as well (definition 3.3).

As discussed in the overview, the construction consists of applying the ORAM encoding on the given circuit twice. These encodings are then hardwired into a program, along with the secret information for decoding only *one* of the two ORAM encodings. This is essentially the two-key paradigm [NY90] implemented with ORAM. The resulting program is then obfuscated using the given IIO scheme. In addition to the encodings and secret information, the program is also provided with a bit b as well as some more information which tells the program which ORAM encoding to pick for evaluation. This is helpful in designing the hybrid experiments.

Our construction. Let $(\mathcal{O}, \mathsf{Update})$ be an IIO scheme for the class of all circuits. Let $(\mathsf{OData}, \mathsf{OAccess})$ be an oblivious RAM scheme as described in Section 2.3. Our new scheme consists of algorithms $(\mathcal{O}', \mathsf{Update}')$ described in figure 5.[4]

Theorem 5.1. *Scheme $(\mathcal{O}', \mathsf{Update}')$ is* increment-private IIO *for all circuits (definition 3.2).*

Proof: The correctness and the pristine updates property of our construction follows directly from the correctness and the pristine updates property of the underlying IIO scheme. We argue the increment-private indistinguishability property of our construction.

We have to show that for any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, for every polynomial t and for all pairs of update sequences I_0, I_1 of length t we have that if $C_0(x) = C_1(x)$ and $F_{I_0}(C_0)(x) = F_{I_1}(C_1)(x)$ for all inputs x, then

$$\left| \Pr\left[D(\mathsf{Expt}(\lambda, C_0, C_1, I_0, I_1, 0)) = 1\right] - \Pr\left[D(\mathsf{Expt}(\lambda, C_0, C_1, I_0, I_1, 1)) = 1\right] \right| \leq \alpha(\lambda).$$

where distribution $\mathsf{Expt}(\lambda, C_0, C_1, I_0, I_1, b)$ outputs as follows: (1) Sample $(C_1^*, s_1) \leftarrow \mathsf{OData}(1^\lambda, C_b)$ and $(C_2^*, s_2) \leftarrow \mathsf{OData}(1^\lambda, C_b)$. (2) Sample $(\widetilde{C}, sk) \leftarrow \mathcal{O}(\lambda, P_{C_1^*, C_2^*, 1, s_1})$ and the sequence $(\widetilde{C}^1, \ldots, \widetilde{C}^t) \leftarrow \mathsf{Update}^{\widetilde{C}}(sk, J)$, and output $(I_0, I_1, C_0, C_1, \widetilde{C}, \widetilde{C}^1, \ldots, \widetilde{C}^t)$. Here $J = (S_1, \ldots, S_t)$ is obtained from $I_b = (S'_{b,1}, \ldots S'_{b,t})$ as follows (also described in figure 5). For each $j \in \{1, \ldots t\}$, set $S'_b = S'_{b,j}$ and proceed as follows.

1. $F_{S'_b}$ be the changes corresponding to S'_b. For each $i \in S'_b$ proceed as follows:
   (a) Let $d_i := \mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, f_i(d_i))$.
   (b) Similarly, let $d'_i := \mathsf{OAccess}^{C_2^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_2^*}(1^\lambda, s, i, f_i(d'_i))$.

[4]We emphasize that even though our scheme uses ORAM, program $P_{C_1^*, C_2^*, s, b}$ in figure 5 is still only a *circuit* and not a "RAM" program.

Algorithm $\mathcal{O}'(1^\lambda, C)$ proceeds as follows:

1. Sample $(C_1^*, s_1) \leftarrow \mathsf{OData}(1^\lambda, C)$ and $(C_2^*, s_2) \leftarrow \mathsf{OData}(1^\lambda, C)$. Let $m = |C|$.
2. Obtain $(\widetilde{C}, uk) \leftarrow \mathcal{O}(P_{C_1^*, C_2^*, 1, s_1})$ where $P_{C_1^*, C_2^*, b, s}$ for $b \in \{1, 2\}$ is a circuit that on input x proceeds as follows:
   (a) For every $i \in [m]$ compute $d_i := \mathsf{OAccess}^{C_b^*}(1^\lambda, s, i, \perp)$.
   (b) Output $D(x)$ where $D = d_1 \| \ldots \| d_m$.
3. Output $(\widetilde{C}, uk')$ where the update key $uk' := (s_1, s_2, uk)$.

Algorithm $\mathsf{Update}'^{\widetilde{C}}(uk', S')$ computes as follows:

1. Parse the update key as $uk' = (s_1, s_2, uk)$ and let $F_{S'}$ be the changes corresponding to S'. For each $i \in S'$ proceed as follows:
   (a) Let $d_i := \mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, f_i(d_i))$.
   (b) Similarly, let $d_i' := \mathsf{OAccess}^{C_2^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_2^*}(1^\lambda, s, i, f_i(d_i'))$.

   where f_i is the update operation in $F_{S'}$ corresponding to $i \in S'$.
2. Let S be the set of locations where $P_{C_1^*, C_2^*, 1, s_1}$ is touched as C_1^* and C_2^* are processed as above[a] and F_S be the corresponding changes.
3. Output $\mathsf{Update}^{\widetilde{C}}(uk, S)$.

[a]Note that the size of S is polynomial in $|S'|, \lambda, \log m$.

Figure 5: Description of $\mathcal{O}'$ and Update'.

2. Let S be the set of locations where $P_{C_1^*, C_2^*, 1, s_1}$ is touched as C_1^* and C_2^* are processed as above[5] and F_S be the corresponding changes.
3. Output $\mathsf{Update}^{\widetilde{C}}(uk, S)$ as S_j.

To prove the claim, consider the following sequence of hybrids:

$\mathcal{H}_1$: This hybrid corresponds to the output of the experiment $\mathsf{Expt}(\lambda, C_0, C_1, I_0, I_1, 0)$ as above.

$\mathcal{H}_2$: In this hybrid we change Step 1b in the experiment above. In particular we instead of changing stored value to $f_i(d_i)$ we always set it to zero. More formally, the procedure is changed as follows.

(a) $F_{S_b'}$ be the changes corresponding to S_b'. For each $i \in S_0'$ proceed as follows:
   i. Let $d_i := \mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, f_i(d_i))$.
   ii. Similarly, let $d_i' := \mathsf{OAccess}^{C_2^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_2^*}(1^\lambda, s, i, 0)$.

(b) Let S be the set of locations where $P_{C_1^*, C_2^*, 1, s_1}$ is touched as C_1^* and C_2^* are processed as above and F_S be the corresponding changes.

(c) Output $\mathsf{Update}^{\widetilde{C}}(uk, S)$ as S_j.

[5]Note that the size of S is polynomial in $|S'|, \lambda, \log m$.

Indistinguishability follows from the security of the IIO scheme. Here we use the property that any changes made to C_2^* do not affect the functionality of the program $P_{C_1^*,C_2^*,1,s_1}$.

$\mathcal{H}_3$: In this hybrid we again change Step 1b in the experiment above. In particular, we make changes to C_2^* at locations S'_1 instead of S'_0. As in $\mathcal{H}_1$ we still set these locations to zero when the change is made. More formally:

(a) $F_{S'_b}$ be the changes corresponding to S'_b. For each $i \in S'_0$ and $k \in S'_1$ proceed as follows:
   i. Let $d_i := \mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, f_i(d_i))$.
   ii. Similarly, let $d'_k := \mathsf{OAccess}^{C_2^*}(1^\lambda, s, k, \perp)$ and execute $\mathsf{OAccess}^{C_2^*}(1^\lambda, s, k, 0)$.

(b) Let S be the set of locations where $P_{C_1^*,C_2^*,1,s_1}$ is touched as C_1^* and C_2^* are processed as above and F_S be the corresponding changes.

(c) Output $\mathsf{Update}^{\widetilde{C}}(uk, S)$ as S_j.

Indistinguishability follows from the security of the oblivious RAM scheme (as in Section 2.3).

$\mathcal{H}_4$: In this hybrid we change how C_2^* is generated and the changes that are made while the increments are performed. More formally:

We generate (C_2^*, s_2) by executing $\mathsf{OData}(1^\lambda, C_1)$. Additionally the increments are not set using I_1 instead of I_0 as follows:

(a) $F_{S'_b}$ be the changes corresponding to S'_b. For each $i \in S'_0$ and $k \in S'_1$ proceed as follows:
   i. Let $d_i := \mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, \perp)$ and execute $\mathsf{OAccess}^{C_1^*}(1^\lambda, s, i, f_i(d_i))$.
   ii. Similarly, let $d'_k := \mathsf{OAccess}^{C_2^*}(1^\lambda, s, k, \perp)$ and execute $\mathsf{OAccess}^{C_2^*}(1^\lambda, s, k, f_k(d'_k))$.

(b) Let S be the set of locations where $P_{C_1^*,C_2^*,1,s_1}$ is touched as C_1^* and C_2^* are processed as above and F_S be the corresponding changes.

(c) Output $\mathsf{Update}^{\widetilde{C}}(uk, S)$ as S_j.

Indistinguishability follows from the security of the IIO scheme. Here we use the property that any changes made to C_2^* do not affect the functionality of the program $P_{C_1^*,C_2^*,1,s_1}$.

$\mathcal{H}_5$: In this hybrid instead of outputting an obfuscation of $P_{C_1^*,C_2^*,1,s_1}$ we output and obfuscation of $P_{C_1^*,C_2^*,2,s_2}$.

Indistinguishability follows from the security of the IIO scheme. Here we use the property that for all x we have that $C_0(x) = C_1(x)$ and $F_{I_0}(C_0)(x) = F_{I_1}(C_1)(x)$. This in particular implies that $P_{C_1^*,C_2^*,1,s_1}(x) = P_{C_1^*,C_2^*,2,s_2}(x)$ and $F_J(P_{C_1^*,C_2^*,1,s_1})(x) = F_J(P_{C_1^*,C_2^*,2,s_2})(x)$ where $J = (S_1, \ldots S_t)$ as obtained in the previous hybrid.

Observe that at this point we can reverse the hybrids presented above and obtain the distribution $\mathsf{Expt}(\lambda, C_0, C_1, I_0, I_1, 1)$. This proves our claim. □

As before, this transformation is not specific to circuits. In particular, if $(\mathcal{O}, \mathsf{Update})$ is a scheme for Turing machines or RAM programs, then $(\mathcal{O}', \mathsf{Update}')$ is increment-private IIO for the same model. Thus, our transformation, together with theorem 4.2, gives the following result.

Theorem 5.2. *If there exists indistinguishability obfuscation for a class of programs $\mathcal{P} = \{P_\lambda\}$ modeled as either boolean circuits, or (bounded/unbounded input) Turing machines, or RAM programs, then there exists* increment-private *indistinguishability obfuscation (definition 3.2) for $\mathcal{P}$.*

6 The Lower Bound

Point functions. Let $\mathcal{I}_n = \{I_x\}_{x \in \{0,1\}^n}$ denote the family of point functions for points in $\{0,1\}^n$ where n is a (potentially large) polynomial in λ. Function I_x takes as input $y \in \{0,1\}^n$ and outputs 1 if $y = x$ and 0 otherwise.

VBB obfuscation schemes for $\mathcal{I}_n$ are known to exist [Wee05, Can97, BS16]. We show that even for a family as simple as $\mathcal{I}_n$, *incremental* VBB obfuscation does not exist. In fact, we rule this out even for VGB obfuscation, which is weaker than VBB; this strengthens our result.

More specifically, we show that the update algorithm of every incremental VGB obfuscation for $\mathcal{I}_n$ must change $\Omega(n)$ bits for a large fraction of functions $I_x \in \mathcal{I}_n$ even if only *one* bit changes in x.

Theorem 6.1. *Every* VGB *obfuscation scheme for $\mathcal{I}_n$ must have incrementality $\Delta \in \Omega(n)$.*

Proof: Let $(\mathcal{O}, \mathsf{Update})$ be a VGB obfuscation scheme for $\mathcal{I}_n$ with incrementality Δ. Let λ be the security parameter so that $n = n(\lambda) \geq \lambda$ is a polynomial determining the length of points in $\mathcal{I}_n$, and Δ is a function of λ.

The proof proceeds by showing that incrementality "leaks" Hamming distance between updated obfuscations, which, by definition, cannot be leaked by VGB obfuscation. Formally, define the following two distributions:

- $\mathcal{D}_1$: Obtain obfuscations for programs I_x and $I_{x'}$ for a random x by obfuscating I_x and updating it for $I_{x'}$ where x, x' differ in only one position, say first. I.e., sample $x \leftarrow \{0,1\}^n$, $\widetilde{I}_x \leftarrow \mathcal{O}(I_x)$, $\widetilde{I}_{x'} \leftarrow \mathsf{Update}^{\widetilde{I}_x}(1^\lambda, uk, \{1, \mathsf{flip}\})$, and output $(\widetilde{I}_x, \widetilde{I}_{x'})$.
- $\mathcal{D}_2$: Return obfuscations of two random points y_1, y_2 through update. I.e., sample y_1, y_2 uniformly, and obtain $\widetilde{I}_{y_1} \leftarrow \mathcal{O}(I_{y_1})$. Let δ denote the set of locations where y_1, y_2 differ. Then, obtain $\widetilde{I}_{y_2} \leftarrow \mathsf{Update}^{\widetilde{I}_{y_1}}(1^\lambda, \delta)$.

Next, define the following adversarial algorithm:

Algorithm A: on input two strings $(\widetilde{I}_1, \widetilde{I}_2)$, A outputs 1 if the Hamming distance between $\widetilde{I}_1, \widetilde{I}_2$ is at most Δ; otherwise A outputs 0.

By definition of VGB security, there exists a simulator S, a polynomial q, and a negligible function α such that S can simulate the output of A (on any two obfuscated circuits) within distinguishing distance α by making at most q queries to the corresponding circuits.

Let us consider the output of A on input $(\widetilde{I}_x, \widetilde{I}_{x'})$ sampled from $\mathcal{D}_1$. Due to the incrementality of the scheme, these inputs differ in at most Δ locations, and hence A outputs 1 with probability 1. Therefore, by VGB security, $S^{I_x[q], I_{x'}[q]}$ outputs 1 with probability at least $1 - \alpha$.

Next, consider the output of A on input $(\widetilde{I}_{y_1}, \widetilde{I}_{y_2})$ sampled from $\mathcal{D}_2$. Let $p(\Delta)$ be the probability that y_1 and y_2 have Hamming distance at most Δ. Due to the correctness of obfuscation, the Hamming distance between $\widetilde{I}_{y_1}, \widetilde{I}_{y_2}$ cannot less than that between y_1, y_2. Therefore, Hamming distance between $\widetilde{I}_{y_1}, \widetilde{I}_{y_2}$ is less than Δ with probability at most $p(\Delta)$. Thus, A outputs 1 on input $(\widetilde{I}_{y_1}, \widetilde{I}_{y_2})$ with probability at most $p(\Delta)$. By VGB security, $S^{I_{y_1}[q], I_{y_2}[q]}$ outputs 1 with probability at most $\alpha + p(\Delta)$.

Now compare the outputs of $S^{I_x[q], I_{x'}[q]}$ and $S^{I_{y_1}[q], I_{y_2}[q]}$. We have that:

$$\delta := \left| \Pr_x \left[S^{I_x[q], I_{x'}[q]}(1^\lambda, 1^n) = 1 \right] - \Pr_{y_1, y_2} \left[S^{I_{y_1}[q], I_{y_2}[q]}(1^\lambda, 1^n) = 1 \right] \right| \geq 1 - (2\alpha + p(\Delta)).$$

On the other hand, since (x, y_1, y_2) were chosen uniformly and independently, queries of S to (the black-box programs) include a point in $\{x, x', y_1, y_2\}$ with probability at most $4q \cdot 2^{-n}$. Therefore, $\delta \leq 4q2^{-n}$. Comparing the two values of δ we get that $p(\Delta) \geq 1 - (2\alpha + 4q2^{-n}) \implies \Delta \in \Omega(n)$. □

7 Best Possible Incremental Obfuscation

Our lower bound on the incrementality of VGB/VBB obfuscations demonstrates that any incremental scheme must leak the size of incremental updates that take place. An interesting question is if this is all that is leaked. In particular, we investigate the possibility of realizing weaker simulation-baed notions which allow for leakage of the size of the incremental updates.

However, a notion along these lines leaves a lot unexplained. For example, it is not clear if such an effort would yield a meaningful notion general programs. Motivated by such issues and inspired by the notion of "best possible obfuscation" [GR07] for the *single use* setting, we define the notion of "incremental best possible obfuscation" or IBPO.

Definition 7.1 (Incremental Best Possible Obfuscator (IBPO)). *A pair of uniform PPT machines* $(\mathcal{O}, \mathsf{Update})$ *is called an* incremental BPO obfuscator *for a circuit class* $\{\mathcal{C}_\lambda\}$ *if it satisfies the* syntax, correctness *and* incrementality *properties (as in definition 3.1) and the following best possible obfuscation property:*

For any (not necessarily uniform) PPT adversaries A, there exists a simulator $\mathcal{S}$ and a negligible function α, such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_1, C_2 \in \mathcal{C}_\lambda$ (with $|C_1| = |C_2|$), for every polynomial t and for all pairs of update sequences I_0, I_1 (with $S_{0,i} = S_{1,i}$ for each $i \in \{1, \ldots, t\}$) of length t we have that if $C_0(x) = C_1(x)$ and $F_{I_0}(C_0)(x) = F_{I_1}(C_1)(x)$ for all inputs x, then

$$\left| \Pr\left[A(\widetilde{C}_0, \widetilde{C}_0^1, \ldots, \widetilde{C}_0^t) = 1\right] - \Pr\left[\mathcal{S}(C_1, I_1) = 1\right] \right| \leq \alpha(\lambda)$$

where $(\widetilde{C}_0, sk) \leftarrow \mathcal{O}(\lambda, C_0)$, and $(\widetilde{C}_0^1, \ldots, \widetilde{C}_0^t) \leftarrow \mathsf{Update}^{\widetilde{C}_0}(sk, I_0)$.

Informally, this definition guarantees that any information that can be efficiently obtained from the obfuscation $\widetilde{C}_0$ along with the obfuscation increments $\widetilde{C}_0^1, \ldots, \widetilde{C}_0^t$, can also be extracted efficiently (i.e., simulated) from any equivalent circuit of a similar size C_1 and corresponding equivalent updates I_1 of similar size. We now prove the following theorem.

Theorem 7.1. $(\mathcal{O}, \mathsf{Update})$ *is* incremental BPO obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if and only if it is *increment-private* IIO obfuscator for $\{\mathcal{C}_\lambda\}$.

Proof Sketch: We need to prove that an IBPO scheme is also a *increment-private* IIO scheme and the other way around. We start with the first direction.

By the definition of best possible obfuscation we have that the distributions $A(\widetilde{C}_0, \widetilde{C}_0^1, \ldots, \widetilde{C}_0^t)$ and $\mathcal{S}(C_1, I_1)$ are close. Similarly the distributions $A(\widetilde{C}_1, \widetilde{C}_1^1, \ldots, \widetilde{C}_1^t)$ and $\mathcal{S}(C_1, I_1)$ are close. In the above expressions for $b \in \{0, 1\}$, $(\widetilde{C}_b, sk_b) \leftarrow \mathcal{O}(\lambda, C_b)$, and $(\widetilde{C}_b^1, \ldots, \widetilde{C}_b^t) \leftarrow \mathsf{Update}^{\widetilde{C}_b}(sk_b, I_b)$. These two facts together imply that the distributions $A(\widetilde{C}_0, \widetilde{C}_0^1, \ldots, \widetilde{C}_0^t)$ and $A(\widetilde{C}_1, \widetilde{C}_1^1, \ldots, \widetilde{C}_1^t)$ are close, implying that $(\mathcal{O}, \mathsf{Update})$ is a *increment-private* incremental indistinguishability obfuscator.

Next we sketch the argument for the other direction. Our simulator $\mathcal{S}$ on input C_1, I_1 computes $(\widetilde{C}_1, sk_1) \leftarrow \mathcal{O}(\lambda, C_1)$, and $(\widetilde{C}_1^1, \ldots, \widetilde{C}_1^t) \leftarrow \mathsf{Update}^{\widetilde{C}_1}(sk_1, I_1)$ and outputs $(\widetilde{C}_1, \widetilde{C}_1^1, \ldots, \widetilde{C}_1^t)$. The indistinguishability of this from obfuscation of C_0 and obfuscation increments for I_0 follows directly from the security of *increment-private* incremental indistinguishability. □

8 Extensions and Future Work

We discuss three possible extensions of our results: *non-sequential* model of updates, *adaptive* updates, and new types of update operations and other refinements.

Non-sequential updates. In our current model, an update at time step i is applied to the obfuscation at time step $i-1$. A more flexible approach would be to allow the update to any obfuscation that is previously present in the history. This results in a "tree like" structure for updates instead of the "line" for sequential updates. This model is interesting for situations where the copies of software reside on several remote machines and may be updated at different times.

Our constructions can be easily adapted for this setting as well. Specifically, in our basic IIO construction, instead of choosing r at random, let it be an encryption of a random value. The update algorithm will simply encrypt a fresh value each time (instead of adding 1 to the previous value). All other operations are performed as before.

The key observation is that, since the values are now encrypted, sequential values look indistinguishable from random. Therefore, in the security proof, we will first change the random values to sequential, and then proceed exactly as before. Note that the the obfuscated program does not need to know the values in the encryption, and hence does not need the secret key. Only the update algorithm needs the secret key. The proof now additionally uses the semantic security of encryption (along with NIZK proofs as before) to switch to sequential values.

Adaptive updates. As mentioned earlier, our constructions do not achieve adaptive security where future updates are chosen adversarially based on previous obfuscations. We leave this as an interesting open problem.

More general updates, and other refinements. We did not consider updates which may *increase the size* of the underlying programs. In many settings, the size of the program would likely increase after updates. Likewise, we also did not explore other refinements such as tamper-proof security (where the obfuscation to be updated may not be "correct" due to tampering by the adversary). It would be interesting to explore these directions in future.

References

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating Circuits via Composite-Order Graded Encoding. In *TCC*, pages 528–556, 2015.

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-Inputs Obfuscation and Applications, 2013. IACR Cryptology ePrint Archive: `http://eprint.iacr.org/2013/689.pdf`.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability Obfuscation from Compact Functional Encryption. In *CRYPTO*, pages 308–326, 2015.

[App14] Benny Applebaum. Bootstrapping Obfuscators via Fast Pseudorandom Functions. In *ASIACRYPT*, 2014.

[AS16] Prabhanjan Ananth and Amit Sahai. Functional Encryption for Turing Machines. In *TCC*, 2016.

[BBC$^+$14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for Evasive Functions. In *TCC*, 2014. Preliminary version on Eprint 2013: `http://eprint.iacr.org/2013/668.pdf`.

[BC10] Nir Bitansky and Ran Canetti. On Strong Simulation and Composable Point Obfuscation. In *CRYPTO*, pages 520–537, 2010.

[BCC$^+$14] Nir Bitansky, Ran Canetti, Henry Cohn, Shafi Goldwasser, Yael Tauman Kalai, Omer Paneth, and Alon Rosen. The Impossibility of Obfuscation with Auxiliary Input or a Universal Simulator. In *CRYPTO*, 2014.

[BCG$^+$11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program Obfuscation with Leaky Hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BCKP14] Nir Bitansky, Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On Virtual Grey Box Obfuscation for General Circuits. In *CRYPTO*, pages 108–125, 2014.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On Extractability (a.k.a. Differing-Inputs) Obfuscation. In *TCC*, 2014. Preliminary version on Eprint 2013: `http://eprint.iacr.org/2013/650.pdf`.

[BCP16] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious Parallel RAM. In *TCC*, 2016.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *STOC*, pages 103–112, 1988.

[BGG94] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental Cryptography: The Case of Hashing and Signing. In *CRYPTO*, pages 216–233, 1994.

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental Cryptography and Application to Virus Protection. In *STOC*, pages 45–56, 1995.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting Obfuscation against Algebraic Attacks. In *EUROCRYPT*, 2014.

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct Randomized Encodings and their Applications. In *STOC*, pages 439–448, 2015.

[BKY01] Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental Unforgeable Encryption. In *FSE*, pages 109–124, 2001.

[BM97] Mihir Bellare and Daniele Micciancio. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In *EUROCRYPT*, pages 163–192, 1997.

[BP15a] Nir Bitansky and Omer Paneth. ZAPs and Non-Interactive Witness Indistinguishability from Indistinguishability Obfuscation. In *TCC*, pages 401–427, 2015.

[BP15b] Elette Boyle and Rafael Pass. Limits of Extractability Assumptions with Distributional Auxiliary Input, 2015. Preliminary version: `http://eprint.iacr.org/2013/703.pdf`.

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect Structure on the Edge of Chaos. In *TCC*, 2016.

[BR13] Zvika Brakerski and Guy N. Rothblum. Obfuscating Conjunctions. In *CRYPTO*, pages 416–434, 2013.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding. In *TCC*, 2014. Preliminary version on Eprint at `http://eprint.iacr.org/2013/563.pdf`.

[BS16] Mihir Bellare and Igors Stepanovs. Point-Function Obfuscation: A Framework and Generic Constructions. In *TCC*, 2016. Preliminary version at IACR Eprint Report 2015/703: `http://eprint.iacr.org/2015/703.pdf`.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability Obfuscation from Functional Encryption. In *FOCS*, 2015.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty Key Exchange, Efficient Traitor Tracing, and More from Indistinguishability Obfuscation. In *CRYPTO*, pages 480–499, 2014.

[Can97] Ran Canetti. Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information. In *CRYPTO*, pages 455–469, 1997.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability Obfuscation of Iterated Circuits and RAM Programs. In *STOC*, 2015.

[CLP14] Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure ORAM with õ($\log^2$ n) overhead. In *ASIACRYPT*, pages 62–81, 2014.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-Round Concurrent Zero-Knowledge from Indistinguishability Obfuscation. In *CRYPTO*, pages 287–307, 2015.

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of Hyperplane Membership. In *TCC*, pages 72–89, 2010.

[CV13] Ran Canetti and Vinod Vaikuntanathan. Obfuscating branching programs using black-box pseudo-free groups. *IACR Cryptology ePrint Archive*, 2013:500, 2013.

[Fis97a] Marc Fischlin. Incremental Cryptography and Memory Checkers. In *EUROCRYPT*, pages 293–408, 1997.

[Fis97b] Marc Fischlin. Lower Bounds for the Signature Size of Incremental Schemes. In *FOCS*, pages 438–447, 1997.

[FLS99] Feige, Lapidot, and Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29, 1999.

[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 *FOCS*, pages 40–49, 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the Implausibility of Differing-Inputs Obfuscation and Extractable Witness Encryption with Auxiliary Input. In *CRYPTO*, pages 518–535, 2014.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the Impossibility of Obfuscation with Auxiliary Input. In *FOCS*, pages 553–562, 2005.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[Gol87] Oded Goldreich. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *STOC*, pages 182–194, 1987.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect Non Interactive Zero Knowledge for NP. In *EUROCRYPT*, pages 339–358, 2006.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, pages 194–213, 2007.

[Had00] Satoshi Hada. Zero-Knowledge and Code Obfuscation. In *ASIACRYPT*, pages 443–457, 2000.

[Had10] Satoshi Hada. Secure obfuscation for encrypted signatures. In *EUROCRYPT*, pages 92–112, 2010.

[HRSV07] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *TCC*, pages 233–252, 2007.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a Random Oracle: Full Domain Hash from Indistinguishability Obfuscation. In *EUROCRYPT*, pages 201–220, 2014.

[HW15] Pavel Hubacek and Daniel Wichs. On the Communication Complexity of Secure Function Evaluation with Long Output. In *ITCS*, pages 163–172, 2015.

[IB] Information is Beautiful. `http://www.informationisbeautiful.net/visualizations/million-lines-of-code`.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-Coin Differing-Inputs Obfuscation and Its Applications. In *TCC*, pages 668–697, 2015.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability Obfuscation for Turing Machines with Unbounded Memory. In *STOC*, 2015.

[KMN$^+$14] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *FOCS*, pages 374–383, 2014.

[LO13] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *TCC*, pages 377–396, 2013.

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, pages 20–39, 2004.

[Mic97] Daniele Micciancio. Oblivious Data Structures: Applications to Cryptography. In *STOC*, pages 456–464, 1997.

[MO14] Antonio Marcedone and Claudio Orlandi. Obfuscation $\Rightarrow$ (IND-CPA security !$\Rightarrow$ circular security). In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 77–90, 2014.

[MPRS12] Ilya Mironov, Omkant Pandey, Omer Reingold, and Gil Segev. Incremental Deterministic Public-Key Encryption. In *EUROCRYPT*, pages 628–644, 2012.

[NY90] Moni Naor and Moti Yung. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *STOC*, pages 427–437, 1990.

[O'N10] Adam O'Neill. Definitional Issues in Functional Encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New Realizations of Somewhere Statistically Binding Hashing and Positional Accumulators. In *ASIACRYPT*, 2015.

[Ost90] Rafail Ostrovsky. Efficient Computation on Oblivious RAMs. In *STOC*, pages 514–523, 1990.

[PPS15] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based Non-black-box Simulation and Four Message Concurrent Zero Knowledge for NP. In *TCC*, 2015.

[SCSL11] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with O((logN)3) Worst-Case Cost. In *ASIACRYPT*, pages 197–214, 2011.

[SvDS$^+$13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *ACM CCS*, pages 299–310, 2013.

[SW14] Amit Sahai and Brent Waters. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. In *STOC*, pages 475–484, 2014.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.