Bounded Collusion ABE for TMs from IBE

Rishab Goyal[1*], Ridwan Syed[2], and Brent Waters[2,3†]

[1] MIT, Cambridge, MA
[2] University of Texas at Austin, Austin, TX
[3] NTT Research, Sunnyvale, CA

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 [33]; 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) [58] 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

*Email: goyal@utexas.edu. Work done in part while at UT Austin supported by IBM PhD Fellowship, and at the Simons Institute for the Theory of Computing supported by Simons-Berkeley research 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.

†Email: bwaters@cs.utexas.edu. Supported by NSF CNS-1908611, CNS-1414082, DARPA SafeWare, Packard Foundation Fellowship, and Simons Investigator Award.

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. [43, 12]) 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 [60] 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.

- The ABE construction of Waters [60] 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 [9, 10, 1, 37, 4, 38] 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 [16, 44, 50, 13] arise by constructing ABE for Turing Machines from obfuscation culminating in the work of Ananth and Sahai [8] 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 [8] for a discussion of the tradeoffs present in prior works.
- In a third line of work, Agrawal and Singh [5] gave a construction of a single-key secure functional encryption scheme provably secure under the Learning with Errors (LWE) [56] 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. [34] 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 [15] gave constructions for DFAs from the LWE assumption and Agrawal, Maitra, and Yamada [3] did this for NFAs, but in the secret key setting. Ananth, Fan and Shi [6] 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 [47] 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) [59, 14]. 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 [14, 20, 35, 22].

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 [5].

Our approach leverages a garbled RAM construction due to Gentry, Halevi, Raykova, and Wichs (GHRW) [33] 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 [22, 23, 31, 27] 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 ques-

tion of giving flexibility for when the key query is placed. To do this we use a very relaxed form of non-committing encryption [17, 21, 48] 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 [17].) 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 [11]. The non-committing encryption we consider in this work is very similar to that of receiver non-committing encryption [18].

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., [57, 39, 7]) 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. In the full version [42], we describe how to lift our core construction to general q-query adaptively secure scheme.

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.[4] Now we already know that the concept of garbled circuits [62] 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) [57, 39]. 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

[4] 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.

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.[5] 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[6], 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 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.

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

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

To this end, we replace our usage of garbled circuits with garbled RAMs [53] 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 [40, 22, 27, 26, 2].

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 $\{(q^{\mathsf{in}}, b^{\mathsf{in}}, q^{\mathsf{out}}, b^{\mathsf{out}}, \mathsf{dir})\}$, 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).[7] 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 [53, 33, 54, 32, 29, 28] is of static (full) simulation security wherein an adversary must not be able to

[7] 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.

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).

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 $t+1$.[8] 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 thought 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

[8] 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.

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 fed 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 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 each label 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*.[9] 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. In the full version, we show that the IBE-based garbled RAM construction in [33] is an iterated simulation secure garbled RAM scheme.

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 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.

[9] Although prior works [53, 33, 54, 32, 34, 29, 28, 50, 13] 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.

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) [17, 21, 48] 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.[10]

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 [39] 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

[10] 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. Our notion is similar to that of receiver non-committing encryption [18].

transformation that takes any non-succinct weak NCE scheme and compiles it into a succinct NCE scheme albeit in the Random Oracel Model (ROM) [11]. 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 in the full version. We want to point out that building a succinct weak NCE scheme as described above is impossible in the standard model [55], 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 [39]. For completeness, we provide it in the full version.

Related work, other suggested approaches, and future directions

Comparison with Agrawal-Singh [5]. Closest to us is the work of Agrawal-Singh [5] 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 [36] 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 [39].

ABE via laconic OT. Cho et al. [19] 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 [23, 49] 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 [51], and prove it to be a 1-sided predicate encryption scheme directly without relying on generic transformations [41, 61]. An interesting open question is whether one could extend our current approach to either achieve succinctness similar to [5], 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 [25, 24, 46, 45, 52] 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 [30] 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

Due to space constraints, we describe our notation, the Turing machine and RAM program formalisms, and definitions of secret key encryption and identity-based encryption later in the full version.

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.

$\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) \to m/\perp$. 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 $\perp$.

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* $negl(\cdot)$*, such that for all* $\lambda \in \mathbb{N}$ *the following holds*

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

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.*

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 $\{\mathsf{lab}^{\mathsf{in}}_{i,b}\}_{i\in[n],b\in\{0,1\}}$.

$\mathsf{Eval}^{\widetilde{D}}(\widetilde{P}, \{\mathsf{lab}^{\mathsf{in}}_i\}_{i\in[n]}) \to \widetilde{y}$. The evaluation algorithm takes as input a garbled database $\widetilde{D}$, a garbled program $\widetilde{P}$, and a collection of n labels $\{\mathsf{lab}^{\mathsf{in}}_i\}_{i\in[n]}$. It outputs a value $\widetilde{y}$. As in [33], 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

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

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

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

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

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

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* $negl(\cdot)$*, such that for all* $\lambda \in \mathbb{N}$*, the following holds*

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

where for $0 \leq k \leq \ell$*, 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*

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

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 [33], 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:

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

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 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 $r_{i,b}^{(j)}$ 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 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

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

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 $(\{\mathsf{ct}_{i,b}\}_{(i,b)\in[n+2]\times\{0,1\}}, \perp)$.

Fig. 1: The step program P.

$\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct}) \to m/\perp$. The decryption algorithm parses the ciphertext and secret key as

$$\mathsf{ct} = (w, t, \widetilde{D}, \{\mathsf{lab}_i^{\mathsf{in}}\}_{i\in[n+2]}, \{\widetilde{P}_j\}_{j\in[t+1]}, \{\widetilde{\mathsf{ct}}_{i,b}^{(j)}\}_{i\in[n+2],b\in\{0,1\},j\in[t]}),$$
$$\mathsf{sk}_T = \left(T, \left\{\mathsf{sk}_{(q^{\mathsf{in}},b^{\mathsf{in}},i,\beta)}\right\}_{(q^{\mathsf{in}},b^{\mathsf{in}},i,\beta)\in\mathcal{ID}_T}\right).$$

Let $t' \leq t$ 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 $\widetilde{y_j} \neq \perp$, the algorithm breaks and exits the loop, sets $m := \widetilde{y_j}$, and outputs m.
3. Otherwise, if $\widetilde{y_j} = \perp$ 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 $\perp$.

4.2 Correctness

Due to space constraints, we describe the correctness proof later in the full version.

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 $|w| \leq t$ and that each P has representation of size $\mathsf{poly}(\lambda) \cdot \mathsf{polylog}(\mathsf{t})$. If $\mathsf{IBE.Enc}(\mathsf{pp}, \cdot)$ has runtime $\mathsf{poly}(\lambda)$, $\mathsf{GData}(1^\lambda, D)$ has runtime $|D| \cdot \mathsf{polylog}(|\mathsf{D}|) \cdot \mathsf{poly}(\lambda)$, and $\mathsf{GProg}(1^\lambda, \log(|D|), P, 1^n, (t_{\mathsf{init}}, t_{\mathsf{fin}}))$ has runtime $|P| \cdot \mathsf{polylog}(|\mathsf{D}|, |\mathsf{P}|) \cdot \mathsf{poly}(\lambda)$, then $\mathsf{Enc}(\mathsf{pp}, m, (w, t))$ has runtime $t \cdot \mathsf{polylog}(\mathsf{t}) \cdot \mathsf{poly}(\lambda)$. 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 $|P| \cdot \mathsf{polylog}(|\mathsf{D}|) \cdot \mathsf{poly}(\lambda)$ then $\mathsf{Dec}(\mathsf{sk}_T, \mathsf{ct})$ has runtime $t' \cdot \mathsf{polylog}(\mathsf{t}) \cdot \mathsf{poly}(\lambda)$ 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,* SKE *be a secure symmetric key encryption scheme, 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 $r_{i,b}^{(j)} \leftarrow \{0,1\}^\lambda, \mathsf{K}_{i,b}^{(j)} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.[11]
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 $P_j := P[\mathsf{pp}, \{\mathsf{K}_{i,b}^{(j)}\}_{i,b}, m_\beta, j; \{r_{i,b}^{(j)}\}_{i,b}]$. It computes $2k(n+2)$ IBE ciphertexts as:
$$\begin{aligned}
&(i,b) \in [n+2] \times \{0,1\}, && \mathsf{ct}_{i,b}^{(k)} \leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}_{i,b}^{(k)}, (q_k^{\mathsf{in}}, b_k^{\mathsf{in}}, i, b)),\\
&(i,j) \in [n+2] \times [k-1], && \mathsf{ct}_{i,x_j[i]}^{(j)} \leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathsf{K}_{i,x_j[i]}^{(j)}, (q_j^{\mathsf{in}}, b_j^{\mathsf{in}}, i, x_j[i])),\\
&(i,j) \in [n+2] \times [k-1], && \mathsf{ct}_{i,1-x_j[i]}^{(j)} \leftarrow \mathsf{IBE.Enc}(\mathsf{pp}, \mathbf{0}, (q_j^{\mathsf{in}}, b_j^{\mathsf{in}}, i, 1-x_j[i]))
\end{aligned}$$
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
$$(\widetilde{P_j}, \{\mathsf{lab}_{i,b}^{\mathsf{in},j}\}_{i,b}) \leftarrow \mathsf{GProg}(1^\lambda, \mathsf{k}_D, t+1+\lceil\log(t+1)\rceil, P_j, 1^{n+2}, (1+(j-1)\cdot\ell, j\cdot\ell)).$$
For $j \in [k]$, the challenger computes a simulated program
$$(\widetilde{P_j}, \{\mathsf{lab}_i^{\mathsf{in},j}\}_{i\in[n+2]}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{k}_D, |P|, (\{\mathsf{ct}_{i,b}^{(j)}\}_{(i,b)\in[n+2]\times\{0,1\}}, \perp), D, \{\mathsf{access}'_{j'}\}_{j'\in[t+1]})$$

[11] 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.

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

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

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

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

Game k.2 ($1 \le k \le t+1$). 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 $P_j := P[\mathsf{pp}, \{\mathsf{K}_{i,b}^{(j)}\}_{i,b}, m_\beta, j; \{r_{i,b}^{(j)}\}_{i,b}]$. It computes $2k(n+2)$ IBE ciphertexts as:

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

Game k.3 ($1 \le k \le t+1$). 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:

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

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* $\mathsf{Adv}_{k.1}^{\mathcal{A}}(\lambda) - \mathsf{Adv}_{k.2}^{\mathcal{A}}(\lambda) \leq negl(\lambda)$ *for some negligible function* $negl(\cdot)$*.*

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* $\mathsf{Adv}^{\mathcal{A}}_{k.2}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k.3}(\lambda) \leq negl(\lambda)$ *for some negligible function* $negl(\cdot)$*.*

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:

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

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

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* $\mathsf{Adv}^{\mathcal{A}}_{k.3}(\lambda) - \mathsf{Adv}^{\mathcal{A}}_{k+1.1}(\lambda) \leq negl(\lambda)$ *for some negligible function* $negl(\cdot)$*.*

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.

References

1. Agrawal, S., Chase, M.: Simplifying design and analysis of complex predicate encryption schemes. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017 (2017)
2. Agrawal, S., Maitra, M., Vempati, N.S., Yamada, S.: Functional encryption for turing machines with dynamic bounded collusion from lwe. In: CRYPTO (2021)
3. Agrawal, S., Maitra, M., Yamada, S.: Attribute based encryption (and more) for nondeterministic finite automata from LWE. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019 (2019)
4. Agrawal, S., Maitra, M., Yamada, S.: Attribute based encryption for deterministic finite automata from dlin. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019 (2019)
5. Agrawal, S., Singh, I.P.: Reusable garbled deterministic finite automata from learning with errors. In: Chatzigiannakis, I., Indyk, P., Kuhn, F., Muscholl, A. (eds.) ICALP 2017 (2017)
6. Ananth, P., Fan, X., Shi, E.: Towards attribute-based encryption for rams from LWE: sub-linear decryption, and more. In: ASIACRYPT 2019 (2019)
7. Ananth, P., Vaikuntanathan, V.: Optimal bounded-collusion secure functional encryption. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019 (2019)
8. Ananth, P.V., Sahai, A.: Functional encryption for turing machines. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A (2016)
9. Attrapadung, N.: Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014 (2014)
10. Attrapadung, N.: Dual system encryption framework in prime-order groups via computational pair encodings. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016 (2016)
11. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS (1993)
12. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy. pp. 321–334 (2007)

13. Bitansky, N., Canetti, R., Garg, S., Holmgren, J., Jain, A., Lin, H., Pass, R., Telang, S., Vaikuntanathan, V.: Indistinguishability obfuscation for RAM programs and succinct randomized encodings. SIAM J. Comput. 47(3), 1123–1210 (2018)
14. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: CRYPTO '01 (2001)
15. Boyen, X., Li, Q.: Attribute-based encryption for finite automata from LWE. In: Au, M.H., Miyaji, A. (eds.) ProvSec 2015 (2015)
16. Boyle, E., Chung, K., Pass, R.: On extractability obfuscation. In: TCC 2014 (2014)
17. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: STOC 1996 (1996)
18. Canetti, R., Halevi, S., Katz, J.: Adaptively-secure, non-interactive public-key encryption. In: Theory of Cryptography Conference. pp. 150–168. Springer (2005)
19. Cho, C., Döttling, N., Garg, S., Gupta, D., Miao, P., Polychroniadou, A.: Laconic oblivious transfer and its applications. In: CRYPTO 2017 (2017)
20. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: IMA Int. Conf. pp. 360–363 (2001)
21. Damgård, I., Nielsen, J.B.: Improved non-committing encryption schemes based on a general complexity assumption. In: Annual International Cryptology Conference. pp. 432–450. Springer (2000)
22. Döttling, N., Garg, S.: Identity-based encryption from the diffie-hellman assumption. In: CRYPTO 2017 (2017)
23. Dttling, N., Garg, S.: From selective ibe to full ibe and selective hibe. TCC (2017)
24. Fuchsbauer, G., Jafargholi, Z., Pietrzak, K.: A quasipolynomial reduction for generalized selective decryption on trees. In: Annual Cryptology Conference. pp. 601–620. Springer (2015)
25. Fuchsbauer, G., Konstantinov, M., Pietrzak, K., Rao, V.: Adaptive security of constrained prfs. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 82–101. Springer (2014)
26. Garg, R., Goyal, R., Lu, G., Waters, B.: Dynamic collusion bounded functional encryption from identity-based encryption. Cryptology ePrint Archive, Report 2021/847 (2021), https://ia.cr/2021/847
27. Garg, S., Hajiabadi, M., Mahmoody, M., Rahimi, A.: Registration-based encryption: Removing private-key generator from IBE. In: TCC 2018 (2018)
28. Garg, S., Lu, S., Ostrovsky, R.: Black-box garbled ram. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science. pp. 210–229. IEEE (2015)
29. Garg, S., Lu, S., Ostrovsky, R., Scafuro, A.: Garbled ram from one-way functions. In: STOC (2015)
30. Garg, S., Ostrovsky, R., Srinivasan, A.: Adaptive garbled ram from laconic oblivious transfer. In: Annual International Cryptology Conference. pp. 515–544. Springer (2018)
31. Garg, S., Srinivasan, A.: Garbled protocols and two-round MPC from bilinear maps. In: FOCS 2017 (2017)
32. Gentry, C., Halevi, S., Lu, S., Ostrovsky, R., Raykova, M., Wichs, D.: Garbled ram revisited. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 405–422. Springer (2014)
33. Gentry, C., Halevi, S., Raykova, M., Wichs, D.: Garbled ram revisited, part i. Cryptology ePrint Archive, Report 2014/082 (2014), https://eprint.iacr.org/2014/082
34. Gentry, C., Halevi, S., Raykova, M., Wichs, D.: Outsourcing private ram computation. In: FOCS (2014)

35. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC. pp. 197–206 (2008)
36. Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: STOC (2013)
37. Gong, J., Waters, B., Wee, H.: ABE for DFA from k-lin. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019 (2019)
38. Gong, J., Wee, H.: Adaptively secure ABE for DFA from k-lin and more. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020 (2020)
39. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption with bounded collusions via multi-party computation. In: CRYPTO (2012)
40. Goyal, R., Koppula, V., Waters, B.: Semi-adaptive security and bundling functionalities made generic and easy. In: TCC 2016-B (2016)
41. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. In: FOCS (2017)
42. Goyal, R., Syed, R., Waters, B.: Bounded collusion abe for tms from ibe. Cryptology ePrint Archive, Report 2021/709 (2021), https://ia.cr/2021/709
43. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: CCS (2006)
44. Ishai, Y., Pandey, O., Sahai, A.: Public-coin differing-inputs obfuscation and its applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC (2015)
45. Jafargholi, Z., Kamath, C., Klein, K., Komargodski, I., Pietrzak, K., Wichs, D.: Be adaptive, avoid overcommitting. In: Annual International Cryptology Conference. pp. 133–163. Springer (2017)
46. Jafargholi, Z., Wichs, D.: Adaptive security of yaos garbled circuits. In: Theory of Cryptography Conference. pp. 433–458. Springer (2016)
47. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions (2021)
48. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Annual International Cryptology Conference. pp. 335–354. Springer (2004)
49. Kitagawa, F., Nishimaki, R., Tanaka, K., Yamakawa, T.: Adaptively secure and succinct functional encryption: improving security and efficiency, simultaneously. In: Annual International Cryptology Conference. pp. 521–551. Springer (2019)
50. Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: STOC (2015)
51. Koppula, V., Waters, B.: Realizing chosen ciphertext security generically in attribute-based encryption and predicate encryption. In: CRYPTO 2019 (2019)
52. Kowalczyk, L., Wee, H.: Compact adaptively secure abe for nc1 from k-lin. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–33. Springer (2019)
53. Lu, S., Ostrovsky, R.: How to garble ram programs? In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 719–734. Springer (2013)
54. Lu, S., Ostrovsky, R.: Garbled ram revisited, part ii. Cryptology ePrint Archive, Report 2014/083 (2014), https://eprint.iacr.org/2014/083
55. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Annual International Cryptology Conference (2002)
56. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) STOC (2005)
57. Sahai, A., Seyalioglu, H.: Worry-free encryption: functional encryption with public keys. In: CCS (2010)

58. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT. pp. 457–473 (2005)
59. Shamir, A.: Identity-based cryptosystems and signature schemes. In: CRYPTO 84 (1985)
60. Waters, B.: Functional encryption for regular languages. In: CRYPTO (2012)
61. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under LWE. In: FOCS (2017)
62. Yao, A.: How to generate and exchange secrets. In: FOCS. pp. 162–167 (1986)