Bounded Collusion ABE for TMs from IBE

Rishab Goyal[*] Ridwan Syed[†] Brent Waters[‡]

Abstract

We give an attribute-based encryption system for Turing Machines that is provably secure assuming only the existence of identity-based encryption (IBE) for large identity spaces. Currently, IBE is known to be realizable from most mainstream number theoretic assumptions that imply public key cryptography including factoring, the search Diffie-Hellman assumption, and the Learning with Errors assumption.

Our core construction provides security against an attacker that makes a single key query for a machine T *before* declaring a challenge string w^* that is associated with the challenge ciphertext. We build our construction by leveraging a Garbled RAM construction of Gentry, Halevi, Raykova and Wichs [GHRW14a]; however, to prove security we need to introduce a new notion of security called iterated simulation security.

We then show how to transform our core construction into one that is secure for an a-priori bounded number $q = q(\lambda)$ of key queries that can occur either before or after the challenge ciphertext. We do this by first showing how one can use a special type of non-committing encryption to transform a system that is secure only if a single key is chosen before the challenge ciphertext is declared into one where the single key can be requested either before or after the challenge ciphertext. We give a simple construction of this non-committing encryption from public key encryption in the Random Oracle Model. Next, one can apply standard combinatorial techniques to lift from single-key adaptive security to q-key adaptive security.

1 Introduction

Attribute-based encryption (ABE) [SW05] provides a method for encrypting data which allows for sharing at a much finer-grained level than standard public key cryptography. In an ABE system one associates a ciphertext with an attribute string w when encrypting message m to form a ciphertext ct. A secret key (as issued by some authority) is associated with a predicate function f. A decryption algorithm using sk_f on the ciphertext will be able to return the message m if and only if $f(w) = 1$.

The initial and many subsequent ABE constructions (e.g. [GPSW06, BSW07]) provided functionality for when f was a boolean formula or circuit that would operate over a fixed set of attributes. This works well for the setting when an attribute string could say represent a record that was of a fixed form, however, would not work as well in a setting where we want the attribute string structure to be less rigid and of arbitrary length. Initial progress towards resolving such issue was by Waters [Wat12] who provided the first ABE construction for a uniform model of computation where the attribute string $w \in \{0,1\}^*$ could be an arbitrary length string and f is a Deterministic Finite Automata (DFA). A user in such a setting can decrypt a ciphertext whenever the DFA f accepts w.

Since then, ABE systems in uniform models of computation have been very well studied, with subsequent works roughly falling into the following three categories grouped by the hardness assumption.

[*] MIT. Email: goyal@utexas.edu. Work done in part while at UT Austin, supported by IBM PhD Fellowship. Research supported in part by NSF CNS Award #1718161, an IBM-MIT grant, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR00112020023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

[†] University of Texas at Austin.

[‡] University of Texas at Austin and NTT Research. Email: bwaters@cs.utexas.edu. Supported by NSF CNS-1908611, CNS-1414082, DARPA SafeWare, Packard Foundation Fellowship, and Simons Investigator Award.

- The ABE construction of Waters [Wat12] for DFAs was built from bilinear maps and was collusion resistant in that it allowed for an unbounded number of private keys to be issued, but was only selectively secure in that the attacker was required to submit a challenge string w^* before seeing the public parameters of the system. Unlike constructions where the length of w is fixed by the security parameter, there is no known way of generically moving from selective to adaptive security using complexity leveraging and assuming sub-exponential hardness. Subsequent works [Att14, Att16, AC17, GWW19, AMY19b, GW20] in the bilinear map setting improved upon the security arguments in this setting as well as gave "ciphertext-policy" variants of the construction.

- A second cohort of constructions [BCP14, IPS15, KLW15, BCG$^+$18] arise by constructing ABE for Turing Machines from obfuscation culminating in the work of Ananth and Sahai [AS16] that achieves functional encryption for Turing Machines from indistinguishability obfuscation with no a-priori bound on the input size or machine description. We refer the reader to [AS16] for a discussion of the tradeoffs present in prior works.

- In a third line of work, Agrawal and Singh [AS17] gave a construction of a single-key secure functional encryption scheme provably secure under the Learning with Errors (LWE) [Reg05] assumption. They could prove security only when the single private key was requested before the challenge ciphertext. Additionally, in their model the encryptor had to specify the maximum time t that the Turing Machine computation is allowed to run for during decryption. The work of Gentry et al. [GHRW14b] also gave a construction for single-key secure functional encryption for RAM computation from single-key secure functional encryption for circuits and garbled RAM, but the key generator not only takes the RAM program as input but the input size and run-time bound as well. Thus, the encryption algorithm could only encrypt messages of a-priori fixed length.

  For unbounded collusions, Boyen and Li [BL15] gave constructions for DFAs from the LWE assumption and Agrawal, Maitra, and Yamada [AMY19a] did this for NFAs, but in the secret key setting. Ananth, Fan and Shi [AFS19] give an LWE solution for unbounded collusions in the problem of constructing ABE for RAM Turing Machines. However, the maximum number of machine steps is given as a parameter to the setup algorithm and will serve as a bound for the system.

One common thread of the above works is that they all depended upon a specific number theoretic setting. Even in the case of indistinguishability obfuscation, the best known construction in the recent breakthrough work [JLS21] relies on a careful combination of *multiple* specific algebraic assumptions.

Here we pursue a new direction of obtaining Attribute-Based Encryption for uniform computation models from general assumptions. In particular, we provide solutions that assume Identity-Based Encryption (IBE) [Sha85, BF01]. We believe IBE is a good platform for this pursuit as it is known under most "mainstream" number theoretic assumptions that imply public key cryptography such as factoring, search Diffie-Hellman, and Learning with Errors [BF01, Coc01, GPV08, DG17a].

Our Results In this work we show how to achieve Attribute-Based Encryption for Turing Machines that is adaptively secure against any attacker that requests at most $q = q(\lambda)$ private keys where q can be any polynomial function determined at system setup. Our work is logically broken into two parts.

In the first part we develop our core construction which is an ABE system for Turing Machines secure against any poly-time attacker that requests a single key *before* declaring the attribute string w^* for a challenge ciphertext. In this system the maximum running time t of the Turing Machine is determined by the encryption algorithm as in [AS17].

Our approach leverages a garbled RAM construction due to Gentry, Halevi, Raykova, and Wichs (GHRW) [GHRW14a] which intuitively allows a sequence of t garbled programs to run while maintaining a persistent database across invocations. We combine this with an IBE system in a spirit motivated by [DG17a, DG17b, GS17, GHMR18] which allows us to securely evaluate a Turing Machine computation that delivers the message on decryption only if the machine accepts. One challenge we encounter is that the GHRW definition of simulation security is defined as distinguishing between a real garbled RAM and a simulated one over the *whole* computation. However, this notion of simulation is not fine-grained enough for our purposes. Instead we need to introduce a notion of *iterated simulation security* where it is hard to distinguish whether the first i or $i+1$ programs were simulated, and not just indistinguishability of the entire computation. Fortunately, we were able to show that the existing GHRW construction satisfies this notion of security as well. Since the GHRW Garbled RAM itself only relies on IBE, the entire security of our construction still depends on IBE only.

The second part of our work is focused on moving from a single key system that is limited to coming before the challenge ciphertext to a q-query system that allows for key requests to come at arbitrary times. We first tackle the question of giving flexibility for when the key query is placed. To do this we introduce a very relaxed form of non-committing encryption [CFGN96, DN00, KO04] where the non-committing simulation property must hold when an attacker is given the secret key of a public key encryption system. (But not the randomness for encryption and key generation as in [CFGN96].) We show that this form of non-committing encryption is strong enough to transform our single key ABE system into one where the key query can come before or after the challenge ciphertext. We follow this transformation with another one to allow for q queries by applying standard combinatorial techniques. To complete the transformation we provide a simple construction for such a non-committing encryption scheme from public key encryption in the case of bounded length messages, while for unbounded messages we additionally rely on hash function modeled as a Random Oracle [BR93].

We want to emphasize that prior to our work, all other ABE systems in uniform computation models either relied on specific algebraic assumptions, or powerful notions such as succinct function encryption and program obfuscation. That is, unlike for non-uniform models where we have numerous generic constructions (e.g., [SS10, GVW12, AV19]) from general assumptions such as public-key encryption, it was believed that relying on algebraic manipulation or powerful encryption/obfuscation primitives might be necessary for handling uniform models where the attribute space is not statically fixed. Ours is the first work that dispels this belief. Thus, we want to highlight that one of our main take-away messages is that the central source of hardness is only full collusion resistance, and not the underlying model of computation in functional encryption.

Organization We begin by providing a technical overview of our approach in the next section. Since our bounded collusion secure ABE system for TM predicates relies extensively on garbled RAM, thus we start by describing our notations and other standard cryptographic primitives in Section 2, and recalling the definition of garbled RAM along with the our proposed iterated simulation security definition in Section 3. In Section 4, we describe our main construction for ABE for TMs via the usage of IBE and garbled RAM. Later in Appendix A, we describe our relaxed notion of non-committing encryption, a simple construction, and its usage in the transformation from key-selective secure ABE to standard security. We then follow this by providing a transformation to go from 1-query security to q-query security in Appendix B. Finally, in Appendix C, we show that the GHRW garbled RAM scheme [GHRW14a] satisfies the iterated simulation security that we rely on.

1.1 Technical Overview

The overview is split into two parts where we first describe our core construction from garbled RAM and identity-based encryption. This construction gives us 1-query secure ABE scheme for TMs where the secret key query must be made before obtaining the challenge ciphertext. In the second part, we describe how to lift the security of any ABE scheme for TMs, which guarantees security in this restricted 1-query key-selective setting, to provide general bounded collusion security via a sequence of generic black-box transformations. We conclude with some interesting open directions for further investigation.

Core construction: 1-query key-selective ABE for TMs

As highlighted in the previous section, the aspect of ABE systems in a uniform model of computation (such as Turing Machines in our case) that makes it quite appealing is that it allows an encryptor to specify an a-priori *unbounded* length attribute during encryption while still enabling a fixed decryption key to work on all such varying length ciphertexts. From a mechnical perspective, this suggests that ciphertexts for such computation models should possess a self-reducibility feature. By this we mean that in a structural sense the ciphertext could be broadly divided into two components — one being reusable, while other being execution time-step dependent. Here we expect the reusable component to store the current state of computation during decryption, and the time-step dependent component to self-reduce, i.e. to be used piece-by-piece (with each piece annotated with an execution time-step) for updating the reusable component thereby guiding the decryption process to either the plaintext or failure depending on the predicate.

Comparing this mechnical view with that for ABE systems in a non-uniform model of computation, the stark difference comes up in the implementation of the reusable component which for non-uniform models could mostly be relegated to the predicate key instead, since each key already fixes an upper bound on the number of such re-use operations/computation steps. This restriction is very consequential both for the construction as well as proof purposes. Circumventing such unbounded reusability problems under standard cryptographic assumptions has been a difficult task so far.

Our approach is to start with the simplest goal which is of security in presence of a single key corruption where the challenge attribute as well as the TM key queried must be selectively chosen by the attacker.[1] Now we already know that the concept of garbled circuits [Yao86] have been tremendously useful in building bounded collusion secure ABE systems in a non-uniform circuit model (and bounded collusion secure functional encryption more generally) [SS10, GVW12]. A natural question is whether the same could be stated if we switch to a uniform computation model such as TMs since, despite the strengthening of the computation model, the targetted encryption primitive still provides only an all-or-nothing style guarantee.

A building block construction. First, note that plugging in TMs as the model of computation in the mechnical picture described above, we get the reusable ciphertext component to correspond to the tape of the TM being operated on, while the time-step dependent component is being used to emulate a step-by-step execution of the TM itself. Next, consider a highly simplified TM model where the number of states as well as the size of the TM tape are a-priori fixed polynomials, say N and L respectively. (Although this simplified model no longer resembles our targetted TM model of computation, this will serve as a good starting point to convey the main idea which we afterwards extend to capture the more general model.) It turns out that for such a model there is a natural candidate ABE system from just plain public-key encryption and garbled circuits.

Let us start by sharing our methodology for encrypting a message m under attribute string w with time bound t.[2] At a high level, the idea is to let encryptor create a sequence of t step circuits, where each step circuit takes as input the entire state of TM (which contains the current state, location of the tape header, and the entire tape of the TM) and it performs one execution step (that is, applies one transition) and its output is the entire TM state after this execution step (that is, output state, tape header and full tape contents). Here the last step circuit simply outputs the encrypted message m if the execution lands in accepting state. An encryptor then garbles each such step circuit starting from the last one (that is, t-th step circuit first), and encodes the wire labels for the i-th garbled step circuit in the $(i-1)$-th step circuit. Now each garbled circuit must not output the wire labels in the clear, thus it instead encrypts the labels corresponding to the TM state for next step circuit under a group of carefully selected PKE public keys. The idea here is that during setup we sample a pair of PKE public-secret keys for each state transition[3], and a secret key for any TM in this system consists of a sequence of PKE secret keys corresponding to all the state transition supported by the corresponding TM.

Intuitively, a ciphertext consists of a sequence of t garbled circuits, and a secret key consists of a polynomial-sized set of PKE secret keys such that to decrypt a ciphertext, one evaluates each garbled circuit in a sequential order thereby revealing the state of the TM computation after each execution step

[1] As we later show, such a core encryption scheme with such simple and weak security guarantees could be generically amplified to better and more general bounded collusion security guarantees.

[2] Recall that in this work we require the encryptor to provide an upper bound on the running time of the TM.

[3] Since the number of states is polynomially bounded, thus this is efficient.

encrypted under appropriate PKE public keys. An honest decryptor can always recover the relevant garbled circuit wires along its path of computation, and finally recovers the message if the machine accepts within the ciphertext specified time bound t. One could also provide security of this construction by a straightforward sequence of hybrids where the simulator would, instead of computing the garbled circuits honestly, replace each garbled circuit with a simulated garbled circuit one by one. And, since our assumption was the tape size and number of states to be polynomially bounded, thus this scheme is efficient (i.e., runs in polynomial time) as well.

Looking ahead, the above approach serves as a good warm-up construction for our core construction which does not suffer from the above limitations. Very briefly, we make the following observations. First, note that the above construction does not exploit the fact that a given step circuit does not need to look at the entire TM tape, but instead it needs to make changes right next to the location of tape header. Thus, instead of passing around the entire TM tape to each step circuit, we can maintain a persistent storage that contains the full TM tape while each step circuit only affects a few particular locations in the storage. To this end, we replace our usage of garbled circuits with garbled RAMs [LO13] thereby bypassing the above problem. Second, we assumed that the number of states are a-priori polynomially bounded. This was mainly needed so to avoid the exponential blow-up due to the exponential state space which we could not hope to generically encode using only public-key encryption. To solve this issue, we use an identity-based encryption scheme to provide a succinct mechanism to encode the state transitions without this exponential blow-up. Similar ideas of encoding exponential size strings succinctly have been used in numerous other contexts [GKW16, DG17a, GHMR18].

Main construction. Before moving to a more technical description of our scheme, we fix our notation and interpretation of a TM. This will help in understanding our main construction more clearly. We consider a TM to be represented by a large set T of state transitions $\left\{(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir})\right\}$, where each transition is associated with an input state q^{in}, the input bit read b^{in}, the output state q^{out}, the bit to written b^{out}, and the direction dir in which the tape head moves. Also, let each state q be represented as an n-bit string.

As hinted previously, a central component of our construction is the notion of garbled RAMs. Recall that a RAM program P gets random access to a *large* memory D (upon which it can perform arbitrary reads and writes) along with a short input x, and at the end of its computation it produces an output y. Here we will be interested in multi-program versions of RAM programs where given a sequence of RAM programs $P_1, \ldots, P_\ell$ and corresponding short inputs $x_1, \ldots, x_\ell$, and an initial memory D, the programs are run in succession on their respective inputs wherein say program P_i outputs some result y_i and updates the database D which is then used by the next program P_{i+1}.

Garbled RAMs. The notion of garbled RAMs is a generalization of circuit garbling to RAM programs, where the memory owner first garbles the memory D generating a pair of garbled database $\widetilde{D}$ along with a garbling key k_D. The garbling key k_D can then be used to garble any RAM program P with respect to program index j to produce a garbled program $\widetilde{P}$ along with input labels $\{\mathsf{lab}_{i,b}\}_{i,b}$. Here the program index j is meant to capture the number of programs that have been run (including P).[4] For example, to garble the previously defined sequence of ℓ RAM programs, when the garbling party runs the program garbling procedure for program P_j it specifies index j as the program index since it wants P_j to be the j-th RAM program being evaluated in the sequence. Also, as in the case of circuit garbling, to evaluate a garbled program $\widetilde{P}$ with labels $\{\mathsf{lab}_{i,b}\}_{i,b}$ on an input x, the evaluator selects the labels corresponding to bits of x, i.e. $y = \mathsf{Eval}^{\widetilde{D}}(\widetilde{P}, \{\mathsf{lab}_{i,x_i}\}_i)$ where y is the output of running P on x with memory D. The standard security property considered in most prior works [LO13, GHRW14a, LO14, GHL+14, GLOS15, GLO15] is of static (full) simulation security wherein an adversary must not be able to distinguish a sequence of honestly garbled RAM programs and database from a simulated sequence of programs and database, where the simulator only knows the corresponding outputs $\{y_i\}_i$ of each RAM program (but not the database D, or any of the individual programs P_i, or their corresponding inputs x_i).

[4] For the purposes of a technical overview, we significantly simplify and relax the notation. Here we consider each program to be of fixed length, and not take time range among other things as additional inputs. Later in the main body, we define it in full generality.

Core construction: *switching from garbled circuits to garbled RAMs.* With all the notation set, our main construction is very simple to follow. The setup of our system simply corresponds to sampling a IBE master public-secret key pair. (Recall that previously in our simplified building block construction, the setup was sampling a large number of PKE public-secret key pairs. As we noted then, here we use IBE instead of do the same more efficiently.) Next, to generate a secret key for a TM represented by a set T of transitions, the key generator encodes each transition $(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir}) \in T$ into $(n+2)$ distinct identities. (The identity-encodings we employ are the well known bit-decomposition style encodings where one encodes the output state q^{out} bit-by-bit into n strings of the form $(i, q^{\mathsf{out}}[i]) \in [n] \times \{0,1\}$.) Here n of these IDs jointly encode the output state q^{out}, while the other two encode the output bit to written b^{out}, and the direction dir separately.

The encryption algorithm in our core construction follows a similar paradigm to that described in our building block construction with the only major change being we move to using garbled RAMs instead. Concretely, to encrypt a message m under an unbounded length attribute string $w \in \{0,1\}^*$ with time bound t, the encryptor first creates an empty memory D of size $t+1$.[5] It then writes the attribute w on the RAM memory D, and garbles it to get the corresponding garbled database $\widetilde{D}$. (Basically this memory is used as the tape of the TM embedded in the predicate keys during decryption.) Next, the encryptor creates a sequence of t RAM programs where the i-th program takes as input the TM state q^{in}, bit to be written b^{out}, and the direction dir that was output by the previous (i.e., $(i-1)$-th) program/TM transition. Given these inputs, the RAM program writes the bit b^{out} at the current tape header, updates the tape header location depending on dir, and encrypts the garbled labels for the next (i.e., $(i+1)$-th) RAM program under appropriate identities. (Note that here we crucially rely on our bit-decomposition style identity-encodings of the output state while encrypting the next program labels.) Thus, a ciphertext contains t such garbled RAM programs in which the programs are garbled one-by-one from the last to first since each program contains labels for the next successive garbled program. Connecting this to original mechnical viewpoint, the garbled database should be though of as the reusable ciphertext component while the garbled programs as the time-step dependent components. During decryption, an evaluator simply decrypts the wire labels depending on the current state of its TM execution and evaluates the garbled programs to recover encryptions of the wire labels for the next program. Doing this successively, an evaluator recovers the message if its TM accepts the attribute word within the ciphertext specified time bound.

Security: *how to prove it?* Although the above simple scheme seems to be secure when an adversary makes only a single key query and that too before receiving the challenge ciphertext, proving the same seems a bit challenging. This stems from the fact that a natural proof strategy seems to be incompatible with the full simulation security guaranteed by the underlying garbled RAM scheme. To better understand this, first recall that the garbled RAM security property for multi-program version states that no adversary can distinguish between a sequence of honestly garbled RAM programs (along with half of the honestly computed corresponding garbled labels) from a sequence of simulated garbled RAM programs (again along with half of the simulated garbled labels), where the garbled labels provided depend on the input to be feeded to each RAM program. Next, observe that in our construction, the RAM programs which we garble are not independent programs but instead each RAM program in our construction directly depends on the garbling of the next RAM program in the sequence (since the i-th RAM program contains labels for the $(i+1)$-th RAM program). Juxtaposing these two facts, we get that no reduction algorithm in the proof could even statically define the sequence of RAM programs it wants garbled without interacting with the garbled RAM challenger.

Thus, this circularity/interdependence prevents a natural proof strategy from working. But it turns out that the problem is a bit deeper than what one can perceive at this point. That is, suppose we could somehow make the RAM programs (that we want to garble) fully independent, the problem is that the underlying sequence of RAM programs that we want to simulate will still be executing the TM step-by-step where each program reveals the labels for the next garbled program, thus a reduction algorithm can only simulate

[5] Since the ciphertexts need only be decryptable by keys whose corresponding TMs accept the word within time t, thus the encryptor only needs to instantiate the database with t bits of memory. To be fully accurate, we actually a little more memory for storing the TM state which we discuss later in the main body.

the garbled programs one at a time, and not all at once. Let us clarify this second issue further by first suggesting a modification to our current construction to solve the first interdependence problem.

The modification to our construction for solving this interdependence problem is to sample a fresh PRF keys for eachlabel of the garbled RAM program at the beginning, and instead of letting a RAM program output encryptions of the labels for the next program, we make each program output encryptions of the corresponding PRF keys. Once we set the underlying RAM programs this way, we garble them and to tie them together we encrypt the labels for the $(i+1)$-th RAM program under PRF keys hardwired in the i-th RAM program. Intuitively, this means evaluating the garbled RAM programs an evaluator learns encryptions of some of the PRF keys which are then used to recover the garbled labels outside of this garbled RAM structure.

Getting back to proving security, the problem we still encounter is that as a reduction algorithm it is unclear on how to simulate all the garbled RAM programs at once, since for simulation the reduction needs to able to generate the ciphertext given only half of the wire labels, but those wire labels are encrypted under PRF keys which are hardwired inside each RAM program. Therefore, for a proof to go through a reduction algorithm needs to first remove information about half of garbled labels from the ciphertexts for which it needs to remove the information about half of the corresponding PRF keys which means the reduction must be able to simulate the garbled programs instead which is what we were trying to do in the first place. This circularity stems from the fact that the garbled RAM full simulation security only guarantees security when all the garbled programs are being simulated at the same time, instead of being partially/sequentially simulated.

Strengthening garbled RAM security. To fix the above problem we introduce a stronger security notion for garbled RAMs which we call *iterated simulation security*.[6] To us, it seems a more natural notion of security for multi-program garbled RAM versions, and also captures the kind of garbled RAM security we need for our proof to go through. We describe it in detail later in Section 3, but very briefly it states that there exists an efficient simulator such that for any sequence of ℓ programs and inputs, it is hard to distinguish between simulations of the first i programs and inputs along with honest garblings of the remaining $\ell - i$ programs from simulations of the first $i+1$ programs and inputs along with honest garblings of the remaining $\ell - i - 1$ programs. That is, partial executions of the multi-program garbled RAMs are also simulatable.

Plugging in the strengthened garbled RAM security property, we are able to prove security of our ABE scheme by organizing an iterated hardwiring-style proof strategy where we start by simulating the first garbled program, then remove the information about labels for the next program by relying on PRF security (and the fact that only half of the PRF keys are needed to simulate the first garbled program), and keep on interleaving garbled RAM security with PRF security to eventually remove the plaintext information whenever the underlying TM does not accept the attribute word.

In order to complete the proof, we need to construct such a garbled RAM scheme that achieves our notion of iterated simulation security. Fortunately, we were able to show that most existing garbled RAM schemes already are secure under this partial simulation framework. Later in Appendix C, we show that the IBE-based garbled RAM construction in [GHRW14a] is an iterated simulation secure garbled RAM scheme. Although we believe that garbled RAM construction of Garg et al. [GLOS15] from one-way functions can also be proven to be secure in this iterated simulation framework, since our ABE construction already relies on IBE thus we chose to provide a proof of iterated simulation security for the simpler IBE-based scheme of Gentry et al. [GHRW14a].

Lifting the core construction to q-query adaptive security

After a closer look at the proof overview provided for our core construction, the reason behind our construction only enabling a proof in key-selective model (that is, where the key query must be made before

[6] Although prior works [LO13, GHRW14a, LO14, GHL+14, GHRW14b, GLOS15, GLO15, KLW15, BCG+18] have studied other adaptive and reusable variants of garbled RAM security notions, our notion of iterated simulation security has not yet been explicitly studied previously to the best of our knowledge.

receiving the challenge ciphertext) becomes apparent. Very briefly, the bottleneck is that the reduction algorithm needs to know the current state of partial TM execution while embedding the challenge ciphertext with partially simulated components. Thus, the reduction must know the TM of the key query before creating the challenge ciphertext.

Now instead of modifying our core construction to resolve this bottleneck, we instead observe that if the adversary gets to corrupt at most one key, then we could generically amplify key-selective security in a black-box manner to adaptive security. The only tool needed for such an amplification is a relaxed notion of non-committing encryption (NCE) [CFGN96, DN00, KO04] which we call *weak* non-committing encryption (wNCE). In a wNCE system, there is an efficient simulator that could "open" the ciphertext to any message by providing a simulated secret key after already committing to the public key in the beginning. For security, it is only required that the distribution of simulated keys and ciphertext is computationally indistinguishable from the distribution generated by the real encryption protocol.[7]

Given such a weak NCE scheme, the idea is pretty straightforward. During setup, we would additionally sample a wNCE key pair, and encryption algorithm will be a simple double encryption where each (key-selective secure) ABE ciphertext will be encrypted under the wNCE system. Each predicate key now contains the wNCE secret key as well as the underlying ABE key, where during decryption, the decryptor first decrypts the outer wNCE ciphertext to learn the core ABE ciphertext which it then decrypts using the core ABE key. Now the adaptive security of this transformed scheme follows directly from wNCE simulation security and the key-selective ABE security. The idea there is that the challenge ciphertext will be computed as a simulated wNCE ciphertext instead, and when the adversary makes the post-challenge key query, then the reduction algorithm *opens* the wNCE ciphertext to the challenge ciphertext provided by the key-selective security ABE challenger, and answers the adversary's key query with a simulated wNCE secret key along with the core ABE key provided by the ABE challenger. Similar ideas were also used in [GVW12] in the context of simulation secure functional encryption.

Since there is a very simple construction for a weak NCE scheme from regular public key encryption, this seems to suggest that any 1-query key-selective secure ABE scheme could be generically lifted to achieve 1-query adaptive security instead, however there is an important caveat. The caveat is that this weak NCE construction from PKE has public-secret keys whose sizes grow linearly with length of the messages. Recall that in our generical transformation we encrypt the key-selective ABE ciphertext using the wNCE scheme. If the size of key-selective ABE ciphertext is fixed at setup time, then the transformation goes through as is, but this is not true for ABE in uniform models of computation where the whole motivation is being able to encrypt messages under unrestricted length attributes, thus the ciphertext sizes are a-priori unbounded. This implies that for the above transformation to work in the case of ABE for TMs we need a succinct weak NCE, where by succinct we mean that the system supports encryption of unbounded length messages. To this end, we show another generic transformation that takes any non-succinct weak NCE scheme and compiles it into a succinct NCE scheme albeit in the Random Oracel Model (ROM) [BR93]. Very briefly, the idea here is use the ROM as an *adaptive programmable* PRF to indistinguishably open simulated ciphertexts to arbitrary messages. During encryption, an encryptor chooses a random λ-bit string K which it encrypts under the non-succinct NCE scheme, and then encrypts the unbounded length message block-by-block using K as a secret key and ROM as a PRF. The simulatability of this scheme follows directly from the simulatability of the non-succinct scheme and programmability of the ROM. This is discussed in detail later in Appendix A. We want to point out that building a succinct weak NCE scheme as described above is impossible in the standard model [Nie02], thus adaptive security of our construction crucially relies on the usage of ROM.

Combining the above ideas, we obtain a 1-query adaptively secure ABE scheme for TMs. To conclude, we show that by using standard combinatorial techniques, the security could be improved to q-query adaptive security for any a-priori fixed polynomial $q(\cdot)$. Since we are dealing with just an ABE scheme, thus this transformation is much simpler than for other related transformations such as the one for functional encryption in [GVW12]. For completeness, we provide it in Appendix B.

[7] In regular notions of non-committing encryption, the simulator must also be able to indistinguishably explain the ciphertexts by providing encryption randomness too. We do not require that, thus regard our notion as a weak NCE system.

Related work, other suggested approaches, and future directions

Comparison with Agrawal-Singh [AS17]. Closest to us is the work of Agrawal-Singh [AS17] who construct a 1-query functional encryption scheme for Turing Machines where, like our ABE system, the encryption algorithm depends on the worst case running time of the TM. Ours and their construction share the same mechnical perspective of traversing through a sequence of garbled circuits for encrypting unbounded length inputs, however differ in overall execution since they rely on a succinct single-key FE scheme with the TM evaluation happening under the FE hood, whereas we work with more general primitives such as IBE and garbled RAM thereby our TM evaluation happens on encrypted pieces that come out as outputs of garbled RAM evaluations. The usage of a succinct single-key FE scheme has the benefit of the resulting encryption scheme being a FE scheme with short keys (and not just an ABE scheme like ours), but given the current state-of-the-art [GKP$^+$13] it also means relying on the LWE assumption, while we rely on much weaker primitives thus are not tethered to the LWE assumption. Like our core key-selective secure ABE scheme, they also prove security in the weaker model where there is a single-key query which must be made before receiving the challenge ciphertext. Although they do not provide any follow-up transformations to improve security like us, we believe our non-committing encryption idea could also be used with their FE construction. However, extending to q-query bounded collusion security would be more tricky than our case, but might be possible to adapt a more elaborate transformation along the lines of [GVW12].

ABE via laconic OT. Cho et al. [CDG$^+$17] introduced the concept of laconic transfer for secure computation over large inputs. They described an application of laconic OT to non-interactively compute in the RAM setting. Although this application does not directly lead to ABE schemes that supports (RAM) Turing Machine computation, it might be possible to repurpose the underlying ideas to build ABE by going through laconic OT along with garbling techniques. One would need to be careful in executing this idea so that the description size of the Turing Machine does not need to be a-priori bounded at setup time, and this might require adjusting the definition of the corresponding primitives. Additionally, such an approach would need one to rely on laconic OT whereas we chose to focus on IBE since it is both supported by multiple number theoretic assumptions as well as there are multiple number theoretic IBE constructions that do not themselves invoke garbling and thus avoid a double layer of garbling in the eventual construction. There have been prior works [DG17b, KNTY19] which observe that laconic OT could be replaced by IBE in certain applications, and it would be interesting to look at whether same could be done for this alternate approach. In our work, we provide a much direct construction directly from any regular IBE scheme.

Future directions. In this work we focus on proving standard semantic security of our ABE scheme, but we believe one could extend it to CCA security by either relying on the ROM, or on other generic transformations such as [KW19a], and prove it to be a 1-sided predicate encryption scheme directly without relying on generic transformations [GKW17, WZ17]. An interesting open question is whether one could extend our current approach to either achieve succinctness similar to [AS17], or extend it to FE without relying on stronger assumptions. Another related question is whether we could avoid the ROM for amplifying the security of our core ABE scheme from key-selective to fully adaptive. It might be useful look at the graph pebbling techniques [FKPR14, FJP15, JW16, JKK$^+$17, KW19b] to develop a more intricate hybrid structure for proving adaptive security directly. Another interesting thought might be to rely on adaptive security of garbled RAM schemes [GOS18] instead, however it is unclear how to leverage having an adaptive garbled RAM in our setting. Briefly, the reason is that the extra adaptivity it provides is useful in cryptosystems where an attacker is able to see some of the garbled RAM programs and then somehow influence the inputs or programs for the rest of them; while in our case all the garbling program calls are bundled together in a single call to the encryption oracle. Lastly, another important question is whether these techniques could be used to build ABE systems for TMs where the encryption algorithm no longer depends on the worst case running of the TM.

2 Preliminaries

Notations For any distribution D, we write $x \leftarrow D$ to denote that x is sampled from D. For any set S, we write $y \leftarrow S$ to denote that y is sampled from the uniform distribution on the S. For any randomized algorithm $\mathcal{A}(\cdot)$, we write $y \leftarrow \mathcal{A}(x)$ to denote that y is chosen as the output of $\mathcal{A}$ when run on input x. For positive integers m, n satisfying $m \leq n$, we write $[m, n]$ to denote the set $\{m, m+1, m+2, \ldots, n\}$. We write, $[n]$ as shorthand to denote the set $[1, n]$. For any set S, we write $|S|$ to denote its cardinality. For any string x, we write $|x|$ to denote its length. For any string x (or other ordered set), we write $x[i]$ to denote the i^{th} element of x. We use λ throughout for the security parameter in all algorithms.

Turing Machine Formalism A Turing Machine T is represented as a tuple

$$T = (Q, \Sigma, q_{\mathsf{start}}, F, \delta)$$

where, Q is a finite set of states, Σ is a finite tape alphabet, $q_{\mathsf{start}} \in Q$ is a unique starting state, $F \subset Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$ is a transition relation[8].

Without loss of generality, we will restrict to Turing Machines operating on a single tape, with state set $Q = \{0, 1\}^n$ for some n, and $\Sigma = \{0, 1\}$. At the start of the computation, we say the machine is in state q_{start}. The input $w \in \Sigma^*$ is written on the tape, and the machine head is scanning the leftmost symbol of w. All remaining cells are blank. Computation then proceeds in discrete time steps where at each step, the machine is initially in some state $q^{\mathsf{in}} \in Q$ and is scanning a cell containing bit $b^{\mathsf{in}} \in \Sigma$. It proceeds by applying the transition relation. If $\delta(q^{\mathsf{in}}, b^{\mathsf{in}})$ is defined and

$$\delta(q^{\mathsf{in}}, b^{\mathsf{in}}) = (q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir})$$

the machine updates its state to q^{out}, overwrites the cell it is currently scanning with b^{out}, and then moves the head by one cell in the direction dir. We call one such application, a *transition*. The machine proceeds in this way until it either enters an accepting state $q \in F$ in which case we say T *accepts* w, or it enters a state $q \in Q \setminus F$ and is scanning a bit b for which $\delta(q, b)$ is undefined. For a positive integer t, if T accepts w after performing at most t transitions, we say T accepts w *within* t *steps*. We write $|T|$ to denote the size of T and in particular the number of transitions defined under δ i.e. $|T| = |\{(q, b) : \delta(q, b) \text{ is defined}\}|$.

RAM Program Formalism We describe the notion of RAM Programs following the notions of [GHRW14a]. A RAM program P takes a *small* input x, has random access to a *large* memory database D upon which it can perform arbitrary reads and writes, performs a sequence of computational steps during a time interval $[t_{\mathsf{init}}, t_{\mathsf{fin}}]$, and at the end of its computation produces an output y. More concretely, the program P is represented as a sequence of t *step circuits* which we denote by C^P_{RAM}. Each of the step circuits performs one of the computational steps of P. Each circuit takes as input a *small* state state as well as the most recently read bit b^{read} from the database D. It outputs an updated state state' along with the next read and write instructions to be executed. More formally we write the circuit C^P_{RAM} computing the function:

$$C^P_{\mathsf{RAM}}(\mathsf{state}, b^{\mathsf{read}}) = (\mathsf{state}', i^{\mathsf{read}}, i^{\mathsf{write}}, b^{\mathsf{write}})$$

where $i^{\mathsf{read}}, i^{\mathsf{write}}, b^{\mathsf{write}}$ are the next location in D to read from, the next location in D to write to, and the next bit to write respectively. Throughout, we will use the notation $C^P_{\mathsf{RAM}^j}$ for the step circuit which represents the computational step performed by program P at time step j.

Computation of program P with input x on database D begins with the initial step circuit $C^P_{\mathsf{RAM}^{t_{\mathsf{init}}}}$ being evaluated with x as the state input and an arbitrary read input b^{read} which by convention we assume the initial circuit ignores. The circuit outputs $\mathsf{state}', i^{\mathsf{read}}, i^{\mathsf{write}}, b^{\mathsf{write}}$. The database is overwritten at location i^{write} with bit b^{write}, and then the next read bit $\tilde{b}^{\mathsf{read}}$ is read from location i^{read}. Then, the subsequent step circuit $C^P_{\mathsf{RAM}^{t_{\mathsf{init}}+1}}$ is evaluated with inputs $(\mathsf{state}', \tilde{b}^{\mathsf{read}})$. Computation proceeds in this way, with the updated state output by $C^P_{\mathsf{RAM}^j}$ being fed as the state input to $C^P_{\mathsf{RAM}^{j+1}}$, the write location output by $C^P_{\mathsf{RAM}^j}$ being

[8] Here we write $\{L, R\}$ to denote the set of directions *left* and *right*.

overwritten with the write bit output by $C^P_{\mathsf{RAM}^j}$, and the bit read at the read location output by $C^P_{\mathsf{RAM}^j}$ being fed as the read bit input to $C^P_{\mathsf{RAM}^{j+1}}$. Finally, the updated state output by the final step circuit $C^P_{\mathsf{RAM}^{t_{\mathsf{fin}}}}$ is taken to be the output y of the computation. By convention, we assume that the final step circuit does not produce any further writes. We also assume without loss of generality that all step circuits of P are of equal size, take the same size input, and produce the same size output.

We allow for multiple programs P_k to be run on database D in sequence, with writes made by each program persisting for the subsequent program. For each program P_k, we define its memory access pattern access_k to be the sequence of reads and writes P_k makes to D throughout its computation. More formally we write

$$\mathsf{access}_k = \{(i_j^{\mathsf{read}}, i_j^{\mathsf{write}}, b_j^{\mathsf{write}})\}_{j \in [t_{\mathsf{init},k}, t_{\mathsf{fin},k}]}$$

where $i_j^{\mathsf{read}}, i_j^{\mathsf{write}}, b_j^{\mathsf{write}}$ are respectively the read location, write location, and write bit output by the step circuit $C^{P_k}_{\mathsf{RAM}^j}$ for time j.

We use the notation $y \leftarrow P(x)^D$ to denote that program P operating on database D with input x outputs y. For multiple programs and inputs, we write $(y_1, \ldots, y_\ell) \leftarrow (P_1(x_1), \ldots, P_\ell(x_\ell))^D$ to denote that for all $k \in \ell$, y_k is the result of running P_k with input x_k on database D after running programs $P_{k'}$ on input $x_{k'}$ in sequence on D for all $k' < k$ with changes made to D persisting across program executions.

2.1 Attribute-Based Encryption for Turing Machines

An Attribute-Based Encryption (ABE) scheme ABE for set of attribute space $\{0,1\}^*$, Turing Machines classes $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda \in \mathbb{N}}$, and message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ. It outputs the public parameters pp and the master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, T) \to \mathsf{sk}_T$. The key generation algorithm takes as input the master secret key msk and a Turing Machine $T \in \mathcal{T}_\lambda$. It outputs a secret key sk_T.

$\mathsf{Enc}(\mathsf{pp}, m, (w, t)) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters pp, a message $m \in \mathcal{M}_\lambda$, and a pair (w, t) consisting of an attribute $w \in \{0,1\}^*$, and a positive integer time bound t. It outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) \to m/\bot$. The decryption algorithm takes as input a secret key sk_T and a ciphertext ct. It outputs either a message $m \in \mathcal{M}_\lambda$ or a special symbol $\bot$.

Correctness. We say an ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies correctness if for all $\lambda \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $T \in \mathcal{T}_\lambda$, $m \in \mathcal{M}_\lambda$, $\mathsf{sk}_T \leftarrow \mathsf{KeyGen}(\mathsf{msk}, T)$, $t \in \mathbb{N}$, and $w \in \{0,1\}^*$ for which T accepts w within t steps, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m, (w, t))$ we have that $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) = m$.

Efficiency. We require that the algorithm $\mathsf{Setup}(1^\lambda)$ runs in time polynomial in the security parameter λ. We require that the algorithm $\mathsf{KeyGen}(\mathsf{msk}, T)$ runs in time polynomial in the security parameter λ and the size $|T|$ of T. We require that the algorithm $\mathsf{Enc}(\mathsf{pp}, m, (w, t))$ runs in time polynomial in the security parameter λ, the length $|m|$ of the message m, the length $|w|$ of the attribute w, and the time bound t. We require that the algorithm $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct})$ runs in time polynomial in the security parameter λ and the size $|\mathsf{ct}|$ of the ciphertext ct.

Security. Next, we define the security notions we consider for ABE systems.

Definition 2.1 (adaptive security)**.** We say an ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is fully secure if for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$ the following holds

$$\Pr\left[\mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{st}, \mathsf{ct}) \;= \beta : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda); \quad \beta \leftarrow \{0,1\} \\ (\mathsf{st}, m_0, m_1, (w, 1^t)) \leftarrow \mathcal{A}_0^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pp}, 1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_\beta, (w, t)) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where all Turing Machines T queried by $\mathcal{A}$ do not accept the word w within t steps.

In this work, we focus on bounded collusion security for ABE systems where the adversary is restricted to make an a-priori bounded number of key generation queries, say at most Q queries (for some polynomially bounded function $Q(\lambda)$). The definition is given below.

Definition 2.2 (Q-query adaptive security)**.** An ABE scheme is said to be Q-query adaptively secure if in the above security game (see Definition 2.1), the adversary can make at most Q queries to the key generation oracle.

We also define the weaker notion which we call key-selective security, where the adversary to must make all key queries before it is given the challenge ciphertext. The definition is given below.

Definition 2.3 (Q-query key-selective security)**.** An ABE scheme is said to be Q-query key-selective secure if in the above security game (see Definition 2.1), the adversary can make at most Q queries to the key generation oracle, and all key queries must be made before getting the challenge ciphertext.

2.2 Secret Key Encryption

A secret key encryption system SKE for message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of three polynomial time randomized algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{K}$. The setup algorithm takes as input the security parameter λ. It outputs the secret key K.

$\mathsf{Enc}(\mathsf{K}, m) \rightarrow \mathsf{ct}$. The encryption algorithm takes as input a secret key K and a message $m \in \mathcal{M}_\lambda$. It outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{K}, \mathsf{ct}) \rightarrow m$. The decryption algorithm takes as input a secret key K and a ciphertext ct. It outputs a message $m \in \mathcal{M}_\lambda$

Correctness. We say a secret key encryption scheme $\mathsf{SKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ satisfies correctness if for all $\lambda \in \mathbb{N}$, $\mathsf{K} \leftarrow \mathsf{Setup}(1^\lambda)$, $m \in \mathcal{M}_\lambda$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{K}, m)$, we have that $\mathsf{Dec}(\mathsf{K}, \mathsf{ct}) = m$.

Definition 2.4. We say a secret key encryption scheme $\mathsf{SKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ is secure if for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}_1^{\mathsf{Enc}(\mathsf{K},\cdot)}(\mathsf{st}, \mathsf{ct}) \;= \beta : \begin{array}{c} \mathsf{K} \leftarrow \mathsf{Setup}(1^\lambda); \quad \beta \leftarrow \{0,1\} \\ (\mathsf{st}, m_0, m_1) \leftarrow \mathcal{A}_0^{\mathsf{Enc}(\mathsf{K},\cdot)}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{K}, m_\beta) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

2.3 Identity-Based Encryption

An Identity-Based Encryption (IBE) scheme IBE for set of identity spaces $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda \in \mathbb{N}}$ and message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{pp}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ. It outputs the public parameters pp and the master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. The key generation algorithm takes as input the master secret key msk and an identity $\mathsf{id} \in \mathcal{I}_\lambda$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pp}, m, \mathsf{id}) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters pp, a message $m \in \mathcal{M}_\lambda$, and an identity $\mathsf{id} \in \mathcal{I}_\lambda$. It outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) \to m/\bot$. The decryption algorithm takes as input a secret key $\mathsf{sk}_{\mathsf{id}}$ and a ciphertext ct. It outputs either a message $m \in \mathcal{M}_\lambda$ or a special symbol $\bot$.

Correctness. We say an IBE scheme $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies correctness if for all $\lambda \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{id} \in \mathcal{I}_\lambda$, $m \in \mathcal{M}_\lambda$, $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m, \mathsf{id})$, we have that $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) = m$.

Definition 2.5. We say an IBE scheme $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is secure if for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{st}, \mathsf{ct}) \;= \beta : \begin{array}{c} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda); \quad \beta \leftarrow \{0,1\} \\ (\mathsf{st}, m_0, m_1, \mathsf{id}) \leftarrow \mathcal{A}_0^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda, \mathsf{pp}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_\beta, \mathsf{id}) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where all id' queried by $\mathcal{A}$ satisfy $\mathsf{id} \neq \mathsf{id}'$.

In our main construction of ABE for Turing Machines, we will make use of the following modified security definition, where the challenge may consist of some polynomial number of message pairs.

Definition 2.6 (multi-challenge security)**.** We say an IBE scheme satisfies multi-challenge security if the adversary can not distinguish with non-negligible probability even when it receives polynomially many challenge ciphertexts on polynomially many message-identity tuples of the form $\{(m_0^{(i)}, m_1^{(i)}, \mathsf{id}_i)\}_{i \in [k]}$.

It follows by a simple hybrid argument that any scheme IBE which is fully secure for a single challenge message pair, also satisfies multi-challenge security described above.

3 Garbled RAM with Iterated Simulation Security

In this section we define a notion of Garbled RAM security which abstracts out properties of Garbled RAM constructions in previous works which we will use in our construction of ABE. At a high level, our security notion, which we call Iterated Simulation Security requires that there exists an efficient simulator such that for any sequence of ℓ programs and inputs, it is hard to distinguish simulations of the first k programs and inputs along with honest garblings of the remaining $\ell - k$ programs from simulations of the first $k+1$ programs and inputs along with honest garblings of the remaining $\ell - k - 1$ programs. Our security definition will actually be a notion of security with Unprotected Memory Access (UMA), that is security in which the garbling may leak the contents of the garbled database D, and the memory access patterns access_j of the programs. We drop the label UMA in the subsequent to reduce clutter.

A Garbled RAM scheme GRAM consists of three polynomial time algorithms $(\mathsf{GData}, \mathsf{GProg}, \mathsf{Eval})$ with the following syntax:

$\mathsf{GData}(1^\lambda, D) \to (\widetilde{D}, \mathsf{k}_D)$. The data garbling algorithm takes as input the security parameter λ and a database $D \in \{0,1\}^m$. It outputs a garbled database $\widetilde{D}$ and program garbling key k_D.

$\mathsf{GProg}(1^\lambda, \mathsf{k}_D, m, P, 1^n, (t_{\mathsf{init}}, t_{\mathsf{fin}})) \to (\widetilde{P}, \{\mathsf{lab}^{\mathsf{in}}_{i,b}\}_{i \in [n], b \in \{0,1\}})$. The program garbling algorithm takes as input the security parameter λ, a program garbling key k_D, a database size m, a program P which operates on a database of size m and takes an input of length n, and time range given as a pair of an initial time t_{init} and final time t_{fin}. It outputs a garbled program $\widetilde{P}$ and a collection of input labels $\{\mathsf{lab}^{\mathsf{in}}_{i,b}\}_{i \in [n], b \in \{0,1\}}$.

$\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}, \{\mathsf{lab}_i^{\mathsf{in}}\}_{i\in[n]}) \to \widetilde{y}$. The evaluation algorithm takes as input a garbled database $\widetilde{D}$, a garbled program $\widetilde{P}$, and a collection of n labels $\{\mathsf{lab}_i^{\mathsf{in}}\}_{i\in[n]}$. It outputs a value $\widetilde{y}$. As in [GHRW14a], we will think of the evaluation algorithm as a RAM program operating on database $\widetilde{D}$ which is able to perform arbitrary reads and writes on $\widetilde{D}$. We slightly abuse notation, and will write

$$\mathsf{Eval}^{\widetilde{D}}((\widetilde{P_1}, \{\mathsf{lab}_i^{\mathsf{in},1}\}_{i\in[n_1]}), ..., (\widetilde{P_\ell}, \{\mathsf{lab}_i^{\mathsf{in},\ell}\}_{i\in[n_\ell]})) \to (\widetilde{y_1}, \dots, \widetilde{y_\ell})$$

to denote that for all $j \in \ell$, $\widetilde{y_j}$ is the result of the evaluation algorithm on garbled program $\widetilde{P_j}$ with labels $\{\mathsf{lab}_i^{\mathsf{in},j}\}_{i\in[n_j]}$ on garbled database $\widetilde{D}$ after running the evaluation algorithm on garbled programs $\widetilde{P_{j'}}$ with labels $\{\mathsf{lab}_i^{\mathsf{in},j'}\}_{i\in[n_{j'}]}$ in sequence on $\widetilde{D}$ for all $j' < j$ with changes made to $\widetilde{D}$ persisting across evaluations.

Correctness. Fix parameters $\lambda, \ell, m \in \mathbb{N}$, a database $D \in \{0,1\}^m$, and programs and inputs $\{(P_j, x_j \in \{0,1\}^{n_j}, n_j, t_{\mathsf{init},j}, t_{\mathsf{fin},j})\}_{j\in[\ell]}$. Let

$$(y_1, \dots, y_\ell) \leftarrow (P_1(x_1), \dots, P_\ell(x_\ell))^D$$

be the result of sequentially running the programs P_j on inputs x_j operating on persistent database D. We say a garbled RAM scheme $\mathsf{GRAM} = (\mathsf{GData}, \mathsf{GProg}, \mathsf{Eval})$ satisfies correctness, if for all $j \in [\ell]$ the following holds

$$\Pr\left[\widetilde{y}_j = y_j : \begin{array}{c} (\widetilde{D}, \mathsf{k}_D) \leftarrow \mathsf{GData}(1^\lambda, D) \\ \forall j \in [\ell],\ (\widetilde{P}_j, \{\mathsf{lab}_{i,b}^{\mathsf{in},j}\}_{i\in[n_j],b\in\{0,1\}}) \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, m, P_j, 1^{n_j}, (t_{\mathsf{init},j}, t_{\mathsf{fin},j})) \\ (\widetilde{y}_1, \dots, \widetilde{y}_\ell) \leftarrow \mathsf{Eval}^{\widetilde{D}}((\widetilde{P}_1, \{\mathsf{lab}_{i,x_j[i]}^{\mathsf{in},j}\}_{i\in[n_j]}), \dots, (\widetilde{P}_1, \{\mathsf{lab}_{i,x_j[i]}^{\mathsf{in},j}\}_{i\in[n_j]})) \end{array}\right] = 1.$$

Definition 3.1 (iterated simulation security)**.** We say a garbled RAM scheme $\mathsf{GRAM} = (\mathsf{GData}, \mathsf{GProg}, \mathsf{Eval})$ satisfies iterated simulation security if there exists a polynomial time simulator Sim such that for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\text{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}_1(\mathsf{st}, \mathsf{chal}) \ = \beta : \begin{array}{c} \beta \leftarrow \{0,1\} \\ (\mathsf{st}, k, D, \{(P_j, x_j, n_j, (t_{\mathsf{init},j}, t_{\mathsf{fin},j}))\}_{j\in[\ell]}) \leftarrow \mathcal{A}_0(1^\lambda) \\ \mathsf{chal} \leftarrow \mathsf{Exp}_{k-\beta}^\lambda(D, \{(P_j, x_j, n_j, (t_{\mathsf{init},j}, t_{\mathsf{fin},j}))\}_{j\in[\ell]}) \end{array}\right] \le \frac{1}{2} + \text{negl}(\lambda),$$

where for $0 \le k \le \ell$, the output of $\mathsf{Exp}_k^\lambda(D, \{(P_j, x_j, n_j, (t_{\mathsf{init},j}, t_{\mathsf{fin},j}))\}_{j\in[\ell]})$ is defined

$$\left\{ \begin{pmatrix} \widetilde{D}, \{(\widetilde{P}_j, \{\mathsf{lab}_i^{\mathsf{in},j}\}_i)\}_{j\in[k]}, \\ \{(\widetilde{P}_j, \{\mathsf{lab}_{i,b}^{\mathsf{in},j}\}_{i,b})\}_{j\in[k+1,\ell]} \end{pmatrix} : \begin{array}{c} (\widetilde{D}, \mathsf{k}_D) \leftarrow \mathsf{GData}(1^\lambda, D) \\ \forall j > k,\ (\widetilde{P}_j, \{\mathsf{lab}_{i,b}^{\mathsf{in},j}\}_{i\in[n_j],b\in\{0,1\}}) \\ \qquad \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, |D|, P_j, 1^{n_j}, (t_{\mathsf{init},j}, t_{\mathsf{fin},j})) \\ \forall j \le k,\ (\widetilde{P}_j, \{\mathsf{lab}_i^{\mathsf{in},j}\}_{i\in[n_j],b\in\{0,1\}}) \\ \qquad \leftarrow \mathsf{Sim}(\mathsf{k}_D, 1^{n_j}, |P_j|, y_j, \mathsf{access}_j, (t_{\mathsf{init},j}, t_{\mathsf{fin},j})) \end{array} \right\}$$

where for all j, $|P_j|$ is the size of the program P_j, y_j is the result of running P_j with input x_j on the database D after having run the previous $j-1$ programs and inputs, and access_j is the memory access pattern of P_j. Note, that all the inputs to $\mathsf{Sim}(\cdot)$, can be computed from the inputs to Exp_k^λ.

Remark 3.2. As a point of comparison with the simulation security notions considered in prior works such as [GHRW14a], we would want to highlight that in prior works the simulator always outputs a fully simulated execution of the garbled RAM which must be indistinguishable from honestly garbled programs. We, on the other hand, consider indistinguishability in between these partial execution steps.

Efficiency. We require that the algorithm $\mathsf{GData}(1^\lambda, D)$ runs in time polynomial in the security parameter λ and the size $|D|$ of the database D. We require that the algorithms $\mathsf{GProg}(1^\lambda, \mathsf{k}_D, m, P, 1^n, (t_{\mathsf{init}}, t_{\mathsf{fin}}))$ and $\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}, \{\mathsf{lab}_i^{\mathsf{in}}\}_{i\in[n]})$ both run in time polynomial in the security parameter λ, $\log(m)$ where m is the size of D, the size $|P|$ of P, the size n of the input taken by P, and the total number of steps $(t_{\mathsf{fin}} - t_{\mathsf{init}})$ taken by P.

4 ABE for Turing Machines

In this section we give our main construction of an ABE scheme for Turing Machines. The scheme will be for message spaces $\mathcal{M} = \{\{0,1\}^\lambda\}_{\lambda \in \mathbb{N}}$.

The primitives used by our construction are as follows. Let $\mathsf{GRAM} = (\mathsf{GData},\ \mathsf{GProg},\ \mathsf{Eval})$ be a garbled RAM scheme satisfying Iterated Simulation Security. In addition, let IBE be a secure IBE scheme which can encrypt messages of length λ, and assume there is some polynomial $n(\cdot)$ for which the identity space of IBE includes identities of length $n'(\lambda) := n(\lambda) + \lceil \log(n+2) \rceil + 2$. In the subsequent discussion we will simply write n as shorthand for $n(\lambda)$. Our scheme additionally uses a secret key encryption scheme $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$. For simplicity of exposition, we assume (w.l.o.g.) that the IBE encryption algorithm takes as input λ-bits of randomness.

In our scheme, we will allow secret key queries for deterministic Turing Machines $T = (Q, \Sigma, q_{\mathsf{start}}, F, \delta)$ with the following restrictions. We assume:

- All machines have state space $Q \subset \{0,1\}^n$.
- The alphabet Σ is binary. That is $\Sigma = \{0,1\}$.
- The all 0 state 0^n is reserved as the unique start state q_{start} for all machines.
- The all 1 state 1^n is reserved as the unique accept state $q_{\mathsf{accept}} \in F$.
- The transition relation δ is a *partial* function. In particular, all machines are deterministic.

The above assumptions are essentially without loss of generality for deterministic Turing Machines. In particular, any deterministic Turing Machine with constant size alphabet and a polynomial number of states can be transformed in to a machine satisfying these assumptions with at most polynomial blowup. We will identify each machine T with the set of possible transitions it can make under δ:

$$T = \{(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir}) \ : \ \delta(q^{\mathsf{in}}, b^{\mathsf{in}}) = (q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir})\}$$

Thus, the notation $|T|$, will simply be the cardinality of the right hand side of the above.

The secret keys of our scheme will be carefully chosen sets of identity secret keys from the IBE scheme IBE. In particular, each identity secret key will be for an identity $\mathsf{id} \in \{0,1\}^{n + \lceil \log(n) \rceil + 2}$.

4.1 Construction

We now formally describe the construction of our ABE scheme, $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$, and outputs $(\mathsf{pp}, \mathsf{msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, T) \to \mathsf{sk}_T$. Let the Turing machine T be given as the set of possible transitions it can make under its transition relation δ:

$$T = \{(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir}) \ : \ \delta(q^{\mathsf{in}}, b^{\mathsf{in}}) = (q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir})\}$$

For each transition $(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir}) \in T$ the key generation algorithm samples $n+2$ identity secret keys. Let $\mathcal{ID}_T$ be the set of $(n+2) \cdot |T|$ identities described below:

$$\mathcal{ID}_T = \left\{ (q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta) \in \{0,1\}^{n'} \ : \ (q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir}) \in T \wedge \begin{pmatrix} (i \in [n] \wedge \beta = q^{\mathsf{out}}[i]) \vee \\ (i = n+1 \wedge \beta = b^{\mathsf{out}}) \vee \\ (i = n+2 \wedge \beta = b^{\mathsf{dir}}) \end{pmatrix} \right\} \quad (1)$$

where in the above $b^{\mathsf{dir}} = 0$ if $\mathsf{dir} = L$ and $b^{\mathsf{dir}} = 1$ if $\mathsf{dir} = R$.

Next, the key generation algorithm samples an IBE secret key for each identity in $\mathcal{ID}_T$. Concretely, it chooses

$$\forall\ (q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta) \in \mathcal{ID}_T, \quad \mathsf{sk}_{(q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta)} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{msk}, (q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta)).$$

Finally, the key generation algorithm sets the key to be the machine description T and the entire set of identity secret keys it chose:

$$\mathsf{sk}_T = \left(T, \left\{\mathsf{sk}_{(q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta)}\right\}_{(q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta) \in \mathcal{ID}_T}\right).$$

$\mathsf{Enc}(\mathsf{pp}, m, (w, t)) \to \mathsf{ct}$. The encryption algorithm garbles a database D along with several copies of a *step program* P. We formally describe the RAM program P in Fig. 1 before moving on to the encryption algorithm.

The step program $P[\mathsf{pp}, \{\mathsf{K}_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}, m, j; \{r_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}](q^{\mathsf{in}}, b^{\mathsf{out}}, b^{\mathsf{dir}})$

The program P operates on a database D of size $t + 1 + \lceil\log(t + 1)\rceil$. For convenience, we will think of the database as a length $t + 1$ array D' concatenated with an integer index $\mathsf{idx} \in [t + 1]$ i.e $D := D'||\mathsf{idx}$. The program P has hard-coded the public parameters pp of an instance of IBE, a set of SKE secret keys $\{\mathsf{K}_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}$, a message m, an integer $j \in [t + 1]$, and a set of randomness strings $\{r_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}$. It takes as input an n-bit state q, a bit b^{out}, and a bit b^{dir}.

1. If $j > 1$, the program reads the index idx, and then it overwrites the idx-th bit of D' with b^{out} i.e. it sets $D'[\mathsf{idx}] := b^{\mathsf{out}}$. Otherwise if $j = 1$, the program ignores the input b^{out}.

2. If $j > 1$, $\mathsf{idx} > 1$, and $b^{\mathsf{dir}} = 0$, the program overwrites idx with $\mathsf{idx} - 1$. Else if $j > 1$, $\mathsf{idx} = 1$, and $b^{\mathsf{dir}} = 0$, for each $i \in [n + 2]$ and $b \in \{0, 1\}$, the program re-sets $\mathsf{K}_{i,b} := \mathbf{0}$. (This instruction is to prevent decryption if the tape head tries to move left off of the tape.) Else, if $j > 1$ and $b^{\mathsf{dir}} = 1$, the program overwrites idx with $\mathsf{idx} + 1$. Else, if $j = 1$, the program ignores the input b^{dir}.

3. The program reads the bit $b^{\mathsf{in}} := D'[\mathsf{idx}]$ at the updated idx. For each pair $(i, b) \in [n + 2] \times \{0, 1\}$, the program computes

$$\mathsf{ct}_{i,b} := \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}_{i,b}, (q^{\mathsf{in}}, b^{\mathsf{in}}, i, b); r_{i,b}).$$

4. Finally, if $q^{\mathsf{in}} = \mathsf{accept}$, the program outputs $(\{\mathsf{ct}_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}, m)$. Otherwise it outputs $(\{\mathsf{ct}_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}, \perp)$.

Figure 1: The step program P.

The encryption algorithm proceeds as follows.

1. The encryption algorithm sets a database $D \in \{0, 1\}^{t+1+\lceil\log(t+1)\rceil}$. It sets the first $|w|$ bits of D to match w, and sets the remaining bits to 0. More formally,

$$D := w||0^{t+1+\lceil\log(t+1)\rceil-|w|}$$

   where $||$ denotes concatenation. The algorithm next garbles the database $(\widetilde{D}, \mathsf{k}_D) \leftarrow \mathsf{GData}(1^\lambda, D)$.

2. For each $(i, b, j) \in [n+2] \times \{0, 1\} \times [t+1]$, the algorithm samples randomness $r^{(j)}_{i,b}$ and SKE secret keys $\mathsf{K}^{(j)}_{i,b}$ as $r^{(j)}_{i,b} \leftarrow \{0, 1\}^\lambda, \mathsf{K}^{(j)}_{i,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

3. Let P be the RAM program described as described in Fig. 1. For each $j \in [t+1]$, the algorithm sets P_j as $P_j := P[\mathsf{pp}, \{\mathsf{K}_{i,b}^{(j)}\}_{i,b}, m, j; \{r_{i,b}^{(j)}\}_{i,b}]$.
4. Let ℓ be the number of steps P takes to run on a database of length $|D|$. For each $j \in [t+1]$, the algorithm garbles the program P_j, computing
$$(\widetilde{P_j}, \{\mathsf{lab}_{i,b}^{\mathsf{in},j}\}_{i,b}) \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, t+1+\lceil\log(t+1)\rceil, P_j, 1^{n+2}, (1+(j-1)\cdot\ell, j\cdot\ell)).$$
5. For each $(i,b,j) \in [n+2]\times\{0,1\}\times[t]$, the algorithm computes ciphertexts $\widetilde{\mathsf{ct}}_{i,b}^{(j)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{i,b}^{(j)}, \mathsf{lab}_{i,b}^{\mathsf{in},j+1})$.
6. Let $\{\mathsf{lab}_{i,b}^{\mathsf{in},1}\}_{i,b}$ be the set of input labels computed when garbling program P_1. Recall that the all zero state is the canonical start state. The algorithm outputs the ciphertext
$$\mathsf{ct} = (w, t, \widetilde{D}, \{\mathsf{lab}_{i,0}^{\mathsf{in},1}\}_{i\in[n+2]}, \{\widetilde{P}_j\}_{j\in[t+1]}, \{\widetilde{\mathsf{ct}}_{i,b}^{(j)}\}_{i\in[n+2],b\in\{0,1\},j\in[t]}).$$

$\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) \to m/\bot$. The decryption algorithm parses the ciphertext and secret key as
$$\mathsf{ct} = (w, t, \widetilde{D}, \{\mathsf{lab}_i^{\mathsf{in}}\}_{i\in[n+2]}, \{\widetilde{P}_j\}_{j\in[t+1]}, \{\widetilde{\mathsf{ct}}_{i,b}^{(j)}\}_{i\in[n+2],b\in\{0,1\},j\in[t]}), \quad \mathsf{sk}_T = \left(T, \left\{\mathsf{sk}_{(q^{\mathsf{in}},b^{\mathsf{in}},i,\beta)}\right\}_{(q^{\mathsf{in}},b^{\mathsf{in}},i,\beta)\in\mathcal{ID}_T}\right).$$

Let $t' \le t$ be the maximum number of well defined transitions the machine T can make on input w within t time steps. Let
$$\{(q_j^{\mathsf{in}}, b_j^{\mathsf{in}}, q_j^{\mathsf{out}}, b_j^{\mathsf{out}}, \mathsf{dir}_j)\}_{j\in[t']}$$
be the t' transitions T makes on input w. The decryption algorithm sets $\mathbf{lab}_1 := \{\mathsf{lab}_i^{\mathsf{in}}\}_{i\in[n+2]}$, and then proceeds to evaluate the garbled RAM programs in ascending order for $j=1$ to $j=t'+1$ as follows:

1. The decryption algorithm evaluates the jth garbled RAM program $\widetilde{P}_j$ on the current value of the garbled database $\widetilde{D}$ with the input given by labels in $\mathbf{lab}_j$:
$$(\{\mathsf{ct}_{i,b}^{(j)}\}_{i,b}, \widetilde{y_j}) \leftarrow \mathsf{Eval}^{\widetilde{D}}(\widetilde{P}_j, \mathbf{lab}_j).$$
Note that the garbled database $\widetilde{D}$ has now been updated after running $\mathsf{Eval}(\cdot)$.
2. If $\widetilde{y_j} \neq \bot$, the algorithm breaks and exits the loop, sets $m := \widetilde{y_j}$, and outputs m.
3. Otherwise, if $\widetilde{y_j} = \bot$ it continues. Let $(q_j^{\mathsf{in}}, b_j^{\mathsf{in}}, q_j^{\mathsf{out}}, b_j^{\mathsf{out}}, \mathsf{dir}_j)$ be the jth transition T makes on input w. Also, for $i \in [n+2]$, let $b_{i,j}$ denote the following bit
$$b_{i,j} := \begin{cases} q_j^{\mathsf{out}}[i] & \text{if } i \in [n] \\ b_j^{\mathsf{out}} & \text{if } i = n+1 \\ b_j^{\mathsf{dir}} & \text{otherwise.} \end{cases}$$
where in the above $b_j^{\mathsf{dir}} = 0$ if $\mathsf{dir}_j = L$ and $b_j^{\mathsf{dir}} = 1$ if $\mathsf{dir}_j = R$. The algorithm computes the labels for the next program as follows. For $i \in [n+2]$, it decrypts the IBE and SKE ciphertexts as:
$$\mathsf{K}_{i,b_{i,j}}^{(j)} = \mathsf{IBE.Dec}(\mathsf{sk}_{(q^{\mathsf{in}},b^{\mathsf{in}},i,b_{i,j})}, \mathsf{ct}_{i,b_{i,j}}^{(j)}), \qquad \mathsf{lab}_{i,b_{i,j}}^{\mathsf{in}} \leftarrow \mathsf{SKE.Dec}(\mathsf{K}_{i,b_{i,j}}^{(j)}, \widetilde{\mathsf{ct}}_{i,b_{i,j}}^{(j)}).$$
4. If $j < t'+1$, the algorithm sets the labels for the garbled program $\widetilde{P}_{j+1}$ as
$$\mathbf{lab}_{j+1} := \{\mathsf{lab}_{i,b_{i,j}}^{\mathsf{in}}\}_{i\in[n+2]}$$
and otherwise if $j = t'+1$, the algorithm exits the loop and returns $\bot$.

4.2 Correctness

We show that the above construction satisfies correctness. Fix message m, attribute w, time bound t, and Turing Machine T. Let $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_T \leftarrow \mathsf{KeyGen}(\mathsf{msk}, T)$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m, (w, t))$. Fix $t' \leq t$, and assume T accepts w, and transitions to being in state accept for the first time after t' transitions. Now, for $j \in [t']$, let

$$(q_j^{\mathsf{in}}, b_j^{\mathsf{in}}, q_j^{\mathsf{out}}, b_j^{\mathsf{out}}, \mathsf{dir}_j)$$

be the jth transition made by T on input w. In particular, note that $q_1^{\mathsf{in}} = \mathsf{start} = 0^n$ and $q_{t'}^{\mathsf{out}} = \mathsf{accept} = 1^n$. Next, for $j \in [t']$, let tape_j and pos_j respectively be the contents of the first $t+1$ cells of the Turing Machine tape and the position of the tape head before the jth transition. Parse the ciphertext ct as

$$\mathsf{ct} = (w, t, \widetilde{D}, \{\mathsf{lab}_i^{\mathsf{in}}\}_{i \in [n+2]}, \{\widetilde{P}_j\}_{j \in [t+1]}, \{\widetilde{\mathsf{ct}}_{i,b}^{(j)}\}_{i \in [n+2], b \in \{0,1\}, j \in [t]}).$$

We now show that $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct})$ outputs m. The decryption algorithm evaluates the garbled programs $\widetilde{P}_j$ in ascending order starting with $\widetilde{P}_1$. We claim that the loop satisfies the following invariant. For each $j \in [t']$, $q_j^{\mathsf{out}}, b_j^{\mathsf{out}}, b_j^{\mathsf{dir}}$ is the input encoded by $\mathbf{lab}_{j+1}$, and the garbled database $\widetilde{D}$ at the start of iteration $j+1$ of the loop has contents $\mathsf{tape}_j || \mathsf{pos}_j$. We establish this claim by induction on j.

First we establish the claim when $j = 1$. At the start of the first iteration of the loop, the garbled database $\widetilde{D}$ has contents $w || 0^{t+1+\lceil \log(t+1) \rceil - |w|}$. The label set $\mathbf{lab}_1$, which is used to evaluate $\widetilde{P}_1$, encodes the all 0 input. Now, the underlying program P_1 has hard-coded pp, the secret keys $\{\mathsf{K}_{i,b}^{(1)}\}$, and $j = 1$. It follows then by construction, that in the first step of the first iteration, $\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}_1, \mathbf{lab}_1)$ outputs $(\{\mathsf{ct}_b^i\}_{(i,b) \in [n+2] \times \{0,1\}}, \perp)$ where for each $i \in [n+2]$ and $b \in \{0,1\}$,

$$\mathsf{ct}_b^i \leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}_{i,b}^{(1)}, (\mathsf{start}, w_1, i, b))$$

By the definition of sk_T and the keys $\mathsf{K}_{i,b}^{(j)}$ as well as the correctness of IBE and SKE, for each $i \in [n]$ the algorithm decrypts $\mathsf{K}_{i,q_1^{\mathsf{out}}[i]}^{(1)}$ and $\mathsf{lab}_{i,q_1^{\mathsf{out}}[i]}^{\mathsf{in}}$, for $i = n+1$ the algorithm decrypts $\mathsf{K}_{i,b_1^{\mathsf{out}}[i]}^{(1)}$ and $\mathsf{lab}_{i,b_1^{\mathsf{out}}}^{\mathsf{in}}$, and for $i = n+2$ the algorithm decrypts $\mathsf{K}_{i,b_1^{\mathsf{dir}}}^{(1)}$ and $\mathsf{lab}_{i,b_1^{\mathsf{dir}}}^{\mathsf{in}}$. Thus $q_1^{\mathsf{out}}, b_1^{\mathsf{out}}, b_1^{\mathsf{dir}}$ is encoded by $\mathbf{lab}_2$. Since P_1 does not modify the database, the contents of the database at the start of the second iteration are $\mathsf{tape}_1 || \mathsf{pos}_1$. This establishes the claim for $j = 1$.

Now, assume the claim holds for some $j < t'$. Then at the start of iteration $j+1$ of the loop, the garbled database $\widetilde{D}$ has contents $\mathsf{tape}_j || \mathsf{pos}_j$. The label set $\mathbf{lab}_{j+1}$, which is used to evaluate $\widetilde{P}_{j+1}$, encodes the input $q_j^{\mathsf{out}}, b_j^{\mathsf{out}}, b_j^{\mathsf{dir}}$. Note, that $q_{j+1}^{\mathsf{in}} = q_j^{\mathsf{out}}$. Now, the underlying program P_{j+1} has hard-coded pp, the secret keys $\{\mathsf{K}_{i,b}^{(1)}\}$, and the integer $j+1$. It follows then by the definition of P_{j+1} and correctness of GRAM, that in the first step of the iteration, $\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}_{j+1}, \mathbf{lab}_{j+1})$ outputs $(\{\mathsf{ct}_b^i\}_{(i,b) \in [n+2] \times \{0,1\}}, \perp)$ where for each $i \in [n+2]$ and $b \in \{0,1\}$,

$$\mathsf{ct}_b^i \leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}_{i,b}^{(j+1)}, (q_{j+1}^{\mathsf{in}}, b_{j+1}^{\mathsf{in}}, i, b))$$

By the definition of sk_T and the keys $\mathsf{K}_{i,b}^{(j)}$ as well as the correctness of IBE and SKE, for each $i \in [n]$ the algorithm decrypts $\mathsf{K}_{i,q_1^{\mathsf{out}}[i]}^{(j+1)}$ and $\mathsf{lab}_{i,q_{j+1}^{\mathsf{out}}[i]}^{\mathsf{in}}$, for $i = n+1$ the algorithm decrypts $\mathsf{K}_{i,b_{j+1}^{\mathsf{out}}[i]}^{(1)}$ and $\mathsf{lab}_{i,b_{j+1}^{\mathsf{out}}}^{\mathsf{in}}$, and for $i = n+2$ the algorithm decrypts $\mathsf{K}_{i,b_{j+1}^{\mathsf{dir}}}^{(j+1)}$ and $\mathsf{lab}_{i,b_{j+1}^{\mathsf{dir}}}^{\mathsf{in}}$. Thus $q_{j+1}^{\mathsf{out}}, b_{j+1}^{\mathsf{out}}, b_{j+1}^{\mathsf{dir}}$ is encoded by $\mathbf{lab}_{j+2}$. Also, by the definition of P_{j+1} and correctness of GRAM, after the first step of iteration $j+1$, $\widetilde{D}$ has been modified so that its contents are $\mathsf{tape}_{j+1} || \mathsf{pos}_{j+1}$. This established the claim for $j+1$.

By induction, the claim holds for all $j \in [t']$. In particular, the label set $\mathbf{lab}_{j+1}$ encodes the input $(q_{t'+1}^{\mathsf{out}}, b_{t'+1}^{\mathsf{out}}, b_{t'+1}^{\mathsf{dir}})$. As we assumed, $q_{t'+1}^{\mathsf{out}} = \mathsf{accept}$. By correctness of GRAM, in iteration $t'+1$, $\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}_{t'+1}, \mathbf{lab}_{t'+1})$ outputs $(\{\mathsf{ct}_b^i\}_{(i,b) \in [n+2] \times \{0,1\}}, m)$. It follows then that the decryption algorithm outputs m, and we are done.

4.3 Efficiency

We discuss the efficiency of the algorithms of the above construction. Since $\mathsf{Setup}(\lambda)$ simply runs $\mathsf{IBE.Setup}(\lambda)$, the runtime is $\mathsf{poly}(\lambda)$ whenever the runtime of $\mathsf{IBE.Setup}(\lambda)$ is $\mathsf{poly}(\lambda)$. Next, the algorithm $\mathsf{KeyGen}(\mathsf{msk}, T)$ runs $\mathsf{IBE.KeyGen}(\mathsf{msk}, \cdot)$ a total of $n+2$ times for each transition of T. Since n is bounded by $\mathsf{poly}(\lambda)$, we have that if the runtime of $\mathsf{IBE.KeyGen}(\mathsf{msk}, \cdot)$ is $\mathsf{poly}(\lambda)$ then $\mathsf{KeyGen}(\mathsf{msk}, T)$ has runtime $|T| \cdot \mathsf{poly}(\lambda)$. Next, the algorithm $\mathsf{Enc}(\mathsf{pp}, m, (w, t))$ runs $\mathsf{GData}(1^\lambda, D)$ on a database of size $O(t)$, and garbles $t+1$ copies of the step-program P. Assume $|w| \le t$ and that each P has representation of size $\mathsf{poly}(\lambda) \cdot \mathsf{polylog(t)}$. If $\mathsf{IBE.Enc}(\mathsf{pp}, \cdot)$ has runtime $\mathsf{poly}(\lambda)$, $\mathsf{GData}(1^\lambda, D)$ has runtime $|D| \cdot \mathsf{polylog(|D|)} \cdot \mathsf{poly}(\lambda)$, and $\mathsf{GProg}(1^\lambda, \log(|D|), P, 1^n, (t_{\mathsf{init}}, t_{\mathsf{fin}}))$ has runtime $|P| \cdot \mathsf{polylog(|D|, |P|)} \cdot \mathsf{poly}(\lambda)$, then $\mathsf{Enc}(\mathsf{pp}, m, (w, t))$ has runtime $t \cdot \mathsf{polylog(t)} \cdot \mathsf{poly}(\lambda)$. Finally, the algorithm $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct})$ evaluates a garbled program and decrypts a set of n ciphertexts of the IBE system $t' + 1$ many times, where t' is the time T takes to accept the underlying attribute w used to compute ct. Thus, if $\mathsf{IBE.Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$ has runtime $\mathsf{poly}(\lambda)$ and if $\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}, \mathbf{lab})$ has runtime $|P| \cdot \mathsf{polylog(|D|)} \cdot \mathsf{poly}(\lambda)$ then $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct})$ has runtime $t' \cdot \mathsf{polylog(t)} \cdot \mathsf{poly}(\lambda)$ where t' is the time T takes to accept the attribute w used when computing ct and t is the time bound set at encryption time when computing ct.

4.4 Security

Next, we prove the following.

Theorem 4.1. Let IBE be a secure IBE scheme as per Definition 2.6, SKE be a secure symmetric key encryption scheme as per Definition 2.4, and GRAM be a garbled RAM scheme satisfying Iterated Simulation Security as per Definition 3.1. Then ABE described above is an ABE scheme satisfying 1-query key-selective security as per Definition 2.3.

We prove Theorem 4.1 via a sequence of hybrid games. First, we describe the games and later on prove that any two adjacent games are indistinguishable.

Game 0. This game corresponds to the original 1-query key-selective security game.

- **Setup Phase**: The challenger chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$, and sends pp to the adversary. (Note that Setup in our scheme is precisely $\mathsf{IBE.Setup}$.)
- **Key Query Phase**: The adversary submits a single key query for machine T to the challenger. Let the Turing machine T be given as the set of possible transitions it can make under its transition relation δ:
$$T = \{(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir}) \ : \ \delta(q^{\mathsf{in}}, b^{\mathsf{in}}) = (q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir})\}.$$
Let $\mathcal{ID}_T$ be the set of $(n+2) \cdot |T|$ identities as defined in Eq. (1). The challenger samples an IBE secret key for each identity in $\mathcal{ID}_T$. Concretely, it chooses
$$\forall\ (q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta) \in \mathcal{ID}_T, \quad \mathsf{sk}_{(q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta)} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{msk}, (q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta)).$$
Finally, it sends the key $\mathsf{sk}_T = \left(T, \left\{\mathsf{sk}_{(q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta)}\right\}_{(q^{\mathsf{in}}, b^{\mathsf{in}}, i, \beta) \in \mathcal{ID}_T}\right)$ to $\mathcal{A}$.
- **Challenge Phase**: The adversary submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to the challenger. It must be the case that the machine T for which the adversary was given a secret key sk_T during the key query phase does not accept the word w within t steps.

  The challenger samples a bit $\beta \leftarrow \{0, 1\}$, and computes the challenge ciphertext as follows.

  1. The challenger sets a database $D \in \{0,1\}^{t+1+\lceil \log(t+1) \rceil}$. It sets the first $|w|$ bits of D to match w, and sets the remaining $\lceil \log(t+1) \rceil$ bits to 0. More formally,
  $$D := w || 0^{t+1+\lceil \log(t+1) \rceil - |w|}$$
  where $||$ denotes concatenation. It next garbles the database $(\widetilde{D}, \mathsf{k}_D) \leftarrow \mathsf{GData}(1^\lambda, D)$.

2. For each $(i, b, j) \in [n+2] \times \{0,1\} \times [t+1]$, the challenger samples randomness $r_{i,b}^{(j)}$ and SKE secret keys $\mathsf{K}_{i,b}^{(j)}$ as $r_{i,b}^{(j)} \leftarrow \{0,1\}^\lambda, \mathsf{K}_{i,b}^{(j)} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
3. Let P be the RAM program described as described in Fig. 1. For each $j \in [t+1]$, the challenger sets P_j as $P_j := P[\mathsf{pp}, \{\mathsf{K}_{i,b}^{(j)}\}_{i,b}, m_\beta, j; \{r_{i,b}^{(j)}\}_{i,b}]$.
4. Let ℓ be the number of steps P takes to run on a database of length $|D|$. For each $j \in [t+1]$, the challenger garbles the program P_j, computing
$$(\widetilde{P_j}, \{\mathsf{lab}_{i,b}^{\mathsf{in},j}\}_{i,b}) \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, t+1+\lceil\log(t+1)\rceil, P_j, 1^{n+2}, (1+(j-1)\cdot\ell, j\cdot\ell)).$$
5. For each $(i, b, j) \in [n+2] \times \{0,1\} \times [t]$, the challenger computes ciphertexts $\widetilde{\mathsf{ct}}_{i,b}^{(j)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{i,b}^{(j)}, \mathsf{lab}_{i,b}^{\mathsf{in},j+1})$.
6. Let $\{\mathsf{lab}_{i,b}^{\mathsf{in},1}\}_{i,b}$ be the set of input labels computed when garbling program P_1. Recall that the all zero state is the canonical start state. The challenger outputs the ciphertext
$$\mathsf{ct}^* = (w, t, \widetilde{D}, \{\mathsf{lab}_{i,0}^{\mathsf{in},1}\}_{i\in[n+2]}, \{\widetilde{P}_j\}_{j\in[t+1]}, \{\widetilde{\mathsf{ct}}_{i,b}^{(j)}\}_{i\in[n+2],b\in\{0,1\},j\in[t]}).$$

- **Guess Phase**: The adversary submits its guess β', and wins the game if $\beta = \beta'$.

Game k.1 $(1 \le k \le t+1)$. This game is defined similar to Game 0, except now the challenger simulates the first k (out of $t+1$) garbled RAM programs and the SKE ciphertexts encrypting the labels for first $k-1$ levels are also simulated (i.e., half of them contain the simulated wire label keys, while other half encrypt all zeros). Note that while setting up the garbled programs to be simulated, the challenger needs to sample the IBE ciphertexts appropriately where the IBE ciphertexts for the first $k-1$ simulated garbled programs encrypt only half of the corresponding SKE keys and the IBE ciphertexts for the k-th simulated program encrypts all the keys honestly. Below we describe it in detail highlighting the differences.

- **Challenge Phase**: The adversary submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to the challenger. It must be the case that the machine T for which the adversary was given a secret key sk_T during the key query phase does not accept the word w within t steps.

  Let $\{(q_j^{\mathsf{in}}, b_j^{\mathsf{in}}, q_j^{\mathsf{out}}, b_j^{\mathsf{out}}, \mathsf{dir}_j)\}_{j\in[t]}$ be the sequence of the first t transitions made by machine T on input w. Let $x_1 = 0^{n+2}$, and for all other j let x_j be the $(n+2)$-bit representation of $(q_{j-1}^{\mathsf{out}}, b_{j-1}^{\mathsf{out}}, b_{j-1}^{\mathsf{dir}})$. Let $D \in \{0,1\}^{t+1+\lceil\log(t+1)\rceil}$ match w in the first $|w|$ bits, and be 0 elsewhere. Let $\{\mathsf{access}_j\}_{j\in[t+1]}$ be the memory access patterns of the $t+1$ step programs P_j run on D in sequence with inputs x_j. Note that the hard-coded inputs do not affect the memory access pattern, so for all $j \in [t+1]$, access_j can be computed as a function of the machine T, the challenge attribute w, and the time bound t.

  The challenger samples a bit $\beta \leftarrow \{0,1\}$, and computes the challenge ciphertext as follows.

  1. The challenger sets a database $D \in \{0,1\}^{t+1+\lceil\log(t+1)\rceil}$. It sets the first $|w|$ bits of D to match w, and sets the remaining $\lceil\log(t+1)\rceil$ bits to 0. More formally,
  $$D := w||0^{t+1+\lceil\log(t+1)\rceil-|w|}$$
  where $||$ denotes concatenation. It next garbles the database $(\widetilde{D}, \mathsf{k}_D) \leftarrow \mathsf{GData}(1^\lambda, D)$.
  2. For each $(i, b, j) \in [n+2] \times \{0,1\} \times [t+1]$, the challenger samples randomness $r_{i,b}^{(j)}$ and SKE secret keys $\mathsf{K}_{i,b}^{(j)}$ as $r_{i,b}^{(j)} \leftarrow \{0,1\}^\lambda, \mathsf{K}_{i,b}^{(j)} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.[9]

[9] We point out that the challenger does not need use all the sampled random coins and secret keys anymore. However, for ease of exposition we still sample all of them as before.

3. Let P be the RAM program as described in Fig. 1. For each $j \in [k+1, t+1]$, the challenger sets P_j as $P_j := P[\mathsf{pp}, \{\mathsf{K}^{(j)}_{i,b}\}_{i,b}, m_\beta, j; \{r^{(j)}_{i,b}\}_{i,b}]$. It computes $2k(n+2)$ IBE ciphertexts as:

$$\begin{aligned} (i,b) &\in [n+2] \times \{0,1\}, & \mathsf{ct}^{(k)}_{i,b} &\leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}^{(k)}_{i,b}, (q^{\mathsf{in}}_k, b^{\mathsf{in}}_k, i, b)), \\ (i,j) &\in [n+2] \times [k-1], & \mathsf{ct}^{(j)}_{i,x_j[i]} &\leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}^{(j)}_{i,x_j[i]}, (q^{\mathsf{in}}_j, b^{\mathsf{in}}_j, i, x_j[i])), \\ (i,j) &\in [n+2] \times [k-1], & \mathsf{ct}^{(j)}_{i,1-x_j[i]} &\leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathbf{0}, (q^{\mathsf{in}}_j, b^{\mathsf{in}}_j, i, 1-x_j[i])) \end{aligned}$$

4. Let ℓ be the number of steps P takes to run on a database of length $|D|$. For each $j \in [k+1, t+1]$, the challenger garbles the program P_j, computing

$$(\widetilde{P_j}, \{\mathsf{lab}^{\mathsf{in},j}_{i,b}\}_{i,b}) \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, t+1+\lceil\log(t+1)\rceil, P_j, 1^{n+2}, (1+(j-1)\cdot\ell, j\cdot\ell)).$$

For $j \in [k]$, the challenger computes a simulated program

$$(\widetilde{P_j}, \{\mathsf{lab}^{\mathsf{in},j}_{i}\}_{i\in[n+2]}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{k}_D, |P|, (\{\mathsf{ct}^{(j)}_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}, \bot), D, \{\mathsf{access}'_j\}_{j'\in[t+1]})$$

5. Next, it computes the ciphertexts $\widetilde{\mathsf{ct}}^{(j)}_{i,b}$ as follows:

$$\begin{aligned} (i,b,j) &\in [n+2] \times \{0,1\} \times [k,t], & \widetilde{\mathsf{ct}}^{(j)}_{i,b} &\leftarrow \mathsf{SKE.Enc}(\mathsf{K}^{(j)}_{i,b}, \mathsf{lab}^{\mathsf{in},j+1}_{i,b}), \\ (i,j) &\in [n+2] \times [k-1], & \widetilde{\mathsf{ct}}^{(j)}_{i,x_{j+1}[i]} &\leftarrow \mathsf{SKE.Enc}(\mathsf{K}^{(j)}_{i,x_{j+1}[i]}, \mathsf{lab}^{\mathsf{in},j+1}_{i}), \\ (i,j) &\in [n+2] \times [k-1], & \widetilde{\mathsf{ct}}^{(j)}_{i,1-x_{j+1}[i]} &\leftarrow \mathsf{SKE.Enc}(\mathsf{K}^{(j)}_{i,1-x_{j+1}[i]}, \mathbf{0}) \end{aligned}$$

6. Let $\{\mathsf{lab}^{\mathsf{in},1}_i\}_i$ be the set of input labels computed when simulating program P_1. The challenger outputs the ciphertext

$$\mathsf{ct}^* = (w, t, \widetilde{D}, \{\mathsf{lab}^{\mathsf{in},1}_i\}_{i\in[n+2]}, \{\widetilde{P}_j\}_{j\in[t+1]}, \{\widetilde{\mathsf{ct}}^{(j)}_{i,b}\}_{i\in[n+2],b\in\{0,1\},j\in[t]}).$$

Game k.2 $(1 \leq k \leq t+1)$. This game is defined identically to Game $k.1$, except now IBE ciphertexts hardwired in the k-th simulated garbled program also encrypt only half of the corresponding SKE keys (as for first $k-1$ simulated programs). Below we simply describe the change in game description when compared with previous game.

- **Challenge Phase**: The adversary submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to the challenger. It must be the case that the machine T for which the adversary was given a secret key sk_T during the key query phase does not accept the word w within t steps. The challenger samples a bit $\beta \leftarrow \{0,1\}$, and computes the challenge ciphertext as in Game $k.1$, except the following:

  3. Let P be the RAM program as described in Fig. 1. For each $j \in [k+1, t+1]$, the challenger sets P_j as $P_j := P[\mathsf{pp}, \{\mathsf{K}^{(j)}_{i,b}\}_{i,b}, m_\beta, j; \{r^{(j)}_{i,b}\}_{i,b}]$. It computes $2k(n+2)$ IBE ciphertexts as:

$$\begin{aligned} (i,j) &\in [n+2] \times [k], & \mathsf{ct}^{(j)}_{i,x_j[i]} &\leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}^{(j)}_{i,x_j[i]}, (q^{\mathsf{in}}_j, b^{\mathsf{in}}_j, i, x_j[i])), \\ (i,j) &\in [n+2] \times [k], & \mathsf{ct}^{(j)}_{i,1-x_j[i]} &\leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathbf{0}, (q^{\mathsf{in}}_j, b^{\mathsf{in}}_j, i, 1-x_j[i])) \end{aligned}$$

Game k.3 $(1 \leq k \leq t+1)$. This game is defined identically to Game $k.2$, except now the SKE ciphertexts encrypting the garbled program labels for the $(k+1)$-th garbled program encrypt only half of the label keys (i.e., only the label keys corresponding to the k-th state transition). Below we simply describe the change in game description when compared with previous game.

- **Challenge Phase**: The adversary submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to the challenger. It must be the case that the machine T for which the adversary was given a secret key sk_T during the key query phase does not accept the word w within t steps. The challenger samples a bit $\beta \leftarrow \{0,1\}$, and computes the challenge ciphertext as in Game $k.2$, except the following:

  5. Next, it computes the ciphertexts $\widetilde{\mathsf{ct}}_{i,b}^{(j)}$ as follows:

$$
\begin{aligned}
(i,b,j) \in [n+2] \times \{0,1\} \times [k+1,t], &\qquad \widetilde{\mathsf{ct}}_{i,b}^{(j)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{i,b}^{(j)}, \mathsf{lab}_{i,b}^{\mathsf{in},j+1}),\\
(i,j) \in [n+2] \times [k], &\qquad \widetilde{\mathsf{ct}}_{i,x_{j+1}[i]}^{(j)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{i,x_{j+1}[i]}^{(j)}, \mathsf{lab}_{i}^{\mathsf{in},j+1}),\\
(i,j) \in [n+2] \times [k], &\qquad \widetilde{\mathsf{ct}}_{i,1-x_{j+1}[i]}^{(j)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_{i,1-x_{j+1}[i]}^{(j)}, \mathbf{0})
\end{aligned}
$$

Analysis of game indistinguishability. We complete the proof by showing that adjacent hybrid games are indistinguishable. For any adversary $\mathcal{A}$ and game **Game s**, we denote by $\mathsf{Adv}_s^{\mathcal{A}}(\lambda)$, the probability that $\mathcal{A}$ wins in **Game s**. For ease of exposition, in the sequel we use **Game 0.3** to denote **Game 0**.

Lemma 4.2. If IBE is a secure IBE scheme, then for any PPT adversary $\mathcal{A}$ and $k \in [t+1]$, we have that $\mathsf{Adv}_{k.1}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{k.2}^{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. Suppose for contradiction that $\mathcal{A}$ is a PPT adversary for which $\mathsf{Adv}_{k.1}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{k.2}^{\mathcal{A}}(\lambda) = \epsilon$, where ϵ is non-negligible. We give a reduction $\mathcal{B}$ which uses $\mathcal{A}$ to break the security of IBE. In particular, our reduction $\mathcal{B}$ will break the multi-challenge security of IBE.

The reduction algorithm $\mathcal{B}$ plays a game with an IBE challenger. The reduction $\mathcal{B}$ samples a bit $\beta \leftarrow \{0,1\}$. The challenger chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$ and sends pp to $\mathcal{B}$ who forwards it to $\mathcal{A}$. Next, $\mathcal{A}$ submits a key query for machine T to the reduction $\mathcal{B}$. Let $\mathcal{ID}_T$ denote the set of identities corresponding to machine T as per Eq. (1). $\mathcal{B}$ then makes a key query for every identity in $\mathcal{ID}_T$ to the challenger, and let S denote set containing all the secret keys sent by the challenger to $\mathcal{B}$. The reduction $\mathcal{B}$ then set $\mathsf{sk}_T = S$, and it sends sk_T to $\mathcal{A}$. Now, $\mathcal{A}$ submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to $\mathcal{B}$. The reduction $\mathcal{B}$ now computes the challenge ciphertext ct^* as in Game $k.1$, except for how it computes the IBE ciphertexts which constitute the output of the k-th simulated program in step 3 of the challenge phase.

Let x_k be the $n+2$ bit representation of $(q_{k-1}^{\mathsf{out}}, b_{k-1}^{\mathsf{out}}, b_{k-1}^{\mathsf{dir}})$, and let $\{\mathsf{K}_{i,b}^{(k)}\}_{(i,b)}$ be the set of SKE secret keys chosen in step 2. First, for each $i \in [n+2]$, $\mathcal{B}$ computes $\mathsf{ct}_{i,x_k[i]}^{(k)} \leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}_{i,x_k[i]}^{(k)}, (q_k^{\mathsf{in}}, b_k^{\mathsf{in}}, i, x_k[i]))$. Next, it sends $\{(\mathsf{K}_{i,1-x_k[i]}^{(k)}, \mathbf{0}, (q_k^{\mathsf{in}}, b_k^{\mathsf{in}}, i, 1-x_k[i]))\}_{i \in [n+2]}$ as its challenge vector of message-identity tuples. (Recall that we are considering the multi-challenge version of IBE security.) Let $\{\mathsf{ct}_i^*\}_i$ denote the set of challenge ciphertexts received by $\mathcal{B}$. It then sets the ciphertexts $\mathsf{ct}_{i,1-x_k[i]}^{(k)}$ as $\mathsf{ct}_{i,1-x_k[i]}^{(k)} = \mathsf{ct}_i^*$ for $i \in [n+2]$. The remaining portion of the challenge ciphertext is computed as in Game $k.1$.

Finally, after sending the challenge ciphertext to $\mathcal{A}$, the adversary outputs a bit γ. If $\gamma = \beta$, then $\mathcal{B}$ guesses 0 to the challenger signalling that ciphertexts $\{\mathsf{ct}_i^*\}_i$ encrypt the PRF keys. Otherwise, $\mathcal{B}$ guesses 1 to the challenger signalling they encrypt all zeros. Observe that the reduction $\mathcal{B}$ perfectly simulates the view of Game $k.1$ and $k.2$ to $\mathcal{A}$, respectively, depending upon the challenger's bit. Note that $\mathcal{B}$ is an admissible adversary as per the multi-challenge IBE game, since the adversary $\mathcal{A}$ makes only a single key query for machine T such that T does not accept w after t steps, and the IBE keys queried by $\mathcal{B}$ are completely disjoint with the set of challenge identities. Thus, the lemma follows. ■

Lemma 4.3. If SKE is a secure secret key encryption scheme, then for any PPT adversary $\mathcal{A}$ and $k \in [t+1]$, we have that $\mathsf{Adv}^{\mathcal{A}}_{k.2}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k.3}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. We prove this lemma by sketching a sequence of $n+3$ intermediate hybrid games Game $k.2.h$, for each $h \in [0,\ldots,n+2]$. Game $k.2.h$ is defined similar to Game $k.2$, except for how the challenge ciphertext is computed. In particular in Game $k.2.h$, we change how the challenger proceeds in step 5 of computing the challenge ciphertext. Concretely, it computes the ciphertexts $\widetilde{\mathsf{ct}}^{(j)}_{i,b}$ as follows:

$$\begin{array}{ccc} (i,b,j) \in \begin{array}{c} [n+2] \times \{0,1\} \times [k+1,t] \\ \cup\ [h+1,n+2] \times \{0,1\} \times \{k\} \end{array}, & & \widetilde{\mathsf{ct}}^{(j)}_{i,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}^{(j)}_{i,b}, \mathsf{lab}^{\mathsf{in},j+1}_{i,b}), \\ (i,j) \in \begin{array}{c} [n+2] \times [k-1] \\ \cup\ [h] \times \{k\} \end{array}, & & \begin{array}{c} \widetilde{\mathsf{ct}}^{(j)}_{i,x_{j+1}[i]} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}^{(j)}_{i,x_{j+1}[i]}, \mathsf{lab}^{\mathsf{in},j+1}_{i}), \\ \widetilde{\mathsf{ct}}^{(j)}_{i,1-x_{j+1}[i]} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}^{(j)}_{i,1-x_{j+1}[i]}, \mathbf{0}) \end{array} \end{array}$$

In short, for each $i \leq h$, if $b \neq x_{k+1}[i]$, the encryption of label $\mathsf{lab}^{\mathsf{in},k+1}_{i,b}$ is replaced with an encryption of $\mathbf{0}$. All other steps are identical to Game $k.2$. It is immediate that Game $k.2.0$ is identical to Game $k.2$ and that Game $k.2.(n+2)$ is identical to Game $k.3$. We claim that if SKE is a secure secret key encryption scheme, that for any $\mathcal{A}$ and $h \in [n+2]$ that $\mathsf{Adv}^{\mathcal{A}}_{k.2.h}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k.2.(h-1)}(\lambda) \leq \mathrm{negl}'(\lambda)$ for some negligible function $\mathrm{negl}'(\lambda)$. The lemma follows immediately from this claim.

Suppose for contradiction that $\mathcal{A}$ is a PPT adversary for which $\mathsf{Adv}^{\mathcal{A}}_{k.2.h}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k.2.(h-1)}(\lambda) = \epsilon$, where ϵ is non-negligible. We give a reduction $\mathcal{B}$ which uses $\mathcal{A}$ to break the security of SKE. The reduction $\mathcal{B}$ samples a bit $\beta \leftarrow \{0,1\}$. It then chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$, and forwards pp to the adversary $\mathcal{A}$. The challenger chooses $\mathsf{K} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$. Next, $\mathcal{A}$ submits a key query for machine T to the reduction $\mathcal{B}$. The reduction $\mathcal{B}$ computes sk_T as in Game $k.2.(h-1)$ (equivalently $k.2$), and sends sk_T to $\mathcal{A}$. Now, $\mathcal{A}$ submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to $\mathcal{B}$. The reduction $\mathcal{B}$ now computes the challenge ciphertext ct^* as in Game $k.2.h$, except it computes the ciphertext $\widetilde{\mathsf{ct}}^{(k)}_{h,1-x_{k+1}[h]}$ by quering the SKE challenger on appropriate challenge messages.

Here x_k denotes the $n+2$ bit representation of $(q^{\mathsf{out}}_{k-1}, b^{\mathsf{out}}_{k-1}, b^{\mathsf{dir}}_{k-1})$. First, $\mathcal{B}$ sample all the SKE secret keys except the key $\mathsf{K}^{(k)}_{h,1-x_{k+1}[h]}$ which is implicitly set to the challenger's secret key. The reduction $\mathcal{B}$ sends the challenge messages $m_0 = \mathsf{lab}^{\mathsf{in},k+1}_{h,1-x_{k+1}[h]}$ and $m_1 = \mathbf{0}$, and let ct' denote the challenger's response. $\mathcal{B}$ now sets $\widetilde{\mathsf{ct}}^{(k)}_{h,1-x_{k+1}[h]} = \mathsf{ct}'$, while for all other $(j,i,b) \neq (k,h,1-x_{k+1}[h])$, it computes $\widetilde{\mathsf{ct}}^{(j)}_{i,b}$ as in Game $k.2.h$. The remaining portion of the challenge ciphertext is computed as in Game $k.2.h$.

Finally, after computing the challenge ciphertext ct^*, $\mathcal{B}$ sends it to $\mathcal{A}$. The adversary $\mathcal{A}$ now sends $\mathcal{B}$ its guess β''. If $\beta'' = \beta$, $\mathcal{B}$ guesses 0 to the challenger. Otherwise, $\mathcal{B}$ guesses 1 to the challenger. Observe that when $\beta' = 0$, the reduction $\mathcal{B}$ perfectly simulates the view of Game $k.2.(h-1)$ to $\mathcal{A}$. On the other hand, when $\beta' = 1$, the reduction $\mathcal{B}$ perfectly simulates the view of Game $k.2.h$ to $\mathcal{A}$. It immediately follows that $\mathcal{B}$ has advantage ϵ against the SKE challenger, which contradicts the security of SKE. This establishes the claim and thus the lemma.

Finally, after sending the challenge ciphertext to $\mathcal{A}$, the adversary outputs a bit γ. If $\gamma = \beta$, then $\mathcal{B}$ guesses 0 to the challenger signalling that ciphertext ct' was an encryption of the garbled label. Otherwise, $\mathcal{B}$ guesses 1 to the challenger signalling its encrypts all zeros. Observe that the reduction $\mathcal{B}$ perfectly simulates the view of Game $k.2.(h-1)$ and $k.2.h$ to $\mathcal{A}$, respectively, depending upon the challenger's bit. Note that $\mathcal{B}$ is an admissible adversary as per the SKE game, since the adversary $\mathcal{A}$ does not need the SKE secret key $\mathsf{K}^{(k)}_{h,1-x_{k+1}[h]}$ for preparing the challenge ciphertext as the garbled program which would have contained the key is already being simulated. Thus, the lemma follows. ■

Lemma 4.4. If GRAM satisfies Iterated Simulation Security, then for any PPT adversary $\mathcal{A}$ and $0 \leq k \leq t$, we have that $\mathsf{Adv}^{\mathcal{A}}_{k.3}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k+1.1}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. Suppose for contradiction that $\mathcal{A}$ is a PPT adversary for which $\mathsf{Adv}^{\mathcal{A}}_{k.3}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k+1.1}(\lambda) = \epsilon$, where

ϵ is non-negligible. We give a reduction $\mathcal{B}$ which uses $\mathcal{A}$ to break the Iterated Simulation Security property of GRAM.

The reduction algorithm $\mathcal{B}$ plays a game with a GRAM challenger. The reduction $\mathcal{B}$ samples a bit $\beta \leftarrow \{0,1\}$. The reduction $\mathcal{B}$ then chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$, and sends pp to $\mathcal{A}$. Next, $\mathcal{A}$ submits a key query for machine T to the reduction $\mathcal{B}$. The reduction $\mathcal{B}$ computes sk_T as it is computed in Game $k.3$. Then, $\mathcal{B}$ sends sk_T to $\mathcal{A}$. Now, $\mathcal{A}$ submits two challenge messages (m_0, m_1) and the challenge attribute and time bound $(w, 1^t)$ to $\mathcal{B}$. The reduction $\mathcal{B}$ now computes the challenge ciphertext ct^* as in Game $k.3$, except it simulates the $(k+1)$-th garbled program instead of computing honestly.

The reduction $\mathcal{B}$ sets up the database D as in Game $k.3$. Let $x_1 = 0^{n+2}$, and for all other j let x_j be the $n+2$ bit representation of $(q^{\mathsf{out}}_{j-1}, b^{\mathsf{out}}_{j-1}, b^{\mathsf{dir}}_{j-1})$. It samples the random coins $r^{(j)}_{i,b}$ and SKE secret keys $\mathsf{K}^{(j)}_{i,b}$ as in step 2. For each $j \in [t+1]$, the reduction $\mathcal{B}$ sets program P_j as

$$P_j := P[\mathsf{pp}, \{\mathsf{K}^{(j)}_{i,b}\}_{(i,b)}, m_\beta, j; \{r^{(j)}_{i,b}\}_{(i,b)}].$$

The reduction sends $(k+1, D, \{(P_j, x_j, n+2, (1+(j-1)\cdot\ell, j\cdot\ell))\}_j)$ to the challenger. The challenger, garbles the database D to compute $\widetilde{D}$, and then honestly garbles the programs P_j for $j \in [k+2, t+1]$, while P_{k+1} is either garbled honestly or simulated, and remaining programs $\widetilde{P_j}$, for $j \in [k]$, are simulated. Finally, the challenger sends $(\widetilde{D}, \{\widetilde{P_j}, \{\mathsf{lab}^{\mathsf{in},j}_i\}_i\}_{j\in[k+1]}, \{\widetilde{P_j}, \{\mathsf{lab}^{\mathsf{in},j}_{i,b}\}_{i,b}\}_{j\in[k+2,t+1]})$ to $\mathcal{B}$.

From this point, the reduction simply computes the challenge ciphertext as in Game $k.3$ but using the garbled database, programs, and input labels as provided by the challenger. Finally, after sending the challenge ciphertext to $\mathcal{A}$, the adversary outputs a bit γ. If $\gamma = \beta$, then $\mathcal{B}$ guesses 0 to the challenger signalling that $\widetilde{P_{k+1}}$ was honestly garbled. Otherwise, $\mathcal{B}$ guesses 1 to the challenger signalling it was simulated. Note that since the reduction $\mathcal{B}$ does not need the garbled labels for $(k+1)$-th garbled program while preparing the challenge ciphertext thus it can perfectly simulate the view of Game $k.3$ and $k+1.1$ to $\mathcal{A}$, respectively, depending upon the challenger's bit. Thus, the lemma follows. ∎

Lemma 4.5. For any adversary, $\mathcal{A}$ we have that $\mathsf{Adv}^{\mathcal{A}}_{t+1.1}(\lambda) = 0$.

Proof. This lemma is immediate, as in Game $t+1.1$, the challenge ciphertext consists only of simulated programs all of which are completely independent of the challenge message m_β. ∎

By combining the above lemmas, the theorem follows.

Acknowledgements

We thank Xiong Fan and Venkata Koppula for useful discussions in the early stages of this work. We also thank Daniel Wichs for helpful discussions about the GHRW garbled RAM scheme [GHRW14a].

References

[AC17] Shashank Agrawal and Melissa Chase. Simplifying design and analysis of complex predicate encryption schemes. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 627–656, 2017.

[AFS19] Prabhanjan Ananth, Xiong Fan, and Elaine Shi. Towards attribute-based encryption for rams from LWE: sub-linear decryption, and more. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2019.

[AMY19a] Shweta Agrawal, Monosij Maitra, and Shota Yamada. Attribute based encryption (and more) for nondeterministic finite automata from LWE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 765–797. Springer, 2019.

[AMY19b] Shweta Agrawal, Monosij Maitra, and Shota Yamada. Attribute based encryption for deterministic finite automata from dlin. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 91–117. Springer, 2019.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 125–153, 2016.

[AS17] Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 557–577. Springer, 2014.

[Att16] Nuttapong Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 591–623, 2016.

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 174–198. Springer, 2019.

[BCG$^+$18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 52–73, 2014.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[BL15] Xavier Boyen and Qinyi Li. Attribute-based encryption for finite automata from LWE. In Man Ho Au and Atsuko Miyaji, editors, *Provable Security - 9th International Conference,* *ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, volume 9451 of *Lecture Notes in Computer Science*, pages 247–267. Springer, 2015.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[CDG$^+$17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In *CRYPTO 2017*, 2017.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM, 1996.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[DG17a] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *CRYPTO 2017*, 2017.

[DG17b] Nico Dttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. TCC, 2017.

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

[FJP15] Georg Fuchsbauer, Zahra Jafargholi, and Krzysztof Pietrzak. A quasipolynomial reduction for generalized selective decryption on trees. In *Annual Cryptology Conference*, pages 601–620. Springer, 2015.

[FKPR14] Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained prfs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 82–101. Springer, 2014.

[GHL$^+$14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 405–422. Springer, 2014.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC 2018*, 2018.

[GHRW14a] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Garbled ram revisited, part i. Cryptology ePrint Archive, Report 2014/082, 2014. https://eprint.iacr.org/2014/082.

[GHRW14b] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private ram computation. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 404–413. IEEE, 2014.

[GKP$^+$13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564, 2013.

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, 2016.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled ram. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 210–229. IEEE, 2015.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 449–458, 2015.

[GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled ram from laconic oblivious transfer. In *Annual International Cryptology Conference*, pages 515–544. Springer, 2018.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98. ACM, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In *FOCS 2017*, 2017.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

[GW20] Junqing Gong and Hoeteck Wee. Adaptively secure ABE for DFA from k-lin and more. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 278–308. Springer, 2020.

[GWW19] Junqing Gong, Brent Waters, and Hoeteck Wee. ABE for DFA from k-lin. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 732–764. Springer, 2019.

[IPS15] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 668–697. Springer, 2015.

[JKK$^+$17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In *Annual International Cryptology Conference*, pages 133–163. Springer, 2017.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. 2021.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of yaos garbled circuits. In *Theory of Cryptography Conference*, pages 433–458. Springer, 2016.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 419–428, New York, NY, USA, 2015. ACM.

[KNTY19] Fuyuki Kitagawa, Ryo Nishimaki, Keisuke Tanaka, and Takashi Yamakawa. Adaptively secure and succinct functional encryption: improving security and efficiency, simultaneously. In *Annual International Cryptology Conference*, pages 521–551. Springer, 2019.

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

[KW19a] Venkata Koppula and Brent Waters. Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In *CRYPTO 2019*, 2019.

[KW19b] Lucas Kowalczyk and Hoeteck Wee. Compact adaptively secure abe for nc1 from k-lin. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2019.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs? In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 719–734. Springer, 2013.

[LO14] Steve Lu and Rafail Ostrovsky. Garbled ram revisited, part ii. Cryptology ePrint Archive, Report 2014/083, 2014. https://eprint.iacr.org/2014/083.

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

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 463–472, New York, NY, USA, 2010. ACM.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[Wat12] Brent Waters. Functional encryption for regular languages. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 218–235. Springer, 2012.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 600–611, 2017.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.

A From 1-query key-selective security to adaptive security

In this section, we show how to lift an ABE scheme that only satisfies a weak notion of security, that is 1-query key-selective, to achieve 1-query full/adaptive security by relying on a very weak form of non-committing encryption. We start by defining the notion of non-committing encryption required for our transformation.

A.1 Weak Non-Committing Encryption

The notion of non-committing encryption was introduced by Canetti et al. [CFGN96]. At a high level, a non-committing encryption scheme enables efficient simulation of public keys and ciphertexts such that, at a later point, the simulator could "open" the ciphertext to any message by providing encryption randomness and corresponding secret keys. For security, it is mostly required that the distribution of simulated keys, ciphertexts, and key generation and encryption randomness is computationally indistinguishable from the distribution generated by the real encryption protocol.

In this work, we rely on a weak form of non-committing encryption where the simulator need not provide indistinguishable random coins used for encryption and key generation. Formally, we define it below.

Syntax. A weak non-committing encryption (wNCE) system wNCE for message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of five polynomial time algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim\text{-}Setup}, \mathsf{Sim\text{-}Open})$ with the following syntax:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$. On input the security parameter λ, it outputs a public-secret key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. It takes as input a public key pk and a message $m \in \mathcal{M}_\lambda$, and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$. It takes as input a secret key sk and a ciphertext ct, and outputs a message $m \in \mathcal{M}_\lambda$.

$\mathsf{Sim\text{-}Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{ct}, t)$. On input the security parameter λ, it outputs a tuple of a public key pk, a ciphertext ct, and trapdoor information t.

$\mathsf{Sim\text{-}Open}(t, m) \to \mathsf{sk}$. On input the trapdoor t and a message $m \in \mathcal{M}_\lambda$, it outputs a secret key sk.

Correctness. A weak non-committing encryption scheme wNCE is said to be correct if for all $\lambda \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $m \in \mathcal{M}_\lambda$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, we have that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m$.

Security. For security, we require the following weak simulatability property which already implies semantic security.

Definition A.1 (weak simulatability)**.** We say a secret key encryption scheme $\mathsf{wNCE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Sim\text{-}Setup}, \mathsf{Sim\text{-}Open})$ is weakly simulation secure if for any stateful PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$, such that for every $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\mathsf{sk}_b, \mathsf{ct}_b) = b : \begin{array}{c} (\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{pk}_1, \mathsf{ct}_1, t) \leftarrow \mathsf{Sim\text{-}Setup}(1^\lambda) \\ b \leftarrow \{0,1\}, \quad m \leftarrow \mathcal{A}(\mathsf{pk}_b) \\ \mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, m), \\ \mathsf{sk}_1 \leftarrow \mathsf{Sim\text{-}Open}(t, m) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Weak NCE for bit messages from PKE

In this section, we show a simple construction for weak NCE from regular public key encryption. We start with a public-key bit encryption scheme, and use it build a weak non-committing bit encryption scheme, that is $\mathcal{M} = \{0,1\}$. By a standard hybrid argument (i.e., the repitition scheme), one could extend it to encrypt multi-bit messages, but that has the limitation of key sizes growing linearly with the bit length of the messages. Later on we sketch a simple extension in the Random Oracle Model (ROM) [BR93] that avoids this key size limitation. Below we sketch a simple scheme wNCE from any plain public key bit encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$.

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$. The algorithm samples two independent public-secret key pairs $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ for $b \in \{0,1\}$, and a random bit $\beta \leftarrow \{0,1\}$. It outputs the public-secret key pair as $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1)$ and $\mathsf{sk} = (\beta, \mathsf{sk}_\beta)$.

$\mathsf{Enc}(\mathsf{pk}, m) \rightarrow \mathsf{ct}$. Let $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1)$. It encrypts m independently under both keys as $\mathsf{ct}_b \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_b, m)$ for $b \in \{0,1\}$, and outputs the ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow m$. Let $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$ and $\mathsf{sk} = (\beta, \mathsf{sk}_\beta)$. It decrypts ct_β using key sk_β, and thus outputs the message as $m \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_\beta, \mathsf{ct}_\beta)$.

$\mathsf{Sim\text{-}Setup}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{ct}, t)$. The algorithm samples two independent public-secret key pairs $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ for $b \in \{0,1\}$, and a random bit $\delta \leftarrow \{0,1\}$. Next, it computes two PKE ciphertexts as $\mathsf{ct}_b \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_b, \delta \oplus b)$ for $b \in \{0,1\}$, and outputs the ciphertext as $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, public key as $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1)$, and trapdoor as $t = (\mathsf{sk}_0, \mathsf{sk}_1, \delta)$.

$\mathsf{Sim\text{-}Open}(t, m) \rightarrow \mathsf{sk}$. Let $t = (\mathsf{sk}_0, \mathsf{sk}_1, \delta)$. It simply outputs the secret key as $\mathsf{sk} = (\delta \oplus m, \mathsf{sk}_{\delta \oplus m})$.

The correctness of the above scheme follows directly from the correctness of the underlying PKE system. For security, we could very easily reduce weak simulatability to semantic security of the underlying PKE system. We provide a brief proof sketch below.

Theorem A.2. If PKE is an IND-CPA secure public key bit encryption scheme, then wNCE scheme described above is a weakly simulation secure non-committing bit encryption scheme as per Definition A.1.

Proof sketch. The proof of weak simulation security follows mostly directly from the semantic security of the PKE system. First, observe that since δ is a random bit, thus $\delta \oplus m$ is a random bit as well. Therefore, the distribution of secret keys is identical in both cases (honest and simulated). Next, observe that the only difference between honestly computed and simulated ciphertext is the way ciphertext component $\mathsf{ct}_{1 \oplus \delta \oplus m}$ is created. In the honest case, it encrypts message m honestly, whereas in the simulated case it encryption $1 \oplus m$. Now since the adversary never received the corresponding PKE secret key, thus a reduction could simply guess the message bit m that it has to be simulate and use semantic security of PKE to switch between honestly computed and simulated ciphertext.

■

Succinct Weak NCE for arbitrarily long messages via ROM

Here we briefly sketch a simple transformation that takes any weakly simulation secure non-committing encryption scheme for fixed length messages and builds another wNCE scheme where the size of public-secret keys does not grow with the length of the messages. We want to emphasize in the ROM, the simulation algorithms also get to program the RO.

The idea is as follows. We will start with a wNCE scheme that encrypts λ-bit messsages, and a hash function $H : \{0,1\}^{2\lambda} \rightarrow \{0,1\}^\lambda$ which we model as a random oracle. During setup, we simply sample a public-secret key pair for the underlying (non-succinct) wNCE scheme. To encrypt a message $m \in \{0,1\}^*$, we first sample a λ-bit random string $K \leftarrow \{0,1\}^\lambda$, and encrypt K using the underlying wNCE scheme to obtain first part of the ciphertext. Next, it creates masking values $r_i = H(K, i)$ for $i \in \{1, \ldots, \lceil |m|/\lambda \rceil\}$, and encrypts each λ-bit block of message as $c_i = r_i \oplus m[1 + \lambda(i-1) : \lambda i]$. To decrypt, one first decrypt the underlying wNCE ciphertext to recover the masking key K, and then uncomputes the masking values by re-computing them using the hash function H.

Now this scheme is succinct since the size of public and secret keys grows only polynomially with λ since we only sample a single key pair for the underlying wNCE scheme for λ-bit messages. Also, the weak simulatability of the system follows weak simulatability of the underlying wNCE scheme and programmability of the random oracle H. Briefly, the simulator computes the first part of the simulated ciphertext as a simulated wNCE ciphertext for λ-bit messages, and sets the remaining part as λ-bit random strings of appropriate length. During simulation opening, it simulates the underlying wNCE ciphertext to a random

λ-bit key K. Since the adversary would not have queried the random oracle on any string of the form $(K, \cdot)$, thus the simulator could very easily program the random oracle such that the ciphertext blocks c_i open to the appropriate message blocks.

A.2 Adaptive security via weak NCE

In this section, we describe a simple transformation that takes any 1-query key-selective secure ABE scheme and converts it into an adaptively secure scheme. Let $\mathsf{sABE} = (\mathsf{sABE.Setup}, \mathsf{sABE.KeyGen}, \mathsf{sABE.Enc}, \mathsf{sABE.Dec})$ be a 1-query key-selective secure ABE scheme with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$, and $\mathsf{wNCE} = (\mathsf{wNCE.Setup}, \mathsf{wNCE.Enc}, \mathsf{wNCE.Dec}, \mathsf{wNCE.Sim\text{-}Setup}, \mathsf{wNCE.Sim\text{-}Open})$ be a weakly simulation secure non-committing encryption scheme with message space $\{0,1\}^*$.[10] We now describe the construction of our 1-query fully secure ABE scheme $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{InstKey}, \mathsf{Enc}, \mathsf{Dec})$.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$. The setup algorithm samples a key pair for both wNCE as well as underlying ABE system as $(\mathsf{nce.pk}, \mathsf{nce.sk}) \leftarrow \mathsf{wNCE.Setup}(1^\lambda)$, and $(\mathsf{abe.pp}, \mathsf{abe.msk}) \leftarrow \mathsf{sABE.Setup}$. It outputs the parameters as $\mathsf{pp} = (\mathsf{nce.pk}, \mathsf{abe.pp})$ and $\mathsf{msk} = (\mathsf{nce.sk}, \mathsf{abe.msk})$.

$\mathsf{InstKey}(\mathsf{msk}, T) \to \mathsf{sk}_T$. It parses the key as above, and runs the ABE key generation as $\mathsf{abe.sk}_T \leftarrow \mathsf{sABE.KeyGen}(\mathsf{abe.msk}, T)$. It outputs the key as $\mathsf{sk}_T = (\mathsf{nce.sk}, \mathsf{abe.sk}_T)$.

$\mathsf{Enc}(\mathsf{pp}, m, (w,t)) \to \mathsf{ct}$. It parses the key as above, and runs the ABE encryption as $\mathsf{abe.ct} \leftarrow \mathsf{sABE.Enc}(\mathsf{abe.pp}, m, (w,t))$. It then encrypts $\mathsf{abe.ct}$ using the wNCE key, and outputs the ciphertext as $\mathsf{ct} \leftarrow \mathsf{wNCE.Enc}(\mathsf{nce.pk}, \mathsf{abe.ct})$.

$\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) \to m/\bot$. It parses the key as above, and first runs the wNCE encryption to recover the ABE ciphertext as $\mathsf{abe.ct} \leftarrow \mathsf{wNCE.Dec}(\mathsf{nce.sk}, \mathsf{ct})$. It then decrypts $\mathsf{abe.ct}$ using the ABE key, and outputs $z = \mathsf{sABE.Dec}(\mathsf{abe.sk}_T, \mathsf{abe.ct})$.

The correctness and efficiency of the above scheme follows directly from the correctness and efficiency of the underlying ABE and weak NCE schemes. Below we show that if the underlying ABE scheme is 1-query key-selective secure, and the wNCE scheme is weak simulation secure, then the resulting scheme is 1-query adaptively secure. Formally, we prove the following.

Theorem A.3. If sABE is a 1-query key-selective secure ABE scheme (Definition 2.3), and wNCE is a weakly simulation secure NCE scheme (Definition A.1), then the scheme ABE described above is 1-query fully secure ABE scheme (Definition 2.2) for the same message space and predicate class as sABE.

Proof. We start by dividing the adversary into two types. We say an adversay $\mathcal{A}$ is "Type-1" if it makes its challenge encryption query before making its key query. Otherwise we say it is "Type-2". Note that a Type-2 adversary is already an admissible adversary as per the key-selective ABE security game, thus for any Type-2 adversary, security of the above ABE scheme follows directly from the key-selective security of the underlying 1-query key-selective secure ABE scheme.

Now for the case of Type-1 attackers, the proof of security following via a short sequence of hybrid games. Below we describe them more formally.

Game 0. This corresponds to the 1-query adaptive security for ABE as defined in Definition 2.2.

[10] Since the size of ciphertexts in a ABE system for TMs is not a-priori bounded, thus we rely on a weak NCE scheme with unbounded length messages. However, if the size of ciphertexts is a-priori bounded, then one could simply rely on a wNCE scheme for fixed length messages (that is, implied by regular PKE itself).

Game 1. This is same as the previous game, except now the challenger makes the following changes:

- it simulates the wNCE parameters during setup (i.e., $(\mathsf{nce.pk}, \mathsf{nce.ct}, t) \leftarrow \mathsf{wNCE.Sim\text{-}Setup}(1^\lambda)$), instead of running the wNCE setup honestly;
- note that the adversary makes an encryption query *before* making its key query, thus the challenger computes the underlying ABE ciphertext honestly (i.e., $\mathsf{abe.ct} \leftarrow \mathsf{sABE.Enc}(\mathsf{abe.pp}, m_b, (w, t))$, where b is the challenge bit and rest of inputs are provided by the attacker), and "opens" the simulated ciphertext $\mathsf{nce.ct}$ to $\mathsf{abe.ct}$ as $\mathsf{nce.sk} \leftarrow \mathsf{wNCE.Sim\text{-}Open}(t, \mathsf{abe.ct})$. It then responds with $\mathsf{nce.ct}$ as the challenge ciphertext, and when/if the adversary makes its key query, then the challenger samples the underlying ABE honestly as $\mathsf{abe.sk}_T \leftarrow \mathsf{sABE.KeyGen}(\mathsf{abe.msk}, T)$, and uses the simulated key $\mathsf{nce.sk}$ to respond to the key query as $\mathsf{sk}_T = (\mathsf{nce.sk}, \mathsf{abe.sk}_T)$.

Let $\mathsf{Adv}_i^{\mathcal{A}}(\lambda)$ denote the adversary $\mathcal{A}$'s advantage in Game i. To complete the proof, we prove the following lemmas.

Lemma A.4. If wNCE is a weakly simulation secure NCE scheme, then for any *Type-1* PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_0^{\mathcal{A}}(\lambda) - \mathsf{Adv}_1^{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. The proof of this lemma follows directly from the simulation security of the wNCE scheme. ■

Lemma A.5. If sABE is a 1-query key-selective secure ABE scheme, then for any *Type-1* PPT adversary $\mathcal{A}$, we have that $\mathsf{Adv}_1^{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. The proof of this lemma follows directly from the key-selective security of the ABE scheme. Briefly, the reduction algorithm proceeds as follows. It receives the ABE public parameters $\mathsf{abe.pp}$ from the challenger, and simulates the wNCE parameters $(\mathsf{nce.pk}, \mathsf{nce.ct}, t)$ at the beginning. It then sends $(\mathsf{nce.pk}, \mathsf{abe.pp})$ to $\mathcal{A}$ as the public key. Since $\mathcal{A}$ is a Type-1 attacker, thus it sends its challenge messages and attribute next. The reduction stores the challenge messages and attribute, and simply responds with $\mathsf{nce.ct}$ as the challenge ciphertext to $\mathcal{A}$. Next, if/when $\mathcal{A}$ makes its only key query, then the reduction algorithm forwards it as its own key query to the challenger and receives the underlying ABE key $\mathsf{abe.sk}_T$. The reduction then forwards the challenge messages and attribute as its own challenge to the underlying ABE challenger. Let $\mathsf{abe.ct}^*$ be the challenger's responce. The reduction then opens the simulated ciphertext to $\mathsf{abe.ct}^*$ as $\mathsf{nce.sk} \leftarrow \mathsf{wNCE.Sim\text{-}Open}(t, \mathsf{abe.ct}^*)$, and sends $(\mathsf{nce.sk}, \mathsf{abe.sk}_T)$ as the response to adversary's key query. Finally, $\mathcal{A}$ outputs its guess which the reduction forwards to its ABE challenger as its own guess.

First, note that the reduction algorithm is an admissible adversary as per the1-query key-selective security game since it makes its key query before asking for the challenge ciphertext. Thus, the reduction perfectly simulates Game 1 for any Type-1 adversary. Hence, the lemma follows. ■

■

B Amplifying to q-query security

In this section, we provide a construction for amplifying security of our ABE scheme described in Section 4 from 1-query security to q-query security for any a-priori chosen parameter q.[11] This mostly follows from standard combinatorial techniques, but we provide it for completeness.

Let $\mathsf{sABE} = (\mathsf{sABE.Setup}, \mathsf{sABE.KeyGen}, \mathsf{sABE.Enc}, \mathsf{sABE.Dec})$ be a single key secure ABE scheme with message space $\{\mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathbb{F}_\lambda$ is a finite field of size to be specified later. When clear from context, we omit the security parameter as subscript. Let $N, d \in \mathbb{N}$ be parameters to be set later. For describing our scheme and proving correctness we only require that the parameters satisfy $|\mathbb{F}| > N > d > 0$. We now describe the construction of our q-query secure ABE scheme $\mathsf{qkABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

[11] Before applying this transformation, we amplify the security of our construction to 1-query adaptive security via the weak NCE approach described in Appendix A.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$: The setup algorithm chooses N distinct instances of the public parameters and master secret key of sABE. In particular, for $i \in [N]$, the algorithm chooses $(\mathsf{pp}_i, \mathsf{msk}_i) \leftarrow \mathsf{sABE.Setup}(1^\lambda)$ and outputs $\mathsf{pp} = \{\mathsf{pp}_i\}_{i\in[N]}$ as the public parameters and $\mathsf{msk} = \{\mathsf{msk}_i\}_{i\in[N]}$ as the master secret key.

- $\mathsf{KeyGen}(\mathsf{msk}, T) \to \mathsf{sk}_T$: Let $\mathsf{msk} = \{\mathsf{msk}_i\}_{i\in[N]}$. The key generation algorithm chooses several sABE secret keys as follows. The algorithm selects a random subset $S \subset [N]$ of size $d+1$. For each $i \in S$, the algorithm chooses $\mathsf{sk}_{T,i} \leftarrow \mathsf{sABE.KeyGen}(\mathsf{msk}_i, T)$. Finally, the algorithm outputs $\mathsf{sk}_T = (S, \{\mathsf{sk}_{T,i}\}_{i\in S})$ as the secret key.

- $\mathsf{Enc}(\mathsf{pp}, m, (w, t)) \to \mathsf{ct}$: Let $\mathsf{pp} = \{\mathsf{pp}_i\}_{i\in[N]}$. The encryption algorithm samples a random degree d polynomial $p(\cdot)$ with coefficients in $\mathbb{F}$ such that it has constant term $p(0) = m$. The algorithm randomly samples N distinct non-zero elements $x_i \in \mathbb{F}$, and for each $i \in [N]$ computes the intermediate ciphertexts $\mathsf{ct}_i \leftarrow \mathsf{sABE.Enc}(\mathsf{pp}_i, (x_i, p(x_i)), (w, t))$. Finally, the algorithm outputs $\mathsf{ct} = \{\mathsf{ct}_i\}_{i\in[N]}$ as the ciphertext.

- $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) \to m$ or $\perp$: Let $\mathsf{sk}_T = (S, \{\mathsf{sk}_{T,i}\}_{i\in S})$. Parse the ciphertext as $\{\mathsf{ct}_i\}_{i\in[N]}$. For each $i \in S$, the algorithm computes $y_i \leftarrow \mathsf{sABE.Dec}(\mathsf{sk}_{T,i}, \mathsf{ct}_i)$. If for any $i \in S$, $(x_i, y_i) = \perp$, the algorithm outputs $\perp$. Otherwise, the algorithm interpolates a degree d polynomial $p(\cdot)$ consistent with the $d+1$ points (x_i, y_i). Finally the algorithm computes $m = p(0)$ and outputs m.

Setting parameters We now briefly discuss the setting of parameters N, d as well as the size of $\mathbb{F}$. We will use the following lemma.

Lemma B.1. Fix $q, \lambda \in \mathbb{N}$. There exist universal constants $c_1, c_2 > 0$ such that the following holds. Let $d > c_1 q^2 \lambda$, $N > c_2 q^4 \lambda$. Sample uniformly random subsets $S_1, \dots, S_q \subset [N]$ each of size $d+1$. We have

$$\Pr\left[|\cup_{i<j} S_i \cap S_j| > d\right] \leq 2^{-\Theta(\lambda)}$$

The proof follows from a standard probabilistic argument. We refer to [GVW12, Lemma B.2] for details. For our scheme, we will take d, N so as to satisfy the hypothesis of this lemma. We will take $\mathbb{F}$ to be any finite field of size larger than N.

Correctness We show that the above construction is correct. Fix $m, (w, t), T$, and let $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk}_T \leftarrow \mathsf{KeyGen}(\mathsf{msk}, T)$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m, (w, t))$. Assume T accepts w within t steps, and let $\mathsf{sk}_T = (S, \{\mathsf{sk}_{T,i}\}_{i\in S})$ Let $p(\cdot)$ be the polynomial sampled when computing ct. By the correctness of sABE, the decryption algorithm recovers k pairs $(x_i, p(x_i))$ for $i \in S$. Since a degree d polynomial is uniquely determined by $d+1$ points, the decryption algorithm correctly recovers the polynomial $p(\cdot)$. Thus by the construction the decryption algorithm recovers the message $m = p(0)$.

Efficiency We briefly remark on the efficiency of the scheme we construct. The polynomial bounds on the runtimes of the algorithms are inherited from the underlying single key secure system, albeit with some loss depending on the collusion bound q. In particular,if we construct the scheme using the construction presented in Section 4 as a building block, we have that the algorithm $\mathsf{KeyGen}(\mathsf{msk}, T)$ runs in time $\widetilde{O}(q^2 \cdot |T|)$, the algorithm $\mathsf{Enc}(\mathsf{pp}, m, (w, t))$ runs in time $\widetilde{O}(q^4 \cdot t)$, and $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct})$ runs in time $\widetilde{O}(q^4 \cdot t)$ where $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m, (w, t))$. We remark that using techniques from [AV19] it is possible to bootstrap our construction to construct a q key query secure scheme such that the dependence on q in all of the above bounds is linear.

B.1 Proof of q-query Security

We prove our construction of qkABE satisfies q-query security. The high level proof idea is as follows. Let $(S_1, \{\mathsf{sk}_{T_1,i}\}_{i\in S_1})$, $\dots$, $(S_q, \{\mathsf{sk}_{T_q,i}\}_{i\in S_q})$ be the responses of the challenger to the q secret key queries made by the adversary. For each machine T_j, we have that T_j does not accept the challenge word w within t steps.

Thus, for each index $i \in [N]$ which is contained in at most one S_j, the security of sABE implies that ct_i is indistinguishable from the encryption of a random message. On the other hand, if i is contained in multiple S_j, the adversary may learn some information about $(x_i, p(x_i))$. However, by our selection of parameters and Lemma B.1, with overwhelming probability the number of i for which this happens will be at most d, and in this case the value of $m = p(0)$ remains information theoretically hidden. The proof proceeds via a sequence of hybrid games. We describe the games here and then prove adjacent games are indistinguishable.

Game 0 : This game corresponds to the original security game.

- **Setup Phase**: The challenger samples a bit $\beta \leftarrow \{0,1\}$. The challenger chooses N distinct instances of the public parameters and master secret key of sABE. In particular, for $i \in [N]$, the algorithm chooses $(\mathsf{pp}_i, \mathsf{msk}_i) \leftarrow \mathsf{sABE.Setup}(1^\lambda)$ and outputs $\mathsf{pp} = \{\mathsf{pp}_i\}_{i \in [N]}$ to the adversary. It sets $\mathsf{msk} = \{\mathsf{msk}_i\}_{i \in [N]}$ as the master secret key.

- **Key Query Phase**: The adversary submits up to q key queries for Turing Machines T_j to the challenger. Let $\mathsf{msk} = \{\mathsf{msk}_i\}_{i \in [N]}$. For $j \in [q]$, on the jth query, the challenger chooses several sABE secret keys as follows. The algorithm selects a random subset $S_j \subset [N]$ of size k. For each $i \in S$, the challenger chooses $\mathsf{sk}_{T_j,i} \leftarrow \mathsf{sABE.KeyGen}(\mathsf{msk}_i, T_j)$. Finally, the challenger outputs $\mathsf{sk}_{T_j} = (S_j, \{\mathsf{sk}_{T_j,i}\}_{i \in S_j})$ as the jth secret key. We note that the adversary may adaptively choose subsequent machines to query based on the secret keys it receives in response to its queries.

- **Challenge Phase**: The adversary submits two challenge messages (m_0, m_1) and the challenge attribute (w, t) to the challenger. Let $\mathsf{pp} = \{\mathsf{pp}_i\}_{i \in [N]}$. The challenger samples a random degree d polynomial $p(\cdot)$ with coefficients in $\mathbb{F}$ such that it has constant term $p(0) = m_\beta$. The challenger samples N distinct non-zero random elements $x_i \in \mathbb{F}$, and for each $i \in [N]$ computes the intermediate ciphertexts $\mathsf{ct}_i \leftarrow \mathsf{sABE.Enc}(\mathsf{pp}_i, (x_i, p(x_i)), (w, t))$. Finally, the challenger outputs $\mathsf{ct}^* = \{\mathsf{ct}_i\}_{i \in [N]}$ as the challenge ciphertext.

- **Key Query Phase**: Same as pre-challenge query phase (if proving full security; disallowed for key-selective security).

- **Guess Phase**: The adversary submits its guess β', and wins the game if $\beta = \beta'$.

Game ℓ : for $\ell \in [N]$. In the ℓ-th game, we modify how the challenge ciphertext $\mathsf{ct}^* = \{\mathsf{ct}_i\}_{i \in [N]}$ is formed. Let $I \subset [\ell]$ consist of all i' such that for two distinct j, j', $i \in S_j \cap S_{j'}$. If $i \notin I$, then we replace ct_i with the encryption of a random message. Otherwise ct_i is computed as in **Game 0**.

We now prove a sequence of lemmas showing that adjacent hybrid games are indistinguishable. For any adversary $\mathcal{A}$ and **Game ℓ**, we denote by $\mathsf{Adv}_\ell^{\mathcal{A}}(\lambda)$, the advantage of $\mathcal{A}$ in that game **Game ℓ**, where the advantage is $\mathcal{A}$'s success probability less 1/2.

Lemma B.2. If sABE is a single key secure ABE scheme, then for any PPT adversary $\mathcal{A}$ and $\ell \in [N]$, $\mathsf{Adv}_\ell^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{\ell-1}^{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. Observe that the difference between **Game $\ell - 1$** and **Game ℓ** is in how the challenge ciphertext is generated. In particular, if ℓ is not contained in two distinct $S_j, S_{j'}$ sampled during the key query phase, then ct_i is replaced with the encryption of a random message. In this case, since no machine queried accepts w within t steps, indistinguishability of the two games follows immediately from security of sABE, which holds since for the ℓ-th instance of the single key scheme $\mathcal{A}$ receives a single key. In the case that ℓ was sampled in two distinct $S_j, S_{j'}$, the games are identical. The lemma follows. ■

Lemma B.3. Let c_1, c_2 be as in Lemma B.1. If $d > c_1 q^2 \lambda$, $N > c_2 q^4 \lambda$, and $|\mathbb{F}| > N$, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_N^{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for some negligible function $\mathrm{negl}(\cdot)$.

Proof. Let $I \subset [N]$ be defined as in **Game N**. Suppose that $|I| \leq d$, and let $p(\cdot)$ be the polynomial sampled by the challenger during the challenge phase. Observe that the challenge ciphertext only consists of encryptions of d many points of p. Indeed, for all $i \notin I$, ct_i is independent of of m_β, and all other ct_i. Moreover, given up to d samples $\{(x_i, p(x_i))\}_{i \in I}$, p is not fully determined, and the constant term $m_\beta = p(0)$ remains information theoretically hidden. Thus, in this case ct^* is independent of β and thus $\mathcal{A}$ has advantage 0. On the other hand, by Lemma B.1, $|I| > d$ with probability exponentially small in λ. The lemma follows. ■

The q-query security of our scheme follows immediately from the above two lemmas.

C Garbled RAM: Iterated Simulation Security of [GHRW14a]

Our ABE construction uses a Garbled RAM scheme GRAM which satisfies our notion of Iterated Simulation Security. [GHRW14a] construct a basic Garbled RAM scheme which satisfies security with Unprotected Memory Access (UMA), a simulation based notion of security where simulator is given the memory access patterns of the programs being simulated. Using Oblivious RAM they then bootstrap this basic scheme to satisfy a stronger notion of full security. In this appendix we sketch the construction of the basic UMA secure scheme of [GHRW14a], and prove that it already satisfies Iterated Simulation Security. We review the required primitives before formally describing the scheme and then proving our security claim.

C.1 Timed IBE

The scheme of [GHRW14a] uses a variant of IBE (in fact a variant of hierarchical IBE) which the authors name Timed IBE. Moreover, they show that Timed IBE can be constructed from any selectively-secure IBE scheme. We review the syntax and security definition here, and refer to [GHRW14a] for the construction and security proof. A Timed Identity-Based Encryption (TIBE) scheme TIBE for set of identity spaces $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda \in \mathbb{N}}$ and message spaces $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of five polynomial time algorithms $(\mathsf{Setup}, \ \mathsf{TimeGen}, \mathsf{KeyGen}, \ \mathsf{Enc}, \ \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$: The setup algorithm takes as input the security parameter λ. It outputs the public parameters pp and the master secret key msk.
- $\mathsf{TimeGen}(\mathsf{msk}, j) \to \mathsf{tsk}_j$: The time-key generation algorithm takes as input the master secret key msk and an integer $j \in \mathbb{N}$. It outputs a time secret key tsk_j.
- $\mathsf{KeyGen}(\mathsf{tsk}_j, (j, v)) \to \mathsf{sk}_{(j,v)}$: The key generation algorithm takes as input a time secret key tsk_j and an identity (j, v). It outputs a secret key $\mathsf{sk}_{(j,v)}$.
- $\mathsf{Enc}(\mathsf{pp}, m, (j, v)) \to \mathsf{ct}$: The encryption algorithm takes as input the public parameters pp, a message $m \in \mathcal{M}_\lambda$, and an identity (j, v). It outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}_{(j,v)}, \mathsf{ct}) \to m$ or $\perp$: The decryption algorithm takes as input a secret key $\mathsf{sk}_{(j,v)}$ and a ciphertext ct. It outputs either a message $m \in \mathcal{M}_\lambda$ or a special symbol $\perp$.

Correctness. We say a Timed IBE scheme $\mathsf{TIBE} = (\mathsf{Setup}, \ \mathsf{TimeGen}, \ \mathsf{KeyGen}, \ \mathsf{Enc}, \ \mathsf{Dec})$ satisfies correctness if for all $\lambda \in \mathbb{N}$, $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $j \in \mathbb{N}$, $m \in \mathcal{M}_\lambda$, $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$, $\mathsf{sk}_{(j,v)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, (j, v))$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m, (j, v))$ the following holds:

$$\Pr[\mathsf{Dec}(\mathsf{sk}_{(j,v)}, \mathsf{ct}) = m] = 1$$

Definition C.1. We say a timed IBE scheme $\mathsf{TIBE} = (\mathsf{Setup}, \ \mathsf{KeyGen}, \ \mathsf{Enc}, \ \mathsf{Dec})$ is secure if for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\text{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, the adversary's advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$, where advantage of $\mathcal{A}$ is defined as

$$\Pr\left[\mathcal{A}_2(\mathsf{st}_1, \mathsf{ct}) \; = b : \begin{array}{c} (t \in \mathbb{N}, j^* \in [t], (j^*, v^*), S_0, S, \mathsf{st}_0) \leftarrow \mathcal{A}_0(1^\lambda) \\ (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda); \quad b \leftarrow \{0,1\} \\ \mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j) \text{ for } j \in [0, t] \\ \mathsf{sk}_{(0,v)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_0, v) \text{ for } (0, v) \in S_0 \\ \mathsf{sk}_{(j,v)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, v) \text{ for } (j, v) \in S \\ (\mathsf{st}_1, m_0, m_1) = \mathcal{A}_1(\mathsf{pp}, \{\mathsf{sk}_{(0,v)}\}_{(0,v) \in S_0}, \{\mathsf{sk}_{(j,v)}\}_{(j,v) \in S}, \{\mathsf{tsk}_j\}_{j > j^*}, \mathsf{st}_0) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_b, (j^*, v^*)) \end{array}\right] - \frac{1}{2}$$

where the adversary may receive all time secret keys tsk_j for times $j > j^*$, an arbitrary set of identity secret keys for identities $(0, v)$ where the time is 0, and precisely one secret key for identity (j, v) for all positive $j \leq j^*$. We additionally require that the adversary not receive a secret key for identity (j^*, v^*).

C.2 Garbled Circuits with Labels

The construction in [GHRW14a] also uses a Garbled Circuit scheme which allows for specifying labeled outputs. A Garbled Circuit scheme *with output labels* allows the garbling algorithm to accept as input a collection of output labels for the output wires to the circuit being garbled. More formally a scheme GC for circuit classes $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of two polynomial time algorithms ($\mathsf{GC.Garble}$, $\mathsf{GC.Eval}$) with the following syntax:

- $\mathsf{GC.Garble}(1^\lambda, C, \{\mathsf{lab}^{\mathsf{out}}_{j,b}\}_{j \in [m], b \in \{0,1\}}) \to (\widetilde{C}, \{\mathsf{lab}^{\mathsf{in}}_{i,b}\}_{i \in [n], b \in \{0,1\}})$: The garbling algorithm takes as input the security parameter, a circuit $C \in \mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ with n input wires and m output wires, and a collection of output wire labels $\{\mathsf{lab}^{\mathsf{out}}_{j,b}\}_{j \in [m], b \in \{0,1\}}$. It outputs a garbled circuit $\widetilde{C}$ and a collection of input wire labels $\{\mathsf{lab}^{\mathsf{in}}_{i,b}\}_{i \in [n], b \in \{0,1\}}$.
- $\mathsf{GC.Eval}(\widetilde{C}, \{\mathsf{lab}^{\mathsf{in}}_{i,}\}_{i \in [n]}) \to \{\mathsf{lab}^{\mathsf{out}}_{j,}\}_{j \in [m]}$: The evaluation algorithm takes as input a garbled circuit $\widetilde{C}$ and a collection of n input wire labels $\{\mathsf{lab}^{\mathsf{in}}_{i,}\}_{i \in [n]}$. It outputs a sequence of output wire labels $\{\mathsf{lab}^{\mathsf{out}}_{j,}\}_{j \in [m]}$.

For any $j \in [m]$ and $b \in \{0,1\}$, we allow the corresponding output label to be set so that $\mathsf{lab}^{\mathsf{out}}_{j,b} = b$. If for a given j, the labels for both $b \in 0,1$ are set this way, we say the output bit is given *in the clear*.

Correctness. We say a garbled circuit scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \ \mathsf{GC.Eval})$ satisfies correctness if for all $\lambda \in \mathbb{N}$, $C \in \mathcal{C}_\lambda$, $(\widetilde{C}, \{\mathsf{lab}^{\mathsf{in}}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, C, \{\mathsf{lab}^{\mathsf{out}}_{j,b}\}_{j \in [m], b \in \{0,1\}})$, and x, y satisfying $C(x) = y$, the following holds:

$$\Pr[\mathsf{GC.Eval}(\widetilde{C}, \{\mathsf{lab}^{\mathsf{in}}_{i,x_i}\}_{i \in [n]}) = \{\mathsf{lab}^{\mathsf{out}}_{j,y_j}\}_{j \in [m]}] = 1$$

Definition C.2. We say a Garbled Circuit scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \ \mathsf{GC.Eval})$ is secure if there exists a polynomial time simulator algorithm $\mathsf{GC.Sim}$ such that for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a negligible function $\mathrm{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$, the adversary's advantage $\mathsf{Adv}_{\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$, where advantage of $\mathcal{A}$ is defined as

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \left| \Pr\left[\mathcal{A}_1(\mathsf{ct}_b) \; = b : \begin{array}{c} b \leftarrow \{0,1\} \\ (C, x) \leftarrow \mathcal{A}_0(1^\lambda) \\ y := C(x) \\ \mathsf{chal}_0 \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^n, 1^{|C|}, \{\mathsf{lab}^{\mathsf{out}}_{j,y_j}\}_{j \in [|y|]}) \\ (\widetilde{C}, \{\mathsf{lab}_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, C) \\ \mathsf{chal}_1 := (\widetilde{C}, \{\mathsf{lab}_{i,x_i}\}_{i \in [n]}) \end{array}\right] - \frac{1}{2} \right|,$$

where $C \in \mathcal{C}_\lambda$ and $x \in \{0,1\}^n$.

C.3 Predictably Timed Writes

The construction of [GHRW14a], allows for garbling programs and inputs so long as the execution of the programs satisfies a property called *predictably timed writes.* Intuitively this property guarantees that for any location i in memory, before making a read at location i, one can efficiently compute the last time location i was written to. More formally, we require that there exist a polynomial sized circuit $\mathsf{WriteTime}$ (which may depend on the database D, programs P_k, and inputs x_k) such that if state is the jth state variable in the computation and i^{read} is the jth location in memory read during the computation, then $\mathsf{WriteTime}(j, \mathsf{state}, i^{\mathsf{read}}) = u$, where $u < j$ is the largest value such that i^{read} is the location in the database read by the uth step of the program. We now argue that any computation can be transformed so as to satisfy predictably timed writes, with only a small blowup in the size of the database, programs, and runtime. Let $\{P_k\}_k$ and D be the programs and database of the computation. We replace D with a binary tree D' so that the leaves of D' are simply the entries of D. The non-leaf nodes of the tree will contain the most recent time step j for which each of its two immediate children were writen to / updated. These values are all initiated to 0. We modify the programs P_k to new programs P'_k so that at each read for (say) location i, the computation follows the path from the root of D' to the leaf corresponding to i. We modify the step circuits, so that the state information stores the most recent write/update times of the children of the most recently read node. Thus, the most recent write time of any node to be read can be trivially computed from the state information. We also modify them so that whenever P'_k writes to a leaf node, P'_k updates the stored last write times in each node in the path from the root of the tree. Observe that transforming the database and programs in this way increases the size of the database so that $|D'| = O(|D| \log(T))$ where T is the total run time of the programs, since there are $2|D|$ many nodes to store, and each node is of size at most $2\log(T)$. Moreover, the size of each program increases so that $|P'_k| = O(|P_k| \log(|D|) \log(T))$ where the $\log(|D|)$ factor is due to traversing the tree D' and the $\log(|T|)$ factor is due to storing the node information in state. In the construction we describe below, we assume that all computations already satisfy predictably timed writes. This is essentially without loss of generality, as GData and GProg can be modified to perform this transformation on the data and programs respectively before garbling.

C.4 GHRW Garbled RAM Construction

Let TIBE be a secure timed IBE scheme, and let GC be a secure garbled circuit scheme with output labels. Assume that the time and identity secret keys reveal which time or identity they are for. We now formally describe the GHRW Garbled RAM scheme.

$\mathsf{GData}(1^\lambda, D) \to (\widetilde{D}, \mathsf{k}_D)$**:** The algorithm proceeds as follows.

1. The algorithm chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
2. It then chooses $\mathsf{tsk}_0 \leftarrow \mathsf{TimeGen}(\mathsf{msk}, 0)$, and for each $i \in [|D|]$ it chooses

$$\mathsf{sk}_{(0,\mathsf{id}_i)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_0, (0, \mathsf{id}_i))$$

   where for each $i \in [|D|]$, $\mathsf{id}_i = (i, D[i])$.
3. It outputs $\widetilde{D} = \{\mathsf{sk}_{0,\mathsf{id}_i}\}_{i \in [|D|]}$ as the garbled database and $\mathsf{k}_D = (\mathsf{pp}, \mathsf{msk})$ as the program garbling key.

$\mathsf{GProg}(1^\lambda, \mathsf{k}_D, 1^m, P, 1^n, (t_{\mathsf{init}}, t_{\mathsf{fin}})) \to (\widetilde{P}, \{\mathsf{lab}^{\mathsf{in}}_{i,b}\})$**:** The algorithm parses the key as $\mathsf{k}_D = (\mathsf{pp}, \mathsf{msk})$, and parses the program as $P = \{C^P_{\mathsf{RAM}^j}\}_{j \in [t_{\mathsf{init}}, t_{\mathsf{fin}}]}$. We assume the state input (and updated state output) for each step program will be of length n. For each j, the algorithm chooses $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$. For each step circuit $C^P_{\mathsf{RAM}^j}$ we define an augmented step circuit $C^P_{\mathsf{RAM}^{j+}}$. The augmented step circuit will have hard-coded the public paramters pp, the time secret key tsk_j, the labels $\mathsf{lab}_0, \mathsf{lab}_1$, and randomness strings r, r_0, r_1. We define the augmented step circuit

$$C^P_{\mathsf{RAM}^{j+}}[\mathsf{pp}, \mathsf{tsk}_j, \mathsf{lab}_0, \mathsf{lab}_1, r, r_0, r_1](\mathsf{state}, b^{\mathsf{read}})$$

which takes as input an n bit long state state and read bit b^{read}. The circuit $C^P_{\mathsf{RAM}^{j+}}$ computes as follows:

1. The circuit computes
$$(\mathsf{state}', i^{\mathsf{read}}, i^{\mathsf{write}}, b^{\mathsf{write}}) := C^P_{\mathsf{RAM}^j}(\mathsf{state}, b^{\mathsf{read}})$$
2. If $(i^{\mathsf{write}}, b^{\mathsf{write}}) \neq \bot$, the circuit computes
$$\mathsf{sk}_{(j,\mathsf{id})} := \mathsf{KeyGen}(\mathsf{tsk}_j, (j, \mathsf{id}))$$
where $\mathsf{id} = (i^{\mathsf{write}}, b^{\mathsf{write}})$ and sets $\mathsf{sk}^{\mathsf{write}} = \mathsf{sk}_{(j,\mathsf{id})}$. If $(i^{\mathsf{write}}, b^{\mathsf{write}}) = \bot$, the circuit sets $\mathsf{sk}^{\mathsf{write}} = \bot$.
3. Let $u = \mathsf{WriteTime}(j, \mathsf{state}, i^{\mathsf{read}})$, be the largest time $u < j$ such that location i^{read} was written to at time u. For each $b \in \{0, 1\}$, the circuit computes
$$\mathsf{ct}_b := \mathsf{Enc}(\mathsf{pp}, \mathsf{lab}_b, (u, \mathsf{id}_b); r_b)$$
where for each b, $\mathsf{id}_b = (i^{\mathsf{read}}, b)$.
4. The circuit outputs $(\mathsf{state}', i^{\mathsf{read}}, i^{\mathsf{write}}, \mathsf{sk}^{\mathsf{write}}, \mathsf{ct}_0, \mathsf{ct}_1)$

Now, the algorithm garbles the augmented versions of the step circuits $\{C^P_{\mathsf{RAM}^j}\}_{t_{\mathsf{init}} \leq j \leq t_{\mathsf{fin}}}$. Let ρ_1, ρ_2 be the number of random bits used by the algorithms $\mathsf{KeyGen}(\cdot)$ and $\mathsf{Enc}(\cdot)$ respectively. Starting with $j = t_{\mathsf{fin}}$ and counting down to $j = t_{\mathsf{init}}$, the algorithm proceeds as follows.

1. The algorithm computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin}}$, then it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the updated state output (which is simply the output y of the program) is given in the clear, and the labels for the other outputs are set to $\bot$. If $j \neq t_{\mathsf{fin}}$ it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the output labels for the updated state output (i.e. for the first n output wires) match the input labels $\{\mathsf{lab}^i_{b,j+1}\}^{i \in n}_{b \in \{0,1\}}$ for the first n input wires of $C^P_{\mathsf{RAM}^{(j+1)+}}$. The labels for all other outputs are set so that they are given in the clear.
2. The algorithm computes hard-coded inputs. It chooses $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$. It sets $\mathsf{lab}_0 := \mathsf{lab}^{n+1}_{0,j+1}$ and $\mathsf{lab}_1 := \mathsf{lab}^{n+1}_{1,j+1}$ to be the labels for the last input wire of $C^P_{\mathsf{RAM}^{(j+1)+}}$. Finally, it chooses randomness strings $r \leftarrow \{0, 1\}^{\rho_1}$ and for each $b \in \{0, 1\}$ it chooses $r_b \leftarrow \{0, 1\}^{\rho_2}$.
3. The algorithm sets the augmented step circuit as
$$C^P_{\mathsf{RAM}^{j+}} := C^P_{\mathsf{RAM}^{j+}}[\mathsf{pp}, \mathsf{tsk}_j, \mathsf{lab}_0, \mathsf{lab}_1, r, r_0, r_1]$$
and then it garbles the circuit
$$(\widetilde{C^P_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_{b,j}\}^{i \in [n+1]}_{b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, C^P_{\mathsf{RAM}^{j+}}, \mathbf{lab}_{\mathsf{out},j})$$

The algorithm sets $\mathsf{lab}^{\mathsf{in}}_{i,b} := \mathsf{lab}^i_{b,t_{\mathsf{init}}}$ for each $i \in [n]$ and $b \in \{0, 1\}$. It outputs the garbled program $\widetilde{P} := \{\widetilde{C^P_{\mathsf{RAM}^{j+}}}\}_{j \in [t_{\mathsf{init}}, \leq t_{\mathsf{fin}}]}$ and the input label set $\{\mathsf{lab}^{\mathsf{in}}_{i,b}\}^{i \in [n]}_{b \in \{0,1\}}$. Note that, for $j = t_{\mathsf{init}}$, the read bit input is ignored and thus the garbled circuit can be evaluated without the final label.

$\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}, \{\mathsf{lab}^{\mathsf{in}}_{i,}\}_{i \in [n]}) \to \widetilde{y}$**:** The algorithm parses the garbled program as $\widetilde{P} = \{\widetilde{C^P_{\mathsf{RAM}^{j+}}}\}_{j \in [t_{\mathsf{init}}, t_{\mathsf{fin}}]}$. It then evaluates the garbled augmented step circuits in sequence. Counting up, for $j = t_{\mathsf{init}}$ to $j = t_{\mathsf{fin}}$ the algorithm does the following:

1. If $j = t_{\mathsf{init}}$, let $\mathbf{lab}_j = \{\mathsf{lab}^{\mathsf{in}}_{i,}\}$, and otherwise let $\mathbf{lab}_j$ be as computed in the final step of the previous iteration of the loop. If $j = t_{\mathsf{fin}}$, the algorithm computes
$$(\widetilde{y}, \bot) \leftarrow \mathsf{GC.Eval}(\widetilde{C^P_{\mathsf{RAM}^{j+}}}, \mathbf{lab}_j)$$
and then breaks out of the loop. Otherwise it computes
$$(\{\mathsf{lab}^{\mathsf{out}}_{i,j}\}_{i \in [n]}, i^{\mathsf{read}}_j, i^{\mathsf{write}}_j, \mathsf{sk}^{\mathsf{write}}_j, \mathsf{ct}_{0,j}, \mathsf{ct}_{1,j}) \leftarrow \mathsf{GC.Eval}(\widetilde{C^P_{\mathsf{RAM}^{j+}}}, \mathbf{lab}_j)$$

2. Let $\mathsf{sk} \in \widetilde{D}$ be the unique secret key in the garbled database for an identity of the form $(j', (i_j^{\mathsf{write}}, b))$. The algorithm replaces sk with $\mathsf{sk}_j^{\mathsf{write}}$. Note that the contents of $\widetilde{D}$ have now been modified.
3. Now, let $\mathsf{sk} \in \widetilde{D}$ be the unique secret key in the garbled database for an identity of the form $(u, i_j^{\mathsf{read}}, b)$. The algorithm decrypts

$$\mathsf{lab}_{b,j} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{b,j})$$

4. The algorithm sets $\mathbf{lab}_{j+1}$ so that the first n labels match $\{\mathsf{lab}_{i,j}^{\mathsf{out}}\}$ as computed in the first step of the loop and the final label is $\mathsf{lab}_{b,j}$ as computed in the previous step of the loop.

After breaking out of the loop, the algorithm simply outputs $\widetilde{y}$.

We omit proofs of correctness and efficiency, and refer to [GHRW14a] for the proofs.

C.5 Iterated Simulation Security

We now turn to proving that the construction in the previous subsection satisfies our notion of Iterated Simulation Security. We first define the simulator Sim.

$\mathsf{Sim}(1^\lambda, \mathsf{k}_D, 1^n, |P|, y, \{\mathsf{access}_k\}_{k\in\ell}, (t_{\mathsf{init}}, t_{\mathsf{fin}}))$**:** The simulator parses the key as $\mathsf{k}_D = (\mathsf{pp}, \mathsf{msk})$. Given n and $|P|$, the simulator sets s to be the circuit size of each augmented step circuit of P. Let $\mathsf{access} = \{(i_j^{\mathsf{read}}, i_j^{\mathsf{write}}, b_j^{\mathsf{write}})\}_{j\in[t_{\mathsf{init}}, t_{\mathsf{fin}}]}$ be the memory access pattern restricted to the time interval $(t_{\mathsf{init}}, t_{\mathsf{fin}}))$. The simulator computes a sequence of simulated garbled circuits, one for each time $j \in [t_{\mathsf{init}}, t_{\mathsf{fin}}])$. Counting down from $j = t_{\mathsf{fin}}$ to $j = t_{\mathsf{init}}$ the simulator proceeds as follows.

1. The simulator computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin}}$, then $\mathbf{lab}_{\mathsf{out},j}$ is set to be y given out in the clear, with all other wires outputting $\bot$. Otherwise, when $j \neq t_{\mathsf{fin}}$, let $\{\mathsf{lab}_{j+1}^i\}^{i\in[n+1]}$ be the input labels computed in the final step of the previous iteration of the loop. The simulator computes $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$ and $\mathsf{sk}_j^{\mathsf{write}} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, (j, \mathsf{id}))$ where $\mathsf{id} = (i_j^{\mathsf{write}}, b_j^{\mathsf{write}})$. Let u be the largest time $u < j$ such that location i_j^{read} was written to at time u i.e. $i_j^{\mathsf{read}} = i_u^{\mathsf{write}}$, and let $b := b_u^{\mathsf{write}}$ be the bit written at time u. This can be computed from $\{\mathsf{access}_k\}$. The simulator computes $\mathsf{ct}_{b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{lab}_{j+1}^{n+1}, (u, \mathsf{id}_b))$ where $\mathsf{id}_b = (i_j^{\mathsf{read}}, b)$, and $\mathsf{ct}_{1-b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{0}, (u, \mathsf{id}_{1-b})$ where $\mathsf{id}_{1-b} = (i_j^{\mathsf{read}}, 1-b)$. Then the simulator sets $\mathbf{lab}_{\mathsf{out},j}$ so that the first n labels match the labels $\{\mathsf{lab}_{j+1}^i\}^{i\in[n]}$, and the remaining labels are set so that outputs $i_j^{\mathsf{read}}, i_j^{\mathsf{write}}, \mathsf{sk}_j^{\mathsf{write}}, \mathsf{ct}_{0,j}, \mathsf{ct}_{1,j}$ are all given out in the clear.
2. The simulator computes

$$(\widetilde{C^P_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}_j^i\}^{i\in[n+1]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^n, 1^s, \mathbf{lab}_{\mathsf{out},j})$$

The simulator sets $\mathsf{lab}_{i,}^{\mathsf{in}} := \mathsf{lab}_{t_{\mathsf{init}}}^i$ for each $i \in n$. It outputs the simulated program $\widetilde{P} := \{\widetilde{C^P_{\mathsf{RAM}^{j+}}}\}_{j\in[t_{\mathsf{init}}, t_{\mathsf{fin}}]}$ and input labels $\{\mathsf{lab}_{i,}^{\mathsf{in}}\}^{i\in[n]}$.

Before stating and proving our main security claim, we define a sequence of experiments. Fix a positive integer ℓ.

Experiment $\mathbf{a}^{\mathsf{real}}$ for $a \in [\ell]$:

- **Phase 1**: The adversary sends the challenger a database $D \in \{0,1\}^m$, and ℓ tuples $\{(P_k, x_k, n_k, (t_{\mathsf{init},k}, t_{\mathsf{fin},k}))\}_{k\in[\ell]}$. The challenger computes $(y_1, \ldots, y_\ell) \leftarrow (P_1(x_1), \ldots, P_\ell(x_\ell))^D$. For each k the challenger computes access_k to be memory access pattern of the k^{th} program.

- **Data Garbling**: The challenger chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$. It then chooses $\mathsf{tsk}_0 \leftarrow \mathsf{TimeGen}(\mathsf{msk}, 0)$, and for each $i \in m$ it chooses
$$\mathsf{sk}_{(0,\mathsf{id}_i)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_0, (0, \mathsf{id}_i))$$
where for each $i \in [|D|]$, $\mathsf{id}_i = (i, D[i])$. The challenger sets $\widetilde{D} := \{\mathsf{sk}_{0,\mathsf{id}_i}\}_{i\in[|D|]}$ as the garbled database, and $\mathsf{k}_D := (\mathsf{pp}, \mathsf{msk})$ as the program garbling key.

- **Honest Garbling**: For $k > a$, the challenger honestly garbles the k^{th} program P_k. In particular, for each $k \in [a+1, \ell]$, the challenger computes
$$(\widetilde{P_k}, \{\mathsf{lab}^{\mathsf{in}}_{i,b,k}\}^{i\in[n_k]}) \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, m, P_k, 1^{n_k}, (t_{\mathsf{init},k}, t_{\mathsf{fin},k}))$$

- **Program** a: The challenger parses the a^{th} program as $P_a = \{C^{P_a}_{\mathsf{RAM}^j}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$. Let ρ_1, ρ_2 be the number of random bits used by the algorithms $\mathsf{KeyGen}(\cdot)$ and $\mathsf{Enc}(\cdot)$ respectively. Starting with $j = t_{\mathsf{fin},a}$ and counting down to $j = t_{\mathsf{init},a}$, the challenger proceeds as follows.

  1. The challenger computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin},a}$, then it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the updated state output (which is simply the output y_a of the program) is given in the clear, and the labels for the other outputs are set to $\perp$. If $j \neq t_{\mathsf{fin},a}$ it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the output labels for the updated state output (i.e. for the first n output wires) match the input labels $\{\mathsf{lab}^i_{b,j+1}\}^{i\in n_a}_{b\in\{0,1\}}$ for the first n_a input wires of $C^{P_a}_{\mathsf{RAM}^{(j+1)+}}$. The labels for all other outputs are set so that they are given in the clear.
  2. The challenger computes hard-coded inputs. It chooses $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$. It sets $\mathsf{lab}_0 := \mathsf{lab}^{n_a+1}_{0,j+1}$ and $\mathsf{lab}_1 := \mathsf{lab}^{n_a+1}_{1,j+1}$ to be the labels for the last input wire of $C^{P_a}_{\mathsf{RAM}^{(j+1)+}}$. Finally, it chooses randomness strings $r \leftarrow \{0,1\}^{\rho_1}$ and for each $b \in \{0,1\}$ it chooses $r_b \leftarrow \{0,1\}^{\rho_2}$.
  3. The challenger sets the augmented step circuit as
  $$C^{P_a}_{\mathsf{RAM}^{j+}} := C^{P_a}_{\mathsf{RAM}^{j+}}[\mathsf{pp}, \mathsf{tsk}_j, \mathsf{lab}_0, \mathsf{lab}_1, r, r_0, r_1]$$
  and then it garbles the circuit
  $$(\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_{b,j}\}^{i\in[n_a+1]}_{b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, C^{P_a}_{\mathsf{RAM}^{j+}}, \mathbf{lab}_{\mathsf{out},j})$$

  The challenger sets $\widetilde{P_a} := \{\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$. The challenger sets $\mathsf{lab}^{\mathsf{in}}_{i,a} := \mathsf{lab}^i_{x_a[i], t_{\mathsf{init},a}}$ for each $i \in [n_a]$.

- **Simulated Programs**: For each $k < a$, the challenger simulates the k^{th} program P_k. In particular, for each $k \in [a-1]$, the challenger computes
$$(\widetilde{P_k}, \{\mathsf{lab}^{\mathsf{in}}_{i,k}\}^{i\in[n_k]}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{k}_D, 1^{n_k}, |P_k|, y_k, \{\mathsf{access}_{k'}\}_{k'\in[\ell]}, (t_{\mathsf{init},k}, t_{\mathsf{fin},k}))$$

- **Guess Phase**: The challenger sends the adversary $\{(\widetilde{P_k}, \{\mathsf{lab}^{\mathsf{in}}_{i,k}\}^{i\in n_k})\}_{k\in[a]}$ and $\{(\widetilde{P_k}, \{\mathsf{lab}^{\mathsf{in}}_{i,b,k}\}^{i\in[n_k]}_{b\in\{0,1\}})\}_{k\in[a+1,\ell]}$. The adversary sends the challenger a guess bit β.

Experiment $\mathsf{a}^{\mathsf{sim}}$ for $a \in [\ell]$: This experiment is identical to **Experiment $\mathsf{a}^{\mathsf{real}}$**, except for how the challenger processes the a^{th} program. Below we only describe the phase **Program** a, as all other phases are identical.

- **Program** a: Let s be the size of the step circuits of P_a, and let $\mathsf{access} = \{(i^{\mathsf{read}}_j, i^{\mathsf{write}}_j, b^{\mathsf{write}}_j)\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$ be the memory access pattern restricted to the time interval $(t_{\mathsf{init},a}, t_{\mathsf{fin},a})$). The challenger computes a sequence of simulated garbled circuits, one for each time $j \in [t_{\mathsf{init},a}, t_{\mathsf{fin},a}]$). Counting down from $j = t_{\mathsf{fin},a}$ to $j = t_{\mathsf{init},a}$ the challenger proceeds as follows.

1. The challenger computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin},a}$, then $\mathbf{lab}_{\mathsf{out},j}$ is set to be y_a given out in the clear, with all other wires outputting $\perp$. Otherwise, when $j \neq t_{\mathsf{fin},a}$, let $\{\mathsf{lab}^i_{j+1}\}^{i\in[n+1]}$ be the input labels computed in the final step of the previous iteration of the loop. The challenger computes $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$ and $\mathsf{sk}^{\mathsf{write}}_j \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, (j, \mathsf{id}))$ where $\mathsf{id} = (i^{\mathsf{write}}_j, b^{\mathsf{write}}_j)$. Let u be the largest time $u < j$ such that location i^{read}_j was written to at time u i.e. $i^{\mathsf{read}}_j = i^{\mathsf{write}}_u$, and let $b := b^{\mathsf{write}}_u$ be the bit written at time u. This can be computed from $\{\mathsf{access}_k\}$. The challenger computes $\mathsf{ct}_{b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{lab}^{n+1}_{j+1}, (u, \mathsf{id}_b))$ where $\mathsf{id}_b = (i^{\mathsf{read}}_j, b)$, and $\mathsf{ct}_{1-b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{0}, (u, \mathsf{id}_{1-b})$ where $\mathsf{id}_{1-b} = (i^{\mathsf{read}}_j, 1-b)$. Then the challenger sets $\mathbf{lab}_{\mathsf{out},j}$ so that the first n_a labels match the labels $\{\mathsf{lab}^i_{j+1}\}^{i\in[n_a]}$, and the remaining labels are set so that outputs $i^{\mathsf{read}}_j, i^{\mathsf{write}}_j, \mathsf{sk}^{\mathsf{write}}_j, \mathsf{ct}_{0,j}, \mathsf{ct}_{1,j}$ are all given out in the clear.
2. The challenger computes
$$(\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_j\}^{i\in[n_a+1]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{n_a}, 1^s, \mathbf{lab}_{\mathsf{out},j})$$

The challenger sets $\mathsf{lab}^{\mathsf{in}}_{i,a} := \mathsf{lab}^i_{t_{\mathsf{init},a}}$ for each $i \in [n_a]$. The challenger sets $\widetilde{P_a} := \{\widetilde{C^{P}_{\mathsf{RAM}^{j+}}}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$.

For each adversary $\mathcal{A}$ and integer $a \in [\ell]$, we let $\mathsf{Adv}^{\mathcal{A}}_{a,\mathsf{real}}(\lambda)$ be the probability that $\mathcal{A}$ outputs a guess $\beta = 0$ in **Experiment a$^{\mathsf{real}}$**, and $\mathsf{Adv}^{\mathcal{A}}_{a,\mathsf{sim}}(\lambda)$ be the probability that $\mathcal{A}$ outputs a guess $\beta = 0$ in **Experiment a$^{\mathsf{sim}}$**. Now, proving that GRAM satisfies Iterated Simulation Security, is equivalent to proving the following.

Theorem C.3. If TIBE is a secure Timed IBE scheme and GC is a secure garbled circuit scheme, then for any PPT adversary $\mathcal{A}$, integer ℓ, and $a \in [\ell]$, we have that $|\mathsf{Adv}^{\mathcal{A}}_{a,\mathsf{real}}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{a,\mathsf{sim}}(\lambda)| \leq \text{negl}(\lambda)$ for some negligible function $\text{negl}(\cdot)$.

We prove this theorem via a sequence of hybrid experiments. Let $(t_{\mathsf{init},a}, t_{\mathsf{fin},a})$ be the time interval for program P_a. We define a sequence of hybrid experiments.
Experiment a.l.1 for $l \in [t_{\mathsf{init},a}, t_{\mathsf{fin},a}]$: This experiment is identical to **Experiment a$^{\mathsf{real}}$**, except for how the challenger processes the a^{th} program. Below we only describe the phase **Program** a, as all other phases are identical.

- **Program** a: The challenger parses the a^{th} program as $P_a = \{C^{P_a}_{\mathsf{RAM}^j}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$, where the augmented versions of each step circuit has size s. Let ρ_1, ρ_2 be the number of random bits used by the algorithms $\mathsf{KeyGen}(\cdot)$ and $\mathsf{Enc}(\cdot)$ respectively. If $l \neq t_{\mathsf{fin},a}$, starting with $j = t_{\mathsf{fin},a}$ and counting down to $j = l+1$, the challenger proceeds as follows.

  1. The challenger computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin},a}$, then it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the updated state output (which is simply the output y_a of the program) is given in the clear, and the labels for the other outputs are set to $\perp$. If $j \neq t_{\mathsf{fin},a}$ it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the output labels for the updated state output (i.e. for the first n output wires) match the input labels $\{\mathsf{lab}^i_{b,j+1}\}^{i\in n_a}_{b\in\{0,1\}}$ for the first n_a input wires of $C^{P_a}_{\mathsf{RAM}^{(j+1)+}}$. The labels for all other outputs are set so that they are given in the clear.
  2. The challenger computes hard-coded inputs. It chooses $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$. It sets $\mathsf{lab}_0 := \mathsf{lab}^{n_a+1}_{0,j+1}$ and $\mathsf{lab}_1 := \mathsf{lab}^{n_a+1}_{1,j+1}$ to be the labels for the last input wire of $C^{P_a}_{\mathsf{RAM}^{(j+1)+}}$. Finally, it chooses randomness strings $r \leftarrow \{0,1\}^{\rho_1}$ and for each $b \in \{0,1\}$ it chooses $r_b \leftarrow \{0,1\}^{\rho_2}$.
  3. The challenger sets the augmented step circuit as
  $$C^{P_a}_{\mathsf{RAM}^{j+}} := C^{P_a}_{\mathsf{RAM}^{j+}}[\mathsf{pp}, \mathsf{tsk}_j, \mathsf{lab}_0, \mathsf{lab}_1, r, r_0, r_1]$$
  and then it garbles the circuit
  $$(\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_{b,j}\}^{i\in[n_a+1]}_{b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, C^{P_a}_{\mathsf{RAM}^{j+}}, \mathbf{lab}_{\mathsf{out},j})$$

Next, counting down from $j = l$ to $j = t_{\mathsf{init},a}$ the challenger proceeds as follows.

1. The challenger computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin},a} = l$, then $\mathbf{lab}_{\mathsf{out},j}$ is set to be y_a given out in the clear, with all other wires outputting $\perp$. When, $j = l < t_{\mathsf{fin},a}$, let $\{\mathsf{lab}^i_{b,j+1}\}^{i\in[n_a+1]}_{b\in\{0,1\}}$ be the labels computed in the final step of the final iteration of the previous loop. Otherwise, when $j < l$, let $\{\mathsf{lab}^i_{j+1}\}^{i\in[n+1]}$ be the input labels computed in the final step of the previous iteration of this loop. The challenger computes $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$ and $\mathsf{sk}^{\mathsf{write}}_j \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, (j, \mathsf{id}))$ where $\mathsf{id} = (i^{\mathsf{write}}_j, b^{\mathsf{write}}_j)$. Let u be the largest time $u < j$ such that location i^{read}_j was written to at time u i.e. $i^{\mathsf{read}}_j = i^{\mathsf{write}}_u$, and let $b := b^{\mathsf{write}}_u$ be the bit written at time u. This can be computed from $\{\mathsf{access}_k\}$. If $j = l$, for each $b' \in \{0,1\}$, the challenger computes $ct_{j,b'} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{lab}^{n_a+1}_{b',j+1}, (u, \mathsf{id}'_b))$ where $\mathsf{id}'_b = (i^{\mathsf{read}}_j, b')$. If $j < l$, the challenger computes $\mathsf{ct}_{b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{lab}^{n+1}_{j+1}, (u, \mathsf{id}_b))$ where $\mathsf{id}_b = (i^{\mathsf{read}}_j, b)$, and $\mathsf{ct}_{1-b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{0}, (u, \mathsf{id}_{1-b})$ where $\mathsf{id}_{1-b} = (i^{\mathsf{read}}_j, 1-b)$. Then, if $j = l$, the challenger sets $\mathbf{lab}_{\mathsf{out},j}$ so that the first n_a labels match the labels $\{\mathsf{lab}^i_{\mathsf{state}'_j[i],j+1}\}^{i\in[n_a]}$ where state'_j is the updated state that would have been output by the j^{th} step circuit if all ℓ programs and inputs were executed on the database. If $j < l$, the first n_a labels of $\mathbf{lab}_{\mathsf{out},j}$ are set to match the first n_a labels $\{\mathsf{lab}^i_{j+1}\}^{i\in[n_a]}$. In both cases, the remaining labels of $\mathbf{lab}_{\mathsf{out},j}$ are set so that outputs $i^{\mathsf{read}}_j, i^{\mathsf{write}}_j, \mathsf{sk}^{\mathsf{write}}_j, \mathsf{ct}_{0,j}, \mathsf{ct}_{1,j}$ are all given out in the clear.
2. The challenger computes

$$(\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_j\}^{i\in[n_a+1]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{n_a}, 1^s, \mathbf{lab}_{\mathsf{out},j})$$

The challenger sets $\mathsf{lab}^{\mathsf{in}}_{i,a} := \mathsf{lab}^i_{t_{\mathsf{init},a}}$ for each $i \in [n_a]$. The challenger sets $\widetilde{P_a} := \{\widetilde{C^P_{\mathsf{RAM}^{j+}}}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$.

Experiment a.l.2 for $l \in [t_{\mathsf{init},a}, t_{\mathsf{fin},a}]$: This experiment is identical to **Experiment a.1.1**, except for how the challenger processes the a^{th} program, and in particular how it sets the output of the l^{th} circuit. Below we only describe the phase **Program** a, as all other phases are identical.

- **Program** a: The challenger parses the a^{th} program as $P_a = \{C^{P_a}_{\mathsf{RAM}^j}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$, where the augmented versions of each step circuit has size s. Let ρ_1, ρ_2 be the number of random bits used by the algorithms $\mathsf{KeyGen}(\cdot)$ and $\mathsf{Enc}(\cdot)$ respectively. If $l \neq t_{\mathsf{fin},a}$, starting with $j = t_{\mathsf{fin},a}$ and counting down to $j = l + 1$, the challenger proceeds as follows.

1. The challenger computes a set $\mathbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin},a}$, then it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the updated state output (which is simply the output y_a of the program) is given in the clear, and the labels for the other outputs are set to $\perp$. If $j \neq t_{\mathsf{fin},a}$ it sets $\mathbf{lab}_{\mathsf{out},j}$ so that the output labels for the updated state output (i.e. for the first n output wires) match the input labels $\{\mathsf{lab}^i_{b,j+1}\}^{i\in n_a}_{b\in\{0,1\}}$ for the first n_a input wires of $C^{P_a}_{\mathsf{RAM}^{(j+1)+}}$. The labels for all other outputs are set so that they are given in the clear.
2. The challenger computes hard-coded inputs. It chooses $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$. It sets $\mathsf{lab}_0 := \mathsf{lab}^{n_a+1}_{0,j+1}$ and $\mathsf{lab}_1 := \mathsf{lab}^{n_a+1}_{1,j+1}$ to be the labels for the last input wire of $C^{P_a}_{\mathsf{RAM}^{(j+1)+}}$. Finally, it chooses randomness strings $r \leftarrow \{0,1\}^{\rho_1}$ and for each $b \in \{0,1\}$ it chooses $r_b \leftarrow \{0,1\}^{\rho_2}$.
3. The challenger sets the augmented step circuit as

$$C^{P_a}_{\mathsf{RAM}^{j+}} := C^{P_a}_{\mathsf{RAM}^{j+}}[\mathsf{pp}, \mathsf{tsk}_j, \mathsf{lab}_0, \mathsf{lab}_1, r, r_0, r_1]$$

and then it garbles the circuit

$$(\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_{b,j}\}^{i\in[n_a+1]}_{b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, C^{P_a}_{\mathsf{RAM}^{j+}}, \mathbf{lab}_{\mathsf{out},j})$$

Next, counting down from $j = l$ to $j = t_{\mathsf{init},a}$ the challenger proceeds as follows.

1. The challenger computes a set $\textbf{lab}_{\mathsf{out},j}$ of output labels. If $j = t_{\mathsf{fin},a} = l$, then $\textbf{lab}_{\mathsf{out},j}$ is set to be y_a given out in the clear, with all other wires outputting $\perp$. When, $j = l < t_{\mathsf{fin},a}$, let $\{\mathsf{lab}^i_{b,j+1}\}^{i\in[n_a+1]}_{b\in\{0,1\}}$ be the labels computed in the final step of the final iteration of the previous loop. Otherwise, when $j < l$, let $\{\mathsf{lab}^i_{j+1}\}^{i\in[n+1]}$ be the input labels computed in the final step of the previous iteration of this loop. The challenger computes $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$ and $\mathsf{sk}^{\mathsf{write}}_j \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, (j, \mathsf{id}))$ where $\mathsf{id} = (i^{\mathsf{write}}_j, b^{\mathsf{write}}_j)$. Let u be the largest time $u < j$ such that location i^{read}_j was written to at time u i.e. $i^{\mathsf{read}}_j = i^{\mathsf{write}}_u$, and let $b := b^{\mathsf{write}}_u$ be the bit written at time u. This can be computed from $\{\mathsf{access}_k\}$. If $j = l$, the challenger sets $\mathsf{lab}^{n+1}_{j+1} := \mathsf{lab}^{n+1}_{b,j+1}$. Then, the challenger computes $\mathsf{ct}_{b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{lab}^{n+1}_{j+1}, (u, \mathsf{id}_b))$ where $\mathsf{id}_b = (i^{\mathsf{read}}_j, b)$, and $\mathsf{ct}_{1-b,j} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{0}, (u, \mathsf{id}_{1-b})$ where $\mathsf{id}_{1-b} = (i^{\mathsf{read}}_j, 1-b)$. Then, if $j = l$, the challenger sets $\textbf{lab}_{\mathsf{out},j}$ so that the first n_a labels match the labels $\{\mathsf{lab}^i_{\mathsf{state}'_j[i],j+1}\}^{i\in[n_a]}$ where state'_j is the updated state that would have been output by the j^{th} step circuit if all ℓ programs and inputs were executed on the database. If $j < l$, the first n_a labels of $\textbf{lab}_{\mathsf{out},j}$ are set to match the first n_a labels $\{\mathsf{lab}^i_{j+1}\}^{i\in[n_a]}$. In both cases, the remaining labels of $\textbf{lab}_{\mathsf{out},j}$ are set so that outputs $i^{\mathsf{read}}_j, i^{\mathsf{write}}_j, \mathsf{sk}^{\mathsf{write}}_j, \mathsf{ct}_{0,j}, \mathsf{ct}_{1,j}$ are all given out in the clear.
2. The challenger computes

$$(\widetilde{C^{P_a}_{\mathsf{RAM}^{j+}}}, \{\mathsf{lab}^i_j\}^{i\in[n_a+1]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{n_a}, 1^s, \textbf{lab}_{\mathsf{out},j})$$

The challenger sets $\mathsf{lab}^{\mathsf{in}}_{i,a} := \mathsf{lab}^i_{t_{\mathsf{init},a}}$ for each $i \in [n_a]$. The challenger sets $\widetilde{P_a} := \{\widetilde{C^P_{\mathsf{RAM}^{j+}}}\}_{j\in[t_{\mathsf{init},a}, t_{\mathsf{fin},a}]}$.

For each hybrid $a.l.v$, define $\mathsf{Adv}^{\mathcal{A}}_{a.l.v}(\lambda)$ to be the probability the adversary guesses $\beta = 0$ in Experiment $a.l.v$.

Lemma C.4. If TIBE is a secure timed IBE scheme, then for all PPT $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{a.l.1}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{a.l.2}(\lambda)| \leq \text{negl}(\lambda)$.

Proof. Suppose for contradiction that $\mathcal{A}$ is a PPT adversary for which $|\mathsf{Adv}^{\mathcal{A}}_{a.l.1}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{a.l.2}(\lambda)| = \epsilon$, where ϵ is non-negligible. We give a reduction $\mathcal{B}$ which uses $\mathcal{A}$ to break the security of TIBE.

The reduction algorithm $\mathcal{B}$ plays a game with a TIBE challenger. It simulates the view of Experiment $a.l.1$ to $\mathcal{A}$. In particular, it receives database $D \in \{0,1\}^m$, and ℓ tuples $\{(P_k, x_k, n_k, (t_{\mathsf{init},k}, t_{\mathsf{fin},k}))\}_{k\in[\ell]}$. The reduction B then computes $(y_1, \ldots, y_\ell) \leftarrow (P_1(x_1), \ldots, P_\ell(x_\ell))^D$. For each k, $\mathcal{B}$ computes access_k to be memory access pattern of the k^{th} program. Next, as a function of D and the memory access pattern, the reduction computes sets $S_0 = \{(0, (i, D[i]))\ :\ i \in [m]\}$, $S = \{(j, (i^{\mathsf{write}}_j, b^{\mathsf{write}}_j))\ : j \leq l\}$. It then sets (j^*, v^*) to be $(u, (i^{\mathsf{read}}_l, 1 - b^{\mathsf{write}}_u))$, where $u < l$ is the largest time value for which $i^{\mathsf{write}}_u = i^{\mathsf{read}}_l$. It sends $S_0, S, (j^*, v^*), l+1$ to the challenger. The challenger chooses $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda)$, $\mathsf{tsk}_j \leftarrow \mathsf{TimeGen}(\mathsf{msk}, j)$ for each $j \in [t_{\mathsf{fin},\ell}]$, $\mathsf{sk}_{(0,v)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_0, (0, v))$ for each $(0, v) \in S_0$, and $\mathsf{sk}_{(j,v)} \leftarrow \mathsf{KeyGen}(\mathsf{tsk}_j, (j, v))$ for each $(j, v) \in S$. Then, the challenger sends $\mathcal{B}$, the public parameters pp, the secret keys $\{\mathsf{sk}_{(t,v)}\}_{(t,v)\in S_0 \cup S}$, and the time keys $\{\mathsf{tsk}_j\}_{j\in[l+1, t_{\mathsf{fin},\ell}]}$. Now, $\mathcal{B}$ can compute the output of challenger of Experiment $a.l.1$, with the change that wherever a program garbling key k_D was previously used, $\mathcal{B}$ substitutes in the relevant keys which it received from the challenger. The reduction continues in this way until it needs to set $\mathsf{ct}_{1-b^{\mathsf{write}}_u, l}$. The reduction $\mathcal{B}$ sends the challenger the messages $m_0 = \mathsf{lab}^{n_a+1}_{1-b^{\mathsf{write}}_u, l+1}$ and $m_1 = \mathbf{0}$. The challenger chooses $\beta' \leftarrow \{0,1\}$, computes $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp}, m_{\beta'}, (u, (i^{\mathsf{read}}_l, 1 - b^{\mathsf{write}}_u)))$ and sends ct^* to $\mathcal{B}$. Now, $\mathcal{B}$ sets $\mathsf{ct}_{1-b^{\mathsf{write}}_u, l} := \mathsf{ct}^*$, and then proceeds in computing its output as before. Finally, it sends the output to $\mathcal{A}$ which responds with a guess β, and then $\mathcal{B}$ forwards its guess to the challenger. It is straightforward to verify that when $\beta' = 0$, $\mathcal{B}$ perfectly simulates the view of Experiment $a.l.1$, and otherwise it perfectly simulates the view of Experiment $a.l.2$. It follows immediately that $\mathcal{B}$ has non-negligible advantage ϵ in the game with the TIBE challenger, which contradicts the assumption that TIBE is a secure Timed IBE scheme, and we are done. ■

In the following lemma, we rename Experiment a^{real} to Experiment $a.(t_{\mathsf{init},a} - 1).2$.

Lemma C.5. If GC is a secure Garbled Circuit Scheme, then for all $l \in [t_{\mathsf{init},a}, t_{\mathsf{fin},a}]$, and PPT adversary $\mathcal{A}$, $|\mathsf{Adv}^{\mathcal{A}}_{a.l.1}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{a.(l-1).2}(\lambda)| \leq \text{negl}(\lambda)$.

Proof. Suppose for contradiction that $\mathcal{A}$ is a PPT adversary for which $|\mathsf{Adv}^{\mathcal{A}}_{a.l.1}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{a.(l-1).2}(\lambda)| = \epsilon$, where ϵ is non-negligible. We give a reduction $\mathcal{B}$ which uses $\mathcal{A}$ to break the security of GC.

The reduction algorithm $\mathcal{B}$ plays a game with a GC challenger. It simulates the view of Experiment $a.(l-1).2$ to $\mathcal{A}$. In particular, it receives database $D \in \{0,1\}^m$, and ℓ tuples $\{(P_k, x_k, n_k, (t_{\mathsf{init},k}, t_{\mathsf{fin},k}))\}_{k\in[\ell]}$. The reduction B then computes $(y_1, \ldots, y_\ell) \leftarrow (P_1(x_1), \ldots, P_\ell(x_\ell))^D$. For each k, $\mathcal{B}$ computes access_k to be memory access pattern of the k^{th} program. Now, $\mathcal{B}$ can compute the output of challenger of Experiment $a.(l-1).1$. The reduction continues in this way until just before it is to compute $\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}$. Let z_l be the input which $\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}$ would have taken as input. The reduction $\mathcal{B}$ sends the challenger the circuit $\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}$, the output labels $\mathbf{lab}_{\mathsf{out},l}$, and input z_l. The challenger computes $w'_l = \widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}(z_l)$. The challenger chooses $\beta' \leftarrow \{0,1\}$, and if $\beta' = 0$ it computes its challenge $(\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}, \{\mathsf{lab}^i_l\}^{i\in[n_a+1]})$ by honestly garbling $\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}$ and if $\beta' = 1$ it computes its challenge using the simulator. It then sends $(\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}, \{\mathsf{lab}^i_l\}^{i\in[n_a+1]})$ to $\mathcal{B}$. Now, $\mathcal{B}$ proceeds in computing its output as the challenger would in Experiment $a.(l-1).2$, substituting with $(\widetilde{C^{P_a}_{\mathsf{RAM}^{l+}}}, \{\mathsf{lab}^i_l\}^{i\in[n_a+1]})$ where appropriate. Finally, it sends the output to $\mathcal{A}$ which responds with a guess β, and then $\mathcal{B}$ forwards its guess to the challenger. It is straightforward to verify that when $\beta' = 0$, $\mathcal{B}$ perfectly simulates the view of Experiment $a.(l-1).2$, and otherwise it perfectly simulates the view of Experiment $a.l.1$. It follows immediately that $\mathcal{B}$ has non-negligible advantage ϵ in the game with the GC challenger, which contradicts the assumption that GC is a secure garbled circuit scheme, and we are done. ■

Observe that Experiment $a.(t_{\mathsf{fin},a}).2$ is identical to Experiment a^{sim}. It follows then that the previous two lemmas imply Theorem C.3.