Bounded Functional Encryption for Turing Machines: Adaptive Security from General Assumptions

Shweta Agrawal[1], Fuyuki Kitagawa[2], Anuja Modi[1], Ryo Nishimaki[2], Shota Yamada[3], and Takashi Yamakawa[2]

[1] IIT Madras, {shweta.a@cse.iitm.ac.in, anujamodi97@gmail.com}
[2] NTT Corporation, {fuyuki.kitagawa.yh,ryo.nishimaki.zk,takashi.yamakawa.ga}@hco.ntt.co.jp
[3] AIST, yamada-shota@aist.go.jp

Abstract. The recent work of Agrawal et al., [Crypto '21] and Goyal et al. [Eurocrypt '22] concurrently introduced the notion of dynamic bounded collusion security for functional encryption (FE) and showed a construction satisfying the notion from identity based encryption (IBE). Agrawal et al., [Crypto '21] further extended it to FE for Turing machines in non-adaptive simulation setting from the sub-exponential learining with errors assumption (LWE). Concurrently, the work of Goyal et al. [Asiacrypt '21] constructed attribute based encryption (ABE) for Turing machines achieving adaptive indistinguishability based security against bounded (static) collusions from IBE, in the random oracle model. In this work, we significantly improve the state of art for dynamic bounded collusion FE and ABE for Turing machines by achieving *adaptive* simulation style security from a broad class of assumptions, in the standard model. In more detail, we obtain the following results:

1. We construct an adaptively secure (AD-SIM) FE for Turing machines, supporting dynamic bounded collusion, from sub-exponential LWE. This improves the result of Agrawal et al. which achieved only non-adaptive (NA-SIM) security in the dynamic bounded collusion model.
2. Towards achieving the above goal, we construct a *ciphertext policy* FE scheme (CPFE) for circuits of *unbounded* size and depth, which achieves AD-SIM security in the dynamic bounded collusion model from IBE and *laconic oblivious transfer* (LOT). Both IBE and LOT can be instantiated from a large number of mild assumptions such as the computational Diffie-Hellman assumption, the factoring assumption, and polynomial LWE. This improves the construction of Agrawal et al. which could only achieve NA-SIM security for CPFE supporting circuits of unbounded depth from IBE.
3. We construct an AD-SIM secure FE for Turing machines, supporting dynamic bounded collusions, from LOT, ABE for NC^1 (or NC) and private information retrieval (PIR) schemes which satisfy certain properties. This significantly expands the class of assumptions on which AD-SIM secure FE for Turing machines can be based. In particular, it leads to new constructions of FE for Turing machines including one based on polynomial LWE and one based on the combination of the bilinear decisional Diffie-Hellman assumption and the decisional Diffie-Hellman assumption on some specific groups. In contrast the only prior construction by Agrawal et al. achieved only NA-SIM security and relied on *sub-exponential* LWE.
   To achieve the above result, we define the notion of CPFE for read only RAM programs and succinct FE for LOT, which may be of independent interest.
4. We also construct an *ABE* scheme for Turing machines which achieves AD-IND security in the *standard model* supporting dynamic bounded collusions. Our scheme is based on IBE and LOT. Previously, the only known candidate that achieved AD-IND security from IBE by Goyal et al. relied on the random oracle model.

Table of Contents

1 Introduction

Functional encryption (FE) [SW05, BSW11] is a powerful generalization of public key encryption, which goes beyond the traditional "all or nothing" access to encrypted data. In FE, a secret key is associated with a function f, a ciphertext is associated with an input $\mathbf{x}$ and decryption allows to recover $f(\mathbf{x})$. Security intuitively requires that the ciphertext and secret keys do not reveal anything other than the output of the computation. This can be formalized by positing the existence of a simulator which can simulate ciphertexts and secret keys given only the functions f_i and their outputs on the messages $\mathbf{x}_j$, namely $f_i(\mathbf{x}_j)$ for all secret keys sk_{f_i} and ciphertexts $\mathsf{ct}_{\mathbf{x}_j}$ seen by the adversary in the real world. This "simulation style" notion of security, commonly referred to as SIM security, is ruled out by lower bounds in a general security game [BSW11, AGVW13]. However, it can still be achieved in the *bounded* collusion model [GVW12], which restricts the adversary to only request an a-priori bounded number of keys and challenge ciphertexts.

There has been intensive research in the community on FE in the last two decades, studying the feasibility for general classes of functions, from diverse assumptions, satisfying different notions of security. An exciting line of research has focused on FE for uniform models of computation supporting *unbounded* input lengths, such as Deterministic or Non-deterministic Finite Automata, Turing machines and Random Access machines [GTKP+13a, AS16, AS17, AM18, AFS19, GSW21], in contrast to non-uniform models such as circuits. While circuits are expressive, they suffer from two major drawbacks in the context of FE. First, they force the input length to be fixed, a constraint that is inflexible and wasteful in most applications. Second, they necessitate the worst-case running time of the function on every input. By overcoming these limitations, FE schemes can fit demands of real world applications more seamlessly.

In this work, we study FE for Turing machines (henceforth TMFE) in the *bounded collusion* model, namely a security model which restricts the adversary to only request a bounded number of keys. Introduced by Gorbunov, Vaikuntanathan and Wee [GVW12], this model has been popular since i) it is sufficient for multiple interesting real world scenarios, ii) it can support SIM style security, and iii) it can enable constructions from weaker assumptions or for more general functionalities. In the context of TMFE, the very recent work of Agrawal et al. [AMVY21] provided the first construction of bounded TMFE from the (sub-exponential) Learning With Errors assumption (LWE).[4] Furthermore, this work achieved the notion of *dynamic* bounded collusion, where the collusion bound Q does not have to be declared during setup and may be chosen by the encryptor differently for each ciphertext, based on the sensitivity of the encrypted data. Thus, in their construction, the encryptor can choose an input $\mathbf{x}$ of unbounded length, a collusion bound Q and a time bound t, the key generator can choose a machine M of unbounded length and the decryptor runs M on $\mathbf{x}$ for t steps and outputs the result.

The work of Agrawal et al. [AMVY21] takes an important step forward in our understanding of bounded TMFE by providing the first feasibility result in a flexible dynamic model. However, it still leaves several important questions unanswered. For instance, the security notion achieved by TMFE is *non-adaptive* (denoted by NA-SIM) [BSW11] where the adversary must send all the secret key requests before seeing the challenge ciphertext. Moreover, this limitation appears as a byproduct of the security notion achieved by the ingredient sub-schemes used for the construction (more on this below). Additionally, [AMVY21] relies on the heavy machinery of *succinct* single key FE for circuits [GTKP+13b], where succinctness means that the ciphertext size does not depend on the size of circuits supported (but may depend on output length and depth). Succinct FE is known to be constructible only from sub-exponential LWE[5] which necessitates the same assumption to underlie TMFE. This seems unnecessarily restrictive – in contrast, for the circuit model, bounded FE can be constructed from the much milder and more general assumption of public key encryption (PKE) [GVW12, AV19]. This raises the question of whether a strong primitive like succinct FE is really necessary to support the Turing machine model. As detailed below, succinct FE is a crucial tool in the construction, on whose properties the design relies heavily, and it is not clear whether this requirement can be weakened.

For the more limited primitive of *Attribute Based Encryption* (ABE), the recent work of [GSW21] does provide a construction supporting Turing machines in the bounded collusion model (albeit without the dynamic property discussed above), assuming only the primitive of identity based encryption (IBE). Recall that ABE is a restricted class of FE in which the ciphertext is associated with both an input $\mathbf{x}$ and a message m and secret key is associated with a machine M.

[4] Here, sub-exponential (resp., polynomial) LWE refers to the assumption that assumes the distinguishing advantage of the adversary for the decision version of LWE is sub-exponentially (resp., negligibly) small. The modulus to error ratio, which is another important parameter in LWE, will be referred to as approximation factor in this paper.

[5] Aside from obfustopia primitives such as compact FE [AJ15, BV18].

Decryption yields m given a secret key sk_M such that $M(\mathbf{x}) = 1$. Since IBE is a much weaker primitive than succinct FE and can be constructed from several weak assumptions such as the computational Diffie-Hellman assumption (CDH), the factoring assumption (Factoring), LWE and such others, this state of affairs is more satisfying. However, ABE is significantly weaker than FE since it does not hide the data on which the computation actually occurs, and is also an "all or nothing" primitive. Moreover, while their construction achieves strong adaptive security (denoted by AD-IND hereon), their construction relies on the random oracle model, unlike [AMVY21] which is NA-SIM in the standard model.

1.1 Our Results

In this work, we significantly improve the state of the art for dynamic bounded collusion TMFE by achieving adaptive simulation style security from a broad class of assumptions. In more detail, we obtain the following results:

1. We construct an adaptively secure (AD-SIM) TMFE, supporting dynamic bounded collusion, from sub-exponential LWE. This improves the result of [AMVY21] which achieved only NA-SIM security in the dynamic bounded collusion model.
2. Towards achieving the above goal, we construct a *ciphertext policy* FE[6] scheme (CPFE) for circuits of *unbounded* size and depth, which achieves AD-SIM security in the dynamic bounded collusion model from IBE and *laconic oblivious transfer* (LOT). Both IBE and LOT can be instantiated from a large number of mild assumptions such as CDH, Factoring or polynomial LWE. This improves the construction of [AMVY21] which could only achieve NA-SIM security for CPFE supporting circuits of unbounded depth from IBE.
3. We construct an AD-SIM secure TMFE, supporting dynamic bounded collusions, from LOT, ABE for NC^1 (or NC) and private information retrieval (PIR) schemes which satisfy certain properties. This significantly expands the class of assumptions on which AD-SIM secure TMFE can be based since ABE for NC^1 can be constructed from pairing based assumptions like the bilinear decisional Diffie-Hellman assumption (DBDH) [GPSW06] as well as *polynomial* LWE with slightly super-polynomial approximation factors[7] [GVW13, BGG+14], LOT can be based on CDH, Factoring and polynomial LWE [DG17b, DGHM18, BLSV18], and PIR with the required properties can also be based on LWE [BV11], the decisional Diffie-Hellman assumption (DDH), or the quadratic residuosity assumption (QR) [DGI+19]. This leads to new constructions of TMFE as follows:
   - one based on the polynomial hardness of LWE with quasi-polynomial approximation factors,
   - one based on the combination of DBDH and DDH on some specific groups.
   - one based on the combination of DBDH and QR.

   If we instantiate PIR with LWE, we need ABE for NC and LWE with quasi-polynomial approximation factors [GV15] since the answer function of PIR from LWE [BV11, GSW13] is in NC. See Section 5 for the detail. In contrast the only prior construction by [AMVY21] achieved only NA-SIM security and relied on *sub-exponential* LWE. When instantiated with LWE, we observe that the above construction improves the first construction we described. However, we still present the first construction because it is much simpler.
4. We also construct an ABE scheme for Turing machines which achieves AD-IND security in the *standard model* supporting dynamic bounded collusions. Our scheme is based on IBE and LOT. Previously, the only known candidate that achieved AD-IND security from IBE relied on the random oracle model [GSW21].

1.2 Other Related Work

A *key policy* FE[8] (KPFE) for Turing machines supporting only a *single* key request was provided by Agrawal and Singh [AS17] based on sub-exponential LWE. Agrawal, Maitra and Yamada [AMY19] provided a construction of KPFE for non-deterministic finite automata (NFA) which is secure against bounded collusions of arbitrary size. However, this construction is in the symmetric key setting. These constructions do not support the dynamic collusion setting. The first

[6] A secret key and a ciphertext are associated with an input x and a function f, respectively unlike the standard FE.
[7] That is, $O(\lambda^{\omega(1)})$.
[8] This is the same as the standard FE. We use this term to distinguish from CPFE.

Table 1 : Comparison for bounded collusion-resistant FE for uniform models of computation

FE	Class	Security	Model	Assumption
[AS17]	TM	1-key NA-SIM	static	(sub-exp, sub-exp)-LWE†
[AMY19] (SKFE)	NFA	Sel-SIM	static	(sub-exp, sub-exp)-LWE†
[AMVY21]	NL	AD-SIM	dynamic	(sub-exp, sub-exp)-LWE†
[AMVY21]	TM	NA-SIM	dynamic	(sub-exp, sub-exp)-LWE†
Ours §4	TM	AD-SIM	dynamic	(sub-exp, sub-exp)-LWE†
Ours §5	TM	AD-SIM	dynamic	(poly, quasi-poly)-LWE†
Ours §5	TM	AD-SIM	dynamic	DDH‡ & DBDH
Ours §5	TM	AD-SIM	dynamic	QR & DBDH

† For adv $\in$ {poly, sub-exp} and apprx $\in$ {quasi-poly, sub-exp}, (adv, apprx)-LWE means that {polynomial, sub-exponential} hardness of LWE with {quasi-polynomial, sub-exponential} approxiation factors, respectively.
‡ DDH over the multiplicative sub-group of $\mathbb{Z}_q$ where q is a prime.

works to (concurrently) introduce and support the notion of dynamic bounded collusion are [GGLW21] and [AMVY21]. Both works obtain simulation secure KPFE schemes for circuits with dynamic collusion resistance. [AMVY21] additionally obtain succinct CPFE/KPFE schemes for circuits with dynamic collusion property, and also to support Turing machines and NL with different security trade-offs. We provide a comparison for bounded collusion-resistant FE for uniform models of computation in Table 1. All the results in the table are about FE whose encryption time depends on the running-time of computation. There are FE schemes whose encryption time does not depend the running-time of computation [GTKP^{+}13a, AS16, AM18, KNTY19]. However, such constructions are based on strong assumptions such as extractable witness encryption [GTKP^{+}13a] and compact FE [AS16, AM18, KNTY19]. We also omit works based on indistinguishability obfuscation. The focus of the present work is on weak assumptions.

1.3 Our Techniques

In this section, we provide an overview of our techniques. Our final construction is obtained by going through number of steps. We refer to Figure 1 for the overview.

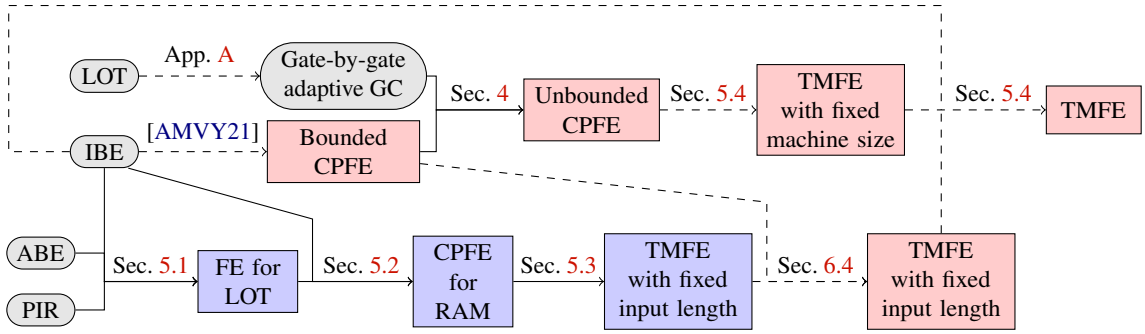

Fig. 1 Illustration of our construction path. Each rectangle represents FE and rounded rectangles represents other primitives. "Bounded CPFE" (resp., "Unbounded CPFE") means CPFE for bounded (resp., unbounded) size circuits. The red (resp., blue) rectangles represent FE with AD-SIM security under bounded dynamic collusion (resp., NA-SIM security under single key collusion). The dashed lines indicate known implications or implications that can be shown by adapting previous techniques relatively easily. The solid lines indicate implications that require new ideas and are shown by us. We do not include (selectively secure) garbled circuits, secret key encryption, and PRF in the figure in order to simplify the presentation.

Recap of TMFE by [AMVY21]: To begin, we recap some of the ideas used in the construction of TMFE provided by [AMVY21]. At a high level, their approach is to separate the cases where the length of the input x and running time bound 1^t is larger than the machine size $|M|$ and one where the opposite is true, i.e. $|(x, 1^t)| \leq |M|$ and $|(x, 1^t)| > |M|$. They observe that running these restricted schemes in parallel allows supporting either case, where the one sub-scheme is used to decrypt a ciphertext if $|(x, 1^t)| \leq |M|$ and the second is used otherwise. We note that such a compiler was first developed by [AMY19] in the symmetric key setting and [AMVY21] uses ideas from [GKW16] to upgrade it to the public key setting.

To construct the restricted sub-schemes, [AMVY21] uses KPFE for the case $|(x, 1^t)| \leq |M|$ and CPFE for $|(x, 1^t)| > |M|$. For concreteness, let us consider the case $|(x, 1^t)| \leq |M|$. Now, using the "delayed encryption" technique of [GKW16] (whose details are not relevant for our purpose), one may assume that there exists an infinite sequence of KPFE instances and the i-th instance supports circuits with input length i. To encrypt a message x with respect to the time bound 1^t, they use the $|(x, 1^t)|$-th instance of KPFE. To generate the secret key for a Turing machine M, they encode M into a set of circuits $C_{i,M}$ for $i = 1, \ldots, |M|$, where $C_{i,M}$ is a circuit that takes as input a string $(x, 1^t)$ of length i, and then runs the machine M for t steps on this input to generate the output. Secret keys for $C_{i,M}$ are generated using the i-th instance of KPFE for all of $i \in [|M|]$. A crucial detail here is that M can be of unbounded size, because each KPFE instance supports unbounded size circuits. Now, decryption is possible when $|(x, 1^t)| \leq |M|$ by using the $|(x, 1^t)|$-th instance. To construct the KPFE scheme, the authors enhance constructions of "succinct KPFE" from the literature [GTKP+13b, Agr17], which can be constructed from (sub-exponential) LWE. The resultant scheme satisfies AD-SIM security against (dynamic) bounded collusions.

To handle the opposite case, namely $|(x, 1^t)| > |M|$, the authors follow the "dual" of the above procedure, using a CPFE scheme in place of KPFE, which supports circuits of unbounded depth (hence size). In more detail, to encrypt a message $(x, 1^t)$, they construct a circuit $\{U_{i,x,t}\}_{i \in [|(x,1^t)|]}$, where $U_{i,x,t}$ is a circuit that takes as input a string M of length i, interprets it as a description of a Turing machine, runs it on input x for t steps and outputs the result. They encrypt the circuit $U_{i,x,t}$ using the i-th instance of CPFE for all of $i \in [|(x, 1^t)|]$. Note that it is necessary that the CPFE scheme support circuits of unbounded *depth* and not just size, since the circuit must run Turing machine for t steps, where t may be arbitrarily large. The authors use IBE to instantiate such a CPFE. However, they can only achieve NA-SIM, where the adversary must make all its key requests before obtaining the challenge ciphertext. This limitation is inherited by the resultant TMFE scheme even though KPFE satisfies AD-SIM security as discussed above.

TMFE with AD-SIM security: As discussed above, the missing piece in constructing TMFE with AD-SIM security from LWE is an instantiation of CPFE supporting unbounded depth circuits with AD-SIM security. We now show how to design this by carefully combining (in a non black-box way) two ingredients – i) an AD-SIM secure CPFE for circuits of *bounded* depth, size and output, denoted by BCPFE, which was constructed in [AMVY21] using IBE, and ii) adaptively secure garbled circuits (GC) based on LOT [GS18]. Our construction makes crucial use of the structural properties of the LOT based adaptive GC constructed by Garg and Srinivasan [GS18]. We describe this next.

Adaptively secure garbled circuits via LOT: Garg and Srinivasan [GS18] provided a construction of adaptively secure GC with near optimal online rate by leveraging the power of LOT. Recall that LOT [CDG+17] is a protocol between two parties: sender and a receiver. The receiver holds a large database $D \in \{0, 1\}^N$ and sends a short digest d (of length λ) of the database to the sender. The sender has as input a location $L \in [N]$ and two messages (m_0, m_1). It computes a read-ciphertext c using its private inputs and the received digest d by running in time $\text{poly}(\log N, |m_0|, |m_1|, \lambda)$ and sends c to the receiver. The receiver recovers the message $m_{D[L]}$ from the ciphertext c and the security requirement is that the message $m_{1-D[L]}$ remains hidden. Updatable LOT additionally allows updates to the database.

The main idea in [GS18] is to "linearize" the garbled circuit, namely to ensure that the simulation of a garbled gate g depends only on simulating one additional gate. With this linearization in place, they designed a careful sequence of hybrids based on a pebbling strategy where the number of changes required in each intermediate hybrid is $O(\log(|C|))$. In more detail, their construction views the circuit C to be garbled as a sequence of step circuits along with a database D, where the i^{th} step circuit implements the i^{th} gate in the circuit. The database D is initialized with the input x and updated to represent the state of the computation as the computation progresses. Thus, at step i, the database contains the output of every gate $g < i$ in the execution of C on x. The i^{th} step circuit reads contents from two pre-determined locations in the database, corresponding to the input wires, and writes a bit, corresponding to the output of the gate, to location i. Thus, they reduce garbling of the circuit to garbling each step circuit along with the database D.

Coming back to our goal of CPFE for unbounded depth circuits, a starting point idea is to use a sequence of bounded size schemes BCPFE to encode a sequence of low depth step circuits described above. Intuitively, we leverage the decomposability of the adaptive GC construction so that a BCPFE scheme can generate each garbled version of the step circuit, using randomness that is derived jointly from the encryptor and key generator via a pseudorandom function (PRF). Specifically, the key generator provides BCPFE keys for input x, along with a PRF input tag, and the encryptor provides BCPFE ciphertexts for the GC step circuits along with a PRF seed. Put together, BCPFE decrypts to provide the inner decomposable GC (generated using jointly computed PRF output), which may then be evaluated to recover $C(x)$. A crucial detail swept under the rug here is how the sequence of GCs interacts with the database which captures the state of the computation. In particular, as the computation proceeds, the database must be updated, and these updates must be taken into account while proceeding with the remainder of the evaluation. This detail is handled via the *updatable* property of our LOT similarly to [GS18]. In the interest of brevity, we do not describe it here, and refer the reader to Section 3 for details.

Assuming the sequence of BCPFE schemes can produce the garbled step circuits and garbled database, we still run into another problem in the security proof – the number of BCPFE ciphertexts is as large as the size of the circuit being encrypted while on the other hand, there is an information-theoretic barrier that the key size of an AD-SIM secure CPFE should grow with the number of challenge ciphertexts [BSW11]. This would bring us right back to where we started as we want to handle unbounded depth circuits and the key generator cannot even know this depth, so we cannot create enough space in the key to support this embedding. Our key observation to overcome this hurdle is that we do not need to simulate all the ciphertexts simultaneously. In particular, by relying on the pebbling-based simulation strategy used in [GS18], we can upper bound the number of ciphertexts in "simulation mode" by a fixed polynomial in each hybrid in the proof. This allows us to embed a simulated GC into the BCPFE secret key which is of fixed size, thereby allowing the post challenge queries required for AD-SIM security. To formalize this idea, we introduce an abstraction which we call "gate-by-gate garbling" (see Section 3), which is similar to locally simulatable garbling introduced by Ananth and Lombardi [AL18]. For more details, please see Sections 3 and 4.

TMFE *without succinct* KPFE *for circuits:* We now describe our construction of TMFE *without* using succinct KPFE for circuits. The high level template for the final construction is the same as discussed earlier, namely, to construct two sub-schemes that handle the cases $|(x, 1^t)| \leq |M|$ and $|(x, 1^t)| > |M|$ separately. Previously, we showed how to construct CPFE with AD-SIM security, for unbounded depth circuits from IBE and LOT, and used this to handle the case $|(x, 1^t)| > |M|$. The counterpart $|(x, 1^t)| \leq |M|$ was handled using KPFE for circuits of unbounded size, which was constructed in [AMVY21] by upgrading the succinct, single key KPFE of Goldwasser et al. [GTKP+13b] from (sub-exponential) LWE. Our goal is to construct FE that can handle the case of $|(x, 1^t)| \leq |M|$ and satisfies AD-SIM security *without* relying on succinct KPFE.

To begin, observe that the generalized bundling technique discussed above lets us focus on the case where $|x, 1^t|$ is fixed, but $|M|$ is unbounded. Moreover, it suffices to restrict ourselves to 1-NA-SIM security, since 1-NA-SIM implies AD-SIM for FE with bounded message length (please see Section 6.4). In [AMVY21], the authors use succinct KPFE to instantiate this scheme by taking advantage of the fact that the size of the circuit associated with the secret key in the succinct KPFE is unbounded. Thus, we can embed a machine M of unbounded size into the secret key. Then we can run the Turing machine inside the circuit for $|M|$ steps, which exceeds t (since we have $t < |M|$) and thus finishes the computation.

Our starting point observation is that since $|(x, 1^t)| \leq |M|$, where $|(x, 1^t)|$ is fixed and $|M|$ is unbounded, we can think of M as a large database, x as a short input and t as bounded running time, which naturally suggest the random access machines (RAM) model of computation. Intuitively, if $|M|$ is massive but $|x|$ and t are small, then running M on x requires only a bounded number of lookups in the transition table and a bounded number of steps, regardless of the size of M. Motivated by this observation, we cast M as a large database and construct a program P which has $(x, 1^t)$ hardwired in it, and executes $M(x)$ via RAM access to M. It is important to note that even if the program does not have M as an input, RAM access to M suffices, because the transition only depends on the description of the current state and the bit that is pointed to by the header. To capture this notion in the setting of FE, we introduce a new primitive, which we call CPFE for (read only) RAM programs. Here, the encryptor encrypts the program $P_{(x,1^t)}$ above, the key generator provides a key for database M and decryption executes $P_{(x,1^t)}$ on M, which is equal to $M(x)$. Crucially, the running time of encryption is required to be independent of $|D|$.

To construct a CPFE for read only RAM, we build upon ideas that were developed in the context of garbled RAM constructions [LO14]. In these constructions, a garbled RAM program consists of t garbled copies of an augmented "step circuit" which takes as input the current CPU state, the last read bit and outputs an updated state and the next read location. Copy i of the CPU step circuit is garbled so that the labels for the output wires corresponding to the output state match the labels of the input wires corresponding to the input state in the next copy $i+1$ of the circuit. The obvious question in this context is how to incorporate data from memory into the computation – clearly, decomposing the computation necessitates some mechanism in which the sequence of garbled circuits communicate with the outside memory[9]. To enable this, previous works have used IBE and oblivious RAM (ORAM) [LO14, GHRW14a]. At a very high level, IBE is used to choose the correct label of the GC as follows – the garbled memory can consist of IBE secret keys for identity (i, b) where i is the given location and b is the bit stored in it, while the garbled circuits can output IBE ciphertexts whose messages are the labels of the next circuit, under identities (i, b). On the other hand, ORAM hides the position read.

However, an immediate hurdle is that ORAM necessitates two parties (client and server) to agree on a secret key. Translated into our setting, this would require that the encryptor and key generator share some secret information – but this is not possible as we are in the public key setting. To overcome this barrier, we introduce the notion of FE for LOT, denoted by LOTFE (Section 5.1). In LOTFE, the encryptor has two messages (μ_0, μ_1) and a database location i. The key generator has a database D as input. Decryption allows to recover $\mu_{D[i]}$ and security hides both $\mu_{1-D[i]}$ as well as the position i. This corresponds to hiding the "other label" as well as the position that was read, in the garbled RAM approach.

It remains to construct LOTFE. Observe that LOTFE must still satisfy the desired succinctness properties, in that a secret key should encode unbounded size data and the running time of the encryption algorithm should be independent of this size. However, by reducing the requisite functionality to something simple like LOT, we earn several benefits over using succinct KPFE. In particular, since we only need to support the table lookup functionality, we can replace the fully homomorphic encryption (FHE) which was used in the succinct KPFE construction [GTKP+13b] with the much weaker *private information retrieval* (PIR). Due to this, we can replace ABE for circuits that is used in the construction of [GTKP+13b] by the much weaker ABE for NC^1 (or NC), which in turn can be constructed by a wider variety of assumptions. We discuss this in more detail below.

Let us briefly recall the main ideas used to construct succinct KPFE. The construction of [GTKP+13b] carefully stitches together ABE for circuits, FHE and GC as follows. At a very high level, FHE is used to encrypt the input x, and this ciphertext $\hat{x}$ is then used as the attribute string for ABE. To encode the circuit f, we construct an ABE secret key for a closely related circuit f', which is used to restrict computation on the FHE ciphertext embedded in the ABE encodings. During decryption, we can check if $f'(\hat{x}) = 1$ and recover the message if so. Intuitively, $f'(\hat{x})$ will represent the bits of an FHE encryption of $f(x)$, denoted by $\widehat{f(x)}$ and provide a message $\mathsf{lbl}_{i,0}$ if the i^{th} bit of $\widehat{f(x)}$ is 0 and $\mathsf{lbl}_{i,1}$ if the i^{th} bit of $\widehat{f(x)}$ is 1. These labels are then used as inputs to the GC which encodes the FHE decryption circuit, so that the decryptor can recover $f(x)$ as desired. Note that the usage of GC implies that the construction can only support a single function key, since otherwise the adversary can recover labels for multiple inputs, violating GC security.

Following a similar template, we can construct 1-NA-SIM secure LOTFE using ABE for NC^1 (or NC), PIR and GC. We encrypt labels under attributes corresponding to the PIR query and provide a key for the PIR answer function to recover the labels corresponding to the PIR answer. These are subsequently fed into the garbled circuit to recover the answer in the clear. We need that the PIR answer function is in NC^1 so that it fits ABE for NC^1. Towards this, we show that PIR from QR or DDH has its answer function in NC^1 and thus can be combined with ABE for NC^1. If we instantiate PIR with an FHE-based scheme [BV11, GSW13], the answer function is in NC and we need ABE for NC, which can be instantiated with LWE with quasi-polynomial approximation factors. Our LOTFE not only allows to use various assumptions other than LWE, which was not possible before, but also allows us to remove the complexity leveraging required for the LWE based construction described before, while achieving AD-SIM secure TMFE at the end[10]. We need Sel-IND secure ABE as a building block to achieve 1-NA-SIM secure LOTFE.

[9] The careful reader may note the similarity with the adaptive garbled circuit construction by [GS18] discussed above.

[10] We observe that the above construction when instantiated with LWE improves the first construction we described. However, we still present the first construction because it is much simpler.

The reason why Sel-IND security suffices for our case is that the reduction algorithm can guess the target attribute the adversary chooses only with polynomial guess. Although there are exponentially many possible PIR queries, the reduction algorithm only has to guess an input that is encoded inside PIR.query. This is because the randomness used for computing the query is not controlled by the adversary, but by the reduction algorithm. Since there are only polynomial number of possible inputs to PIR query function, the guessing can be done only with polynomial security loss. Please see Section 5 for the complete description.

ABE for TM from LOT *and* IBE*:* Lastly, we construct ABE for Turing machines supporting AD-SIM security with dynamic bounded collusions. Our construction relies on LOT and IBE. In contrast to the construction of [GSW21], we do not rely on the random oracle and moreover, we support dynamic bounded collusions.

As before, we consider two cases, namely $|(x, 1^t)| \leq |M|$ and the opposite $|(x, 1^t)| > |M|$. Observe that for the case of longer input, the LOT based CPFE construction discussed above suffices since it implies TMFE where $|(x, 1^t)| > |M|$ with dynamic bounded collusions as discussed above. Therefore we focus on the case of shorter input. For this, our starting point is the *single* key, NA-IND secure (non-adaptively indistinguishable) ABE for TM constructed by the recent work of Goyal et al. [GSW21], which relies on IBE. We upgrade this to adaptively (AD-SIM) secure ABE for TM supporting dynamic bounded collusions of arbitrary size, when $|(x, 1^t)| \leq |M|$, as follows. To begin, we observe that single key NA-IND security in fact implies single key NA-SIM security in the context of ABE (see Remark 2.5 for an argument). Then, we combine the above single key NA-SIM ABE for Turing machines, denoted by 1-TMABE and AD-SIM secure BCPFE (for bounded circuits) with bounded dynamic collusion similarly to [AMVY21, Sec. 4].

In more detail, the master public key and master secret key of the final ABE scheme are those of BCPFE. To encrypt message m under attribute $(x, 1^t)$ for a collusion bound 1^Q, the encryptor first constructs a circuit $\mathsf{1\text{-}TMABE.Enc}(\cdot, x, 1^t, m)$, which is an encryption algorithm of the single-key ABE for TM scheme, that takes as input a master public key and outputs an encryption of the attribute $(x, 1^t)$ and message m under the key. The encryptor then encrypts the circuit using the BCPFE scheme with respect to the bound 1^Q. To generate a secret key for a Turing machine M, the key generator freshly generates a master key pair of 1-TMABE, namely $(\mathsf{1\text{-}TMABE.mpk}, \mathsf{1\text{-}TMABE.msk})$. It then generates a BCPFE secret key BCPFE.sk corresponding to the string 1-TMABE.mpk and an ABE secret key $\mathsf{1\text{-}TMABE.sk}_M$ for the machine M. The final secret key is $(\mathsf{BCPFE.sk}, \mathsf{1\text{-}TMABE.sk}_M)$. Decryption is done by first decrypting the BCPFE ciphertext using the BCPFE secret key to recover $\mathsf{1\text{-}TMABE.Enc}(\mathsf{1\text{-}TMABE.mpk}, x, 1^t, m)$ and then using the secret key $\mathsf{1\text{-}TMABE.sk}_M$ to perform ABE decryption and recover the message m if $M(x) = 1$ within t steps. Security follows from the individual security of the two underlying schemes and yields an AD-SIM secure ABE for TM for an a-priori bounded $|(x, 1^t)|$ with bounded dynamic collusion.

To remove the restriction on $|(x, 1^t)|$, we use the "generalized bundling" trick of [AMVY21, Sec 6.2.2], for the case of $|(x, 1^t)| < |M|$. Thus, we obtain AD-SIM ABE for TM where $|(x, 1^t)| < |M|$ with dynamic bounded collusions. Finally, we combine AD-SIM ABE for TM with $|(x, 1^t)| > |M|$ and one with $|(x, 1^t)| \leq |M|$ as described above. The transformation yields AD-SIM secure ABE for TM with dynamic bounded collusions, which readily implies AD-IND security. Please see Section 6 for further details.

2 Preliminaries

2.1 Universal Circuits

In our construction of FE in Section 4, we use a universal circuit U_C such that $U_C(x)$=$C(x)$ for a circuit C. We need to define U_C in such a way that the topology of U_C does not reveal anything beyond the size of C. To construct such U_C, we first consider a universal circuit U s.t. $U(C, x) = C(x)$. Our U_C first prepares wires whose values take the description of C by a circuit of fixed topology and then input C and x into U. The preparation of the description of C can be done in a bit-wise manner by using the equations $1 = (\neg x) \vee x$ and $0 = (\neg x) \wedge x$.

2.2 Turing Machines

Here, we recall the definition of a Turing machine (TM) following [LL20]. Our definition here is the simplified version by [AMVY21] where we focus on the single tape setting rather than two tapes setting.

Definition 2.1 (Turing Machine). *A (deterministic) TM M is represented by the tuple $M = (Q, \delta, F)$ where Q is the number of states (we use $[Q]$ as the set of states and 1 as the initial state), $F \subset [Q]$ is the set of accepting state and*

$$\begin{aligned} \delta : [Q] \times \{0,1\} &\to [Q] \times \{0,1\} \times \{0, \pm 1\} \\ (q, b) &\mapsto (q', b', \Delta i) \end{aligned}$$

is the state transition function, which, given the current state q and the symbol b on the tape under scan, specifies the new state q', the symbol b', overwriting b, and the direction Δi to which the tape pointers moves. The machine is required to hang (instead of halting) once it reaches an accepting state, i.e., for all $q \in [Q]$ such that $q \in F$ and $b \in \{0,1\}$, it holds that $\delta(q, b) = (q, b, 0)$.

For input length $n \geq 1$ and time complexity bound $t \geq 1$, the set of internal configurations of M is

$$\mathcal{Q}_{M,n,t} = [t] \times \{0,1\}^t \times [Q]$$

where $(i, W, q) \in \mathcal{Q}_{M,n,t}$ specifies the input tape pointer $i \in [t]$, the content of the tape $W \in \{0,1\}^t$ and the machine state $q \in [Q]$.

For any bit-string $x \in \{0,1\}^n$ for $n \geq 1$ and time complexity bound t, the machine M accepts x within time t if there exists a sequence of internal configurations (computation path of t steps) $c_0, \ldots, c_t \in \mathcal{Q}_{M,n,s}$ with $c_k = (i_k, W_k, q_k)$ such that $(i_0, W_0, q_0) = (1, x \| 0^{t-n}, 1)$ (initial configuration),

$$\textit{for all } 0 \leq k < t\textit{:} \quad \begin{cases} \delta(q_k, W_k[i_k]) = (q_{k+1}, W_{k+1}[i_k], i_{k+1} - i_k) \\ W_{k+1}[i] = W_k[i] \quad \textit{for all } i \neq i_k \qquad \textit{(valid transitions)}; \end{cases}$$

where $W_k[i]$ is the i the bit of the string W_k, and $q_t \in F$ (accepting).

2.3 Functional Encryption

Here, we define functional encryption (FE) with dynamic bounded collusion, which is introduced by [AMVY21, GGLW21]. The notion is stronger than conventional bounded collusion FE [GVW12, AV19] in that the collusion bound can be determined by an encryptor dynamically, rather than being determined when the system is setup. We note that some of the definitions here are taken verbatim from [AMVY21].

Let $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\}^*$ be a two-input function where $\mathcal{X}$ and $\mathcal{Y}$ denote "message space" and "key attribute space", respectively. Ideally, we would like to have an FE scheme that handles the relation R directly, where we can encrypt any message $x \in \mathcal{X}$ and can generate a secret key for any key attribute $y \in \mathcal{Y}$. However, in many cases, we are only able to construct a scheme that poses restrictions on the message space and key attribute space. To capture such restrictions, we introduce a parameter prm and consider subsets of the domains $\mathcal{X}_{\mathsf{prm}} \subseteq \mathcal{X}$ and $\mathcal{Y}_{\mathsf{prm}} \subseteq \mathcal{Y}$ specified by it and the function R_{prm} defined by restricting the function R on $\mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}}$. An FE scheme for $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$ is defined by the following PPT algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{prm}) \to (\mathsf{mpk}, \mathsf{msk})$: The setup algorithm takes as input the security parameter λ in unary and a parameter prm that restricts the domain and range of the function and outputs the master public key mpk and a master secret key msk.

$\mathsf{Encrypt}(\mathsf{mpk}, x, 1^Q) \to \mathsf{ct}$: The encryption algorithm takes as input a master public key mpk, a message $x \in \mathcal{X}_{\mathsf{prm}}$, and a bound on the collusion Q in unary. It outputs a ciphertext ct.

$\mathsf{KeyGen}(\mathsf{msk}, y) \to \mathsf{sk}$: The key generation algorithm takes as input the master secret key msk, and a key attribute $y \in \mathcal{Y}_{\mathsf{prm}}$. It outputs a secret key sk. We assume that y is included in sk.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, 1^Q) \to m$ or $\perp$: The decryption algorithm takes as input a ciphertext ct, a secret key sk, and a bound Q associated with the ciphertext. It outputs the message m or $\perp$ which represents that the ciphertext is not in a valid form.

Remark 2.1. We also consider single collusion FE, which is a special case where Q is always fixed to be $Q = 1$. In such a case, we drop 1^Q from the input to the algorithms for simplicity of the notation.

Definition 2.2 (Correctness). *An FE scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is correct if for all* prm, $x \in \mathcal{X}_{\mathsf{prm}}$, $y \in \mathcal{Y}_{\mathsf{prm}}$, *and* $Q \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm}) : \\ \mathsf{Dec}\big(\mathsf{Enc}(\mathsf{mpk}, x, 1^Q), \mathsf{KeyGen}(\mathsf{msk}, y), 1^Q\big) \neq R(x, y)\end{array}\right] = \mathrm{negl}(\lambda)$$

where probability is taken over the random coins of Setup, KeyGen *and* Enc.

We define simulation-based security notions for FE in the following.

Definition 2.3 (AD-SIM **Security for FE with Dynamic Bounded Collusion).** *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a (public key) FE scheme with dynamic bounded collusion for the function family* $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$. *For every stateful PPT adversary* A *and a stateful PPT simulator* $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$, *we consider the experiments in Figure 2.*

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{A}}(1^\lambda)$	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\lambda)$
1. $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$	1. $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$
2. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$	2. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$
3. $(x, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$	3. $(x, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$ – Let $(y^{(1)}, \ldots, y^{(Q_1)})$ be A's oracle queries. – Let $\mathsf{sk}^{(q)}$ be the oracle reply to $y^{(q)}$. – Let $\mathcal{V} := \{(z^{(q)} := R(x, y^{(q)}), y^{(q)}, \mathsf{sk}^{(q)})\}_{q \in [Q_1]}$.
4. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$	4. $(\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}, 1^{\lvert x\rvert}, 1^Q)$
5. $b \leftarrow \mathsf{A}^{\mathcal{O}(\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$	5. $b \leftarrow \mathsf{A}^{\mathcal{O}'(\mathsf{st},\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$
6. Output b	6. Output b

Fig. 2 AD-SIM security for FE

We emphasize that the adversary A *is stateful, even though we do not explicitly include the internal state of it into the output above for the simplicity of the notation. On the other hand, the above explicitly denotes the internal state of the simulator* Sim *by* st. *In the experiments:*

- *The oracle* $\mathcal{O}(\mathsf{msk}, \cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *with* $1 \le Q_1 \le Q$, *and*
- *The oracle* $\mathcal{O}'(\mathsf{st}, \mathsf{msk}, \cdot)$ *takes as input the* q*-th key query* $y^{(q)}$ *for* $q \in [Q_1 + 1, Q_1 + Q_2]$ *and returns* $\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, R(x, y^{(q)}), y^{(q)})$, *where* $Q_1 + Q_2 \le Q$

The FE scheme FE *is then said to be simulation secure for one message against adaptive adversaries (*AD-SIM*-secure, for short) if there is a* PPT *simulator* Sim *such that for every* PPT *adversary* A, *the following holds:*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\lambda) = 1]\right| = \mathrm{negl}(\lambda). \tag{2.1}$$

Remark 2.2 (Non-adaptive security). We can consider a variant of the above security definition where the adversary is not allowed to make a secret key query after the ciphertext ct is given (i.e., $Q_1 = Q$). We call the notion non-adaptive simulation security (NA-SIM). In particular, when we consider single collusion FE, the notion is called 1-NA-SIM. We refer to Section 2.4 for the formal definitions.

Special Classes of FE. We define various kinds of FE by specifying the relation.

CPFE for circuits. To define CPFE for circuits, we set $\mathcal{X}$ to be the set of all circuits and $\mathcal{Y} = \{0,1\}^*$ and define $R(C, x) = C(x)$ if the length of the string x and the input length of C match and otherwise $R(C, x) = \bot$. In this paper, we will consider the circuit class $\mathcal{C}_{\mathsf{inp}}$ that consists of circuits with input length $\mathsf{inp} := \mathsf{inp}(\lambda)$. To do so, we set $\mathsf{prm} = 1^{\mathsf{inp}}$, $\mathcal{X}_{\mathsf{prm}} = \mathcal{C}_{\mathsf{inp}}$, and $\mathcal{Y}_{\mathsf{prm}} = \{0,1\}^{\mathsf{inp}}$.

Remark 2.3. In the definition of CPFE for circuits, even though the input length of the circuits in $\mathcal{C}_{\mathsf{inp}}$ is bounded, the size of the circuits is unbounded.

FE for Turing Machines. To define FE for Turing machines, we set $\mathcal{X} = \{0,1\}^*$, $\mathcal{Y}$ to be set of all Turing machine, and define $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ as

$$R((x,1^t),M) = \begin{cases} 1 & \text{if } M \text{ accepts } x \text{ in } t \text{ steps} \\ 0 & \text{otherwise.} \end{cases}.$$

2.4 Additional Definitions for FE

Here, we define non-adaptive simulation security for FE with dynamic bounded collusion.

Definition 2.4 (NA-SIM **Security for FE with Dynamic Bounded Collusion).** *To define* NA-SIM *security for FE with dynamic bounded collusion, we consider the same experiments as Figure 2 in Definition 2.3 except that the oracles* $\mathcal{O}(\mathsf{msk},\cdot)$ *and* $\mathcal{O}'(\mathsf{msk},\cdot)$ *are both the "empty" oracles that return nothing. The FE scheme* FE *is then said to be simulation secure for one message against non-adaptive queries (*NA-SIM*-secure, for short) if there is* PPT *simulator* $\mathsf{Sim} = (\mathsf{SimEnc}, \perp)$ *such that for every* PPT *adversaries* A*, Eq. (2.1) holds. Note that in the non-adaptive case, we can ignore* st *since* SimKG *is not present in the game and it is never used by other algorithm.*

We then define AD-SIM and NA-SIM security for single collusion FE as well.

Definition 2.5 (1-AD-SIM **and** 1-NA-SIM **Security for single collusion FE [GVW12]).** *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a (public key) single collusion FE scheme for the function family* $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$*. We define* 1-AD-SIM *(resp.,* 1-NA-SIM*) security for* FE *by considering the same game as Definition 2.3 (resp., Definition 2.4) with the following changes:*

- *The adversary can only choose* $Q = 1$*.*
- *The adversary cannot make more than one key query throughout the experiment.*

2.5 Attribute-Based Encryption

Here, we define ABE as a special case of FE. Let us consider a function R whose plaintext space consists of pairs of values from $\mathcal{X} = \mathcal{X}' \times \mathcal{M}$ where $\mathcal{X}'$ and $\mathcal{M}$ are attribute space and message space respectively. We then consider function $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\}^*$ that can be described as

$$R((x',m),y) = \begin{cases} (x',m) & \text{if } R'(x,y) = 1 \\ (x',1^{|m|}) & \text{otherwise.} \end{cases}$$

using another function $R' : \mathcal{X}' \times \mathcal{Y} \to \{0,1\}$. By specifying R', we can consider various types of ABE. Similarly to the case of general FE, we often restricts the attribute space and key space by a parameter prm. In such cases, we consider a function family $\{R'_{\mathsf{prm}} : \mathcal{X}'_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}\}_{\mathsf{prm}}$ where $\mathcal{X}'_{\mathsf{prm}} \subseteq \mathcal{X}$ and $\mathcal{Y}_{\mathsf{prm}} \subseteq \mathcal{Y}$.

2.5.1 Security Notions Since ABE is a special case of FE, we can consider security notions defined for FE for ABE as well. Here, we also define indistinguishability based security notions for ABE below. We first consider the definition with dynamic bounded collusion setting and then for other settings.

Definition 2.6 (AD-IND **Security).** *Let* $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an ABE scheme for the relation family* $\{R'_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}\}_{\mathsf{prm}}$*. For a stateful PPT adversary* A *and a coin* $\beta \in \{0,1\}$ *consider the following experiments:*

1. ***Setup phase:*** *On input* 1^λ, A *submits* prm *to the challenger. The challenger samples* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$ *and replies to* A *with* mpk.
2. ***Key Queries:*** *During the game,* A *adaptively makes secret key queries. When it chooses a key attribute* $y \in \mathcal{Y}_{\mathsf{prm}}$, *the challenger replies with* $\mathsf{sk}_y \leftarrow \mathsf{KeyGen}(\mathsf{msk}, y)$.
3. ***Encryption Query:*** *At some point,* A *can make a single challenge query. When* A *declares its target attribute* $x' \in \mathcal{X}_{\mathsf{prm}}$, *two messages* (m_0, m_1), *and the collusion bound* 1^Q, *the challenger runs* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, (x', m_\beta), 1^Q)$ *and replies with* ct.
4. ***Key Queries:*** *After the challenge query,* A *continues to make secret key queries.*
5. ***Output phase:*** A *outputs a guess bit* β' *as the output of the experiment.*

We say that the adversary is admissible *if* $R(x', y) = 0$ *holds for all the key queries* y *and its encryption query* x' *and* $|m_0| = |m_1|$ *and* A *does not make more than* Q *key queries.* ABE *is said to satisfy adaptive indistinguishability based security against dynamic bounded collusion if for any admissible adversary* A, *there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that*

$$\mathsf{Adv}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) = \left|\Pr\left[\mathsf{Exp}^{(0)}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{(1)}_{\mathsf{ABE},\mathsf{A}}(1^\lambda) = 1\right]\right| \leq \mathrm{negl}(\lambda).$$

Definition 2.7 (1-NA-IND, and Sel-IND Security). *We consider variants of the security notion for* ABE *by changing the game as follows. Depending on the timing of when the challenge query and key queries are allowed, we consider two types of settings.*

- *In the non-adaptive setting (denoted as* NA*), the adversary is not allowed to make a secret key query after the challenge query.*
- *In the selective setting (denoted as* Sel*), the adversary should choose its target* x' *along with* prm *at the beginning of the game (even before seeing* mpk*). However, the choice of messages* (m_0, m_1) *can be still adaptive.*

We can also consider two types of the settings depending on the number of key queries allowed for the adversary.

- *In unbounded collusion setting, we change the syntax of the encryption algorithm so that it does not take* 1^Q *as an additional input. In the security game, the adversary can make any number of secret key queries.*
- *In single collusion setting, the adversary is allowed to make only a single key query.*

Remark 2.4. In the above definition, the encryption query is allowed for only once. However, we can consider multi-ciphertext variant where the adversary can make multiple challenge queries. In the multi-ciphertext variant, we require that $R(x', y) = 0$ holds for all the encryption queries x' and secret key queries y. These definitions are shown to be equivalent by a simple hybrid argument.

Remark 2.5. Here, we note that 1-NA-IND security and 1-NA-SIM security are in fact equivalent. To see this, we show that 1-NA-IND security implies 1-NA-SIM, since the other direction is trivial. To do so, we define the simulator as follows. If the adversary has made a secret key for y such that $R'(x, y) = 1$, the simulation of the ciphertext is trivial. In this case, the simulator is given the message m as an input and it simply encrypts it. Otherwise, the simulator is only given $(x', 1^{|m|})$ and does not know the message. In this case, it encrypts the message $1^{|m|}$ under the attribute x'. The ciphertext is indistinguishable from the real ciphertext by 1-NA-IND security, since the adversary does not have a secret key for y such that $R'(x, y) = 1$. We note that the equivalence of NA-IND security and NA-SIM security can also be shown by similar argument.

2.5.2 Special Classes of ABE

Here, we define several classes of ABE.

ABE for NC^1 circuits. To define ABE for NC^1 circuit, we set $\mathcal{X}' = \{0,1\}^*$ and $\mathcal{Y}$ as the set of all strings of the form $(C, 1^{2^d})$, where C is a circuit and d is its depth. We then define $R'(x, (C, 1^{2^d})) = C(x)$ if the length of the string x and the input length of C matches and otherwise $R'(x, C) = \bot$. Here, the string 1^{2^d} is introduced to reflect the idea that the depth of the circuit is logarithmically bounded. In this paper, we will consider the circuit class $\mathcal{C}_{\mathsf{inp}}$ that consists of circuits with input length $\mathsf{inp} := \mathsf{inp}(\lambda)$. To do so, we set $\mathsf{prm} = 1^{\mathsf{inp}}$, $\mathcal{X}'_{\mathsf{prm}} = \{0,1\}^{\mathsf{inp}}$, and $\mathcal{Y}_{\mathsf{prm}} = \{(C, 1^{2^d}) : C \in \mathcal{C}_{\mathsf{inp}} \text{ and } C\text{'s depth is } d\}$.

Remark 2.6. In the definition of ABE for NC^1 circuits, even though the input length of the circuits in $\mathcal{C}_{\mathsf{inp}}$ is fixed, the depth of the circuits is unbounded. However, since 1^{2^d} is input to the key generation algorithms along with C in our syntax, this effectively limits the input circuit to have logarithmic depth in the security parameter.

ABE for Turing machines. To define ABE for Turing machines, we set $\mathcal{X}' = \{0,1\}^*$, $\mathcal{Y}$ to be set of all Turing machine, and define $R' : \mathcal{X}' \times \mathcal{Y} \to \{0,1\} \cup \{\bot\}$ as

$$R'((x, 1^t), M) = \begin{cases} 1 & \text{if } M \text{ accepts } x \text{ in } t \text{ steps} \\ 0 & \text{otherwise.} \end{cases}.$$

IBE. We define IBE as a special case of ABE where $\mathcal{X}' = \mathcal{Y} = \{0,1\}^*$ and define $R'(x, y) = 1$ if $x = y$ and 0 otherwise.

2.6 Generalized Bundling of Functionality

Consider an FE scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for a parameter $\mathsf{prm} = 1^i$ and a relation $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\} \cup \{\bot\}$ for all $i \in \mathbb{N}$. Using such FE, we will construct a new FE with message space $\mathcal{A}$ and key space $\mathcal{B}$. We assume that there exist efficiently computable maps $\mathcal{S} : \mathbb{N} \to 2^{\mathbb{N}}$ and $\mathcal{T} : \mathbb{N} \to 2^{\mathbb{N}}$ such that $\max \mathcal{S}(n)$ and $\max \mathcal{T}(n)$ can be bounded by some fixed polynomial in n. We also assume that there exist maps f with domain $\mathcal{A}$ and g with domain $\mathcal{B}$ such that

$$f(x) \in \prod_{i \in \mathcal{S}(|x|)} \mathcal{X}_i \quad \text{and} \quad g(y) \in \prod_{i \in \mathcal{T}(|y|)} \mathcal{Y}_i,$$

where $|x|$ and $|y|$ are the lengths of x and y as binary strings. Namely, f and g are maps such that

$$f : \mathcal{A} \ni x \mapsto \{f(x)_i \in \mathcal{X}_i\}_{i \in \mathcal{S}(|x|)}, \quad g : \mathcal{B} \ni y \mapsto \{g(y)_i \in \mathcal{Y}_i\}_{i \in \mathcal{T}(|y|)}.$$

Here, we require that the length of $|f(x)|_i$ and $|g(x)|_i$ can be computed from the length of $|x|$ alone and they do not depend on the actual value of x. In this setting, we can construct an FE scheme $\mathsf{BFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for a two input function $R^{\mathsf{bndl}} : \mathcal{A} \times \mathcal{B} \to \{0,1\}^*$ defined in the following

$$R^{\mathsf{bndl}}(x, y) = \{R_i(f(x)_i, g(y)_i)\}_{i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y|)}, \tag{2.2}$$

where $f(x)_i \in \mathcal{X}_i$ and $g(y)_i \in \mathcal{Y}_i$ are the i-th entries of $f(x)$ and $g(x)$, respectively.

Theorem 2.1 ([AMVY21]). *Suppose* PRF *is a secure pseudorandom function,* GC *is secure garbled circuit scheme, and* IBE *is* IND-CPA *secure identity-based encryption scheme. If* FE *is an* NA-SIM/AD-SIM *secure FE scheme with dynamic bounded collusion for* $\mathsf{prm} = 1^i$ *and* $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\} \cup \{\bot\}$, BFE *is also an* NA-SIM/AD-SIM *secure FE scheme with dynamic bounded collusion for* $R^{\mathsf{bndl}} : \mathcal{A} \times \mathcal{B} \to \{0,1\}^*$*, respectively.*

2.7 Yao's Garbling

Definition 2.8 (Yao's Garbling). *Yao's garbling scheme* GC *is a tuple* $(\mathsf{Garble}, \mathsf{Eval})$ *of two PPT algorithms.*

$\mathsf{Garble}(1^\lambda, C, \{\mathsf{label}_{k,b}\}_{k \in [n], b \in \{0,1\}}) \to \widetilde{C}$*: The garbling algorithm takes as input the security parameter* 1^λ*, a circuit* C *with* n*-bit input, and input labels* $\{\mathsf{label}_{k,b}\}_{k \in [n], b \in \{0,1\}}$ *where* $\mathsf{label}_{k,b} \in \{0,1\}^\lambda$ *for each* $k \in [n]$ *and* $b \in \{0,1\}$ *and outputs a garbled circuit* $\widetilde{C}$*.*

$\mathsf{Eval}(\widetilde{C}, \{\mathsf{label}_k\}_{k \in [n]}) \to y$*: The evaluation algorithm takes as input a garbled circuit* $\widetilde{C}$ *and* n *labels* $\{\mathsf{label}_k\}_{k \in [n]}$ *and outputs* y*.*

Correctness. *We require* $\mathsf{Eval}(\widetilde{C}, \{\mathsf{label}_{k,x_k}\}_{k\in[n]}) = C(x)$ *for every* $\lambda \in \mathbb{N}$*, a circuit* C *with* n*-bit input, and* $x \in \{0,1\}^n$*, where* $\widetilde{C} \leftarrow \mathsf{Garble}(1^\lambda, C, \{\mathsf{label}_{k,b}\}_{k\in[n],b\in\{0,1\}})$ *and* x_k *is the* k*-th bit of* x *for every* $k \in [n]$*.*

Selective security. *There is a PPT algorithm* Sim *such that for any stateful PPT algorithm* A*, we have*

$$\left|\Pr\left[b=1 \;\middle|\; \begin{array}{l}(x, C, \{\mathsf{label}_{k,x_k}\}_{k\in[|x|]}) \leftarrow \mathsf{A}(1^\lambda),\\ \mathsf{label}_{k,1-x_k} \leftarrow \{0,1\}^\lambda \text{ for } k \in [|x|],\\ \widetilde{C} \leftarrow \mathsf{Garble}(1^\lambda, C, \{\mathsf{label}_{k,b}\}_{k,b})\\ b' \leftarrow \mathsf{A}(\widetilde{C}),\\ b := 1 \text{ if } b' = 1 \text{ and } \mathsf{inp}(C) = |x|.\\ \text{Otherwise, } b := 0\end{array}\right] - \Pr\left[b=1 \;\middle|\; \begin{array}{l}(x, C, \{\mathsf{label}_{k,x_k}\}_{k\in[|x|]}) \leftarrow \mathsf{A}(1^\lambda),\\ \widetilde{C} \leftarrow \mathsf{Sim}(1^\lambda, 1^{|C|}, C(x), \{\mathsf{label}_{k,x_k}\}_k)\\ b' \leftarrow \mathsf{A}(\widetilde{C}),\\ b := 1 \text{ if } b' = 1 \text{ and } \mathsf{inp}(C) = |x|.\\ \text{Otherwise, } b := 0\end{array}\right]\right| = \mathrm{negl}(\lambda)$$

where x_k *and* $|x|$ *denote the* k*-th bit of* x *and the length of the string* x*, respectively and* $\mathsf{inp}(C)$ *means input length of* C*.*

Lemma 2.1 ([Yao86]). *If there exists OWF, there exists Yao's garbling scheme.*

2.8 Private Information Retrieval

A single-server private information retrieval (PIR) protocol [KO97] allows a client to query a large database of entries, which is stored in a remote server(s), without leaking the index of the entries of interest.

Definition 2.9 (Private Information Retrieval (PIR)). *A Private Information Retrieval* PIR *is a triple of* PPT *algorithms* $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *with the following syntax:*

- $\mathsf{Query}(1^\lambda, i, N) \to (\mathsf{query}, \mathsf{st})$*: The* Query *algorithm takes as input the security parameter, an index* i*, and the size of a database* N *and outputs a receiver message* query *and a receiver state* st*.*
- $\mathsf{Answer}\left(\mathsf{query}, D, 1^N\right) \to \mathsf{answer}$*: The* Answer *algorithm takes as input a receiver message* query*, a sender input* $D \in \{0,1\}^*$ *and an integer* N *and outputs a sender message* answer*.*
- $\mathsf{Reconstruct}\left(\mathsf{st}, \mathsf{answer}, N\right) \to b$*: The* Reconstruct *algorithm takes as input a receiver state* st*, a sender message* answer *and an integer* N *and outputs a receiver output* $b \in \{0,1\}$*.*

These algorithms satisfy the following properties of correctness and privacy [].

Correctness. *A* PIR *scheme* $(\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ *is* $(1-\epsilon)$*-correct if for any* $D \in \{0,1\}^N$ *and for any* i*,*

$$\Pr\left[\begin{array}{c}(\mathsf{query}, \mathsf{st}) \leftarrow \mathsf{Query}(1^\lambda, i, N),\\ \mathsf{answer} \leftarrow \mathsf{Answer}(\mathsf{query}, D, 1^N) :\\ \mathsf{Reconstruct}(\mathsf{st}, \mathsf{answer}, N) = D[i]\end{array}\right] \geq 1 - \epsilon(\lambda).$$

where the probability is taken over the random tapes of Query, Answer, Reconstruct*. We call* ϵ *the error probability of the PIR scheme. We say that* PIR *is correct if* $\epsilon(\lambda) = \mathrm{negl}(\lambda)$*.*

Privacy. *The standard definition of computational privacy for* PIR *requires that an adversary cannot efficiently distinguish between queries for different indices. Formally, a* PIR *scheme is private if for any probabilistic polynomial-time algorithm* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*, we have*

$$\left|\Pr\left[b' = b \;\middle|\; \begin{array}{l}(1^\lambda, i_0, i_1, \tau) \leftarrow \mathsf{A}_1(1^\lambda, N),\\ b \leftarrow \{0,1\},\\ (\mathsf{query}, \mathsf{st}) \leftarrow \mathsf{Query}(1^\lambda, i_b, N),\\ b' \leftarrow \mathsf{A}_2(N, \mathsf{query}, \tau),\end{array}\right] - \frac{1}{2}\right| = \mathrm{negl}(\lambda)$$

Here τ *denotes the state that* A_1 *passes on to* A_2*.*

Efficiency Requirement. We require that Answer algorithm runs in polynomial time on its input length. In addition, the running time of Query and Reconstruct should be of fixed polynomial in λ for any $N(\lambda) \in [2^{\log^2 \lambda}]$, when we run $\mathsf{Query}(1^\lambda, i, N) \to (\mathsf{query}, \mathsf{st})$, $\mathsf{Answer}\left(\mathsf{query}, D, 1^N\right) \to \mathsf{answer}$, and $\mathsf{Reconstruct}\left(\mathsf{st}, \mathsf{answer}, N\right) \to b$ in a row.[11]

[11] Here, we cannot hope that Answer is efficient in the case where N is super-polynomial because it takes N in unary form as an input. However, Query and Reconstruct can be efficient and we require this.

This in particular means that query, st, and answer computed as above is of fixed polynomial size even for super polynomial N.

2.9 Number Theoretical Assumptions

Decisional Diffie-Hellman Assumption. In the following we state the decisional version of the Diffie-Hellman (DDH) assumption.

Definition 2.10 (Decisional Diffie-Hellman (DDH) Assumption). *A (prime-order) group generator is an algorithm $\mathcal{G}$ that takes as an input a security parameter 1^λ and outputs $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is the description of a multiplicative cyclic group, p is the order of the group which is always a prime number, and g is a generator of the group. We say that $\mathcal{G}$ satisfies the* DDH *assumption (or is* DDH*-hard) if for any* PPT *adversary,* A *it holds that*

$$|\Pr[\mathsf{A}\left((\mathbb{G}, p, g), (g^{a_1}, g^{a_2}, g^{a_1, a_2})\right) = 1] - \Pr[\mathsf{A}\left((\mathbb{G}, p, g), (g^{a_1}, g^{a_2}, g^{a_3})\right) = 1]| = \mathrm{negl}(\lambda)$$

where $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$ and $a_1, a_2, a_3 \leftarrow \mathbb{Z}_p$.

The Quadratic Residuosity Assumption. We say that N is a Blum integer if $N = pq$ for some primes p and q such that $(p \mod 4) = (q \mod 4) = 3$. We denote by $\mathbb{J}_N$ the multiplicative group of the elements in $\mathbb{Z}_N^*$ with Jacobi symbol +1 and by $\mathbb{QR}_N$ the multiplicative group of quadratic residues modulo N with generator g. Note that $\mathbb{QR}_N$ is a subgroups of $\mathbb{J}_N$ and they have order $\frac{\varphi(N)}{4}$ and $\frac{\varphi(N)}{2}$, respectively, where $\varphi(.)$ is Euler's totient function. It is useful to write $\mathbb{J}_N : \mathbb{H} \times \mathbb{QR}_N$, where $\mathbb{H}$ is the multiplicative group $(\pm 1, .)$ of order 2. Note that if N is a Blum integer then $\gcd(2, \frac{\varphi(N)}{4}) = 1$ and $-1 \in \mathbb{J}_N / \mathbb{QR}_N$. We recall he quadratic residuosity (QR) assumption.

Definition 2.11 (Quadratic Residuosity Assumption). *Let $\mathcal{G}$ be a generator that takes as an input a security parameter 1^λ and outputs a Blum integer $N = pq$ along with its factorization p, q and let $\mathbb{QR}_N$ be the multiplicative group of quadratic residues modulo N with generator g. We say the Quadratic Residuosity assumption holds with respect to $\mathcal{G}$ if for any* PPT *adversary* A *it holds that*

$$|\Pr[\mathsf{A}\left(N, g, a\right) = 1] - \Pr[\mathsf{A}\left(N, g, (-1)\, a\right) = 1]| = \mathrm{negl}(\lambda)$$

where $(N, p, q) \leftarrow \mathcal{G}(1^\lambda)$ $a \leftarrow \mathbb{QR}_N$.

The Learning with Errors Assumptions. The learning with errors (LWE) problem was defined by Regev [Reg09]. In this work we introduce the decisional version.

Definition 2.12 (Learning with Errors (LWE) assumption). *The $\mathsf{LWE}_{n,m,q,\chi}$ problem, for $(n, m, q) \in \mathbb{N}$ and for a distribution χ supported over $\mathbb{Z}$, is to distinguish between the distributions $(\mathbf{A}, \mathbf{sA} + \mathbf{e} \mod q)$ and $(\mathbf{A}, \mathbf{u})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$. The $\mathsf{LWE}_{n,m,q,\chi}$ assumption is that the two distributions are computationally indistinguishable. Sub-exponential $\mathsf{LWE}_{n,m,q,\chi}$ assumption assumes that the distinguishing advantage is sub-exponentially small, i.e., $2^{-O(\lambda^c)}$ for some constant $0 < c < 1$ whereas polynomial $\mathsf{LWE}_{n,m,q,\chi}$ assumption assumes that the advantage is $\mathrm{negl}(\lambda)$.*

As shown in previous works (see [PRSD17] and references theirin), the $\mathsf{LWE}_{n,q,\chi}$ problem with χ being the discrete Gaussian distribution with parameter $\sigma = \alpha q \geq 2\sqrt{n}$ is shown to be at least as hard as approximating the shortest independent vector problem (SIVP) to within a factor of $\gamma = \tilde{O}\left(n/\alpha\right)$ in the *worst case* dimension n lattices by quantum/classical reductions. With slight abuse of notation, we call γ an *approximation factor* of LWE. The assumption is conjectured to hold against quantum adversary for sub-exponential approximation factors.

3 Gate-by-Gate Garbling with Pebbling-based Simulation

We define a notion of *gate-by-gate garbling* and its *pebbling-based simulation*. This is an abstraction of the backbone of the adaptive garbling by Garg and Srinivasan [GS18].

First, we define a syntax of a standard garbling scheme.[12] A garbling scheme for a circuit class $\mathcal{C}$ cosists of PPT algorithms $\mathsf{GC} = (\mathsf{GCkt}, \mathsf{GInp}, \mathsf{GEval})$ with the following syntax:

$\mathsf{GCkt}(1^\lambda, C) \to (\widetilde{C}, \mathsf{st})$: The circuit garbling algorithm takes as input the unary representation of the security parameter λ and a circuit $C \in \mathcal{C}$ and outputs a garbled circuit $\widetilde{C}$ and state information st.

$\mathsf{GInp}(\mathsf{st}, x) \to \widetilde{x}$: The input garbling algorithm takes as input the state information st and an input x and outputs a garbled input $\widetilde{x}$.

$\mathsf{GEval}(\widetilde{C}, \widetilde{x}) \to y$: The evaluation algorithm takes as input the garbled circuit $\widetilde{C}$ and garbled input $\widetilde{x}$ and outputs an output y.

Definition 3.1 (Correctness). *A garbling scheme* $\mathsf{GC} = (\mathsf{GCkt}, \mathsf{GInp}, \mathsf{GEval})$ *is correct if for all circuits* $C \in \mathcal{C}$ *and its input* x*,*

$$\Pr[(\widetilde{C}, \mathsf{st}) \leftarrow \mathsf{GCkt}(1^\lambda, C), \widetilde{x} \leftarrow \mathsf{GInp}(\mathsf{st}, x) : \mathsf{GEval}(\widetilde{C}, \widetilde{x}) = C(x)] = 1.$$

In addition to the security notion for a standard garbling scheme in Definition 2.8, we introduce a new security notion specific to gate-by-gate garbling, which we call *pebbling-based security*. For defining gate-by-gate garbling, we prepare some notations about circuits.

Notations. For a circuit C, we denote by Gates the set of all gates of C. We use inp and out to mean the input-length and output-length of C, respectively. A unique index from 1 to N is assigned to each bit of input and gate where N is the sum of the input-length and the number of gates of C. In particular, each bit of input is assigned by indices from 1 to inp, each intermediate gate is assigned by indices from $\mathsf{inp}+1$ to $N-\mathsf{out}$, and each output gate is assigned by indices from $N-\mathsf{out}+1$ to N. We assume that a circuit has fan-in 2 and unbounded fan-out without loss of generality. A gate $\mathsf{G} \in \mathsf{Gates}$ is represented as $(g_\mathsf{G}, i_\mathsf{G}, A_\mathsf{G}, B_\mathsf{G})$ where $g_\mathsf{G} : \{0,1\} \times \{0,1\} \to \{0,1\}$ is a function corresponding to G, i_G is the index of G, and A_G and B_G are indices of the input bits or gates whose values are passed to G as input. We assume $A_\mathsf{G} < i_\mathsf{G}$ and $B_\mathsf{G} < i_\mathsf{G}$ without loss of generality. We call $(i_\mathsf{G}, A_\mathsf{G}, B_\mathsf{G})$ the *topology* of G and denote it by $\mathsf{top}(\mathsf{G})$. We call the set of topology of all gates the topology of C and denote it by $\mathsf{top}(C)$.

Intuitively, gate-by-gate garbling is a garbling scheme whose circuit garbling algorithm can be decomposed into gate garbling algorithms for each gate, whose efficiency is independent of the size of the circuit. The formal definition is given below:

Definition 3.2 (Gate-by-Gate Garbling:). *A garbling scheme* $\mathsf{GC} = (\mathsf{GCkt}, \mathsf{GInp}, \mathsf{GEval})$ *for a circuit class* $\mathcal{C}$ *is said to be gate-by-gate garbling with* $N_\mathsf{rand} = N_\mathsf{rand}(\mathsf{top}(C))$ *randomness slots and randomness length* ℓ *if* GCkt *can be decomposed into PPT sub-algorithms* $(\mathsf{GSetup}, \mathsf{GGate})$ *as follows:*

$\mathsf{GCkt}(1^\lambda, C)$*: The circuit garbling algorithm proceeds as follows:*

1. *Run* $\mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$ *to generate a public parameter* pp*.*
2. *For* $i \in [N_\mathsf{rand}]$*, generate a randomness* $\mathsf{R}_i \leftarrow \{0,1\}^\ell$ *for* $\ell = \mathrm{poly}(\lambda)$ *that does not depend on* C*.*
3. *For* $\mathsf{G} \in \mathsf{Gates}$*, run* $\mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i \in S(\mathsf{G})})$ *to generate a garbled gate* $\widetilde{\mathsf{G}}$*. Here,* $S(\mathsf{G}) \subseteq [N_\mathsf{rand}]$ *is a subset of size* $O(1)$ *that is efficiently computable from* G *and* $\mathsf{top}(C)$*.*
4. *Output a garbled circuit* $\widetilde{C} := \left(\mathsf{pp}, \{\widetilde{\mathsf{G}}\}_{\mathsf{G} \in \mathsf{Gates}}\right)$ *and the state information* $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i \in S_\mathsf{st}})$ *where* $S_\mathsf{st} \subseteq [N_\mathsf{rand}]$ *is a subset of size* $O(\mathsf{inp}+\mathsf{out})$ *that is efficiently computable from* $\mathsf{top}(C)$*.*

We require GC *to satisfy the following requirements.*

1. $\mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i \in S(\mathsf{G})})$ *runs in time* $\mathrm{poly}(\lambda)$ *independently of the size of* C *where* $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$ *and* $\mathsf{R}_i \leftarrow \{0,1\}^\ell$ *for* $i \in [N_\mathsf{rand}]$*.*
2. $\mathsf{GInp}(\mathsf{st}, x)$ *is deterministic and runs in time* $\mathrm{poly}(\lambda, \mathsf{inp}, \mathsf{out})$ *independently of the size of* C *where* $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$*,* $\mathsf{R}_i \leftarrow \{0,1\}^\ell$ *for* $i \in [N_\mathsf{rand}]$*, and* $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i \in S_\mathsf{st}})$*.*

[12] Note that the syntax defined here is more general than that of Yao's garbling defined in Definition 2.8.

We require gate-by-gate garbling to satisfy (M, T)-*pebbling-based security* for some parameters M and T, which intuitively requires the following: There are three modes of gate garbling algorithms, the white mode, black mode, and gray mode. The white mode gate garbling algorithm is identical to the real gate garbling algorithm. The black mode gate garbling algorithm is a "simulation" algorithm that simulates a garbled gate without knowing the functionality of the gate. The gray mode garbling algorithm is an "input-dependent simulation" algorithm that simulates a garbled gate by using both the circuit C and its input x that are being garbled. We call a sequence of modes of each gate of C a configuration of C. There is a configuration-based input-garbling algorithm that simulates a garbled input by using C and a configuration as additional inputs. When the configuration is all-white, it corresponds to the real input garbling algorithm, and when the configuration is all-black, it corresponds to a legitimate "simulation" algorithm that only uses $C(x)$ instead of C and x. The security requires that there is a sequence of configurations of length T that starts from the all-white configuration, which corresponds to the real garbling algorithm, to the all-black configuration, which corresponds to the simulation algorithm, such that:

1. the number of gates in the gray mode in any intermediate configuration is at most M, and
2. garbled circuits and garbled inputs generated in neighboring configurations are computationally indistinguishable even if the distinguisher can specify all the randomness needed for generating garbled gates whose modes are black or white in both of those configurations.

3.1 Definition of Pebbling-based Security

We give a formal definition of pebbling-based security for gate-by-gate garbling. For defining this, we prepare the following definition.

Definition 3.3 (Circuit Configuration). *For a circuit C, a circuit configuration* $\mathsf{conf} = \{\mathsf{mode}(\mathsf{G})\}_{\mathsf{G}\in\mathsf{Gates}}$ *is a sequence of* $\mathsf{mode}(\mathsf{G}) \in \{\mathsf{White}, \mathsf{Gray}, \mathsf{Black}\}$ *for* $\mathsf{G} \in \mathsf{Gates}$.

The pebbling-based simulation security is defined below.

Definition 3.4 (Pebbling-based Simulator). *Let* $\mathsf{GC} = (\mathsf{GCkt}, \mathsf{GInp}, \mathsf{GEval})$ *be a gate-by-gate garbling scheme for a circuit class $\mathcal{C}$. Let* $(\mathsf{GSetup}, \mathsf{GGate})$ *be the associated sub-algorithms as defined in Definition 3.2. A pebbling-based simulator for* GC *consists of PPT algorithms* $\mathsf{Sim}_{\mathsf{GC}} = (\{\mathsf{SimGate}_{\mathsf{mode}}\}_{\mathsf{mode}\in\{\mathsf{White},\mathsf{Gray},\mathsf{Black}\}}, \mathsf{SimInpConf})$ *with the following syntax.*

$\mathsf{SimGate}_{\mathsf{White}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{White}}(\mathsf{G})})$*: This must be the same algorithm as* $\mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S(\mathsf{G})})$*. In particular, we must have* $S_{\mathsf{White}}(\mathsf{G}) = S(\mathsf{G})$.

$\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})})$*: The black mode gate simulation algorithm takes as input the public parameter* pp*, a gate* G*, and a tuple of randomness* $\{\mathsf{R}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}$ *and outputs a simulated garbled gate* $\widetilde{\mathsf{G}}$ *where* $S_{\mathsf{Black}}(\mathsf{G}) \subseteq [N_{\mathsf{rand}}]$ *is a subset of size* $O(1)$ *that is efficiently computable from* G *and* $\mathsf{top}(C)$.

$\mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in[N_{\mathsf{rand}}]}, C, x)$*: The gray mode gate simulation algorithm takes as input the public parameter* pp*, a gate* G*, a tuple of randomness* $\{\mathsf{R}_i\}_{i\in[N_{\mathsf{rand}}]}$*, a circuit C, and input x and outputs a simulated garbled gate* $\widetilde{\mathsf{G}}$*. Remark that this algorithm additionally takes* R_i *for all* $i \in [N_{\mathsf{rand}}]$ *and C and x as input differently from white and black modes.*

$\mathsf{SimInpConf}(\mathsf{st}, x, y, \mathsf{conf})$*: The input simulation algorithm is a deterministic algorithm that takes as input the state information* st*, an input x, an output y, and a configuration* conf *and outputs a simulated garbled input* $\widetilde{x}$*. Remark that this algorithm additionally takes as input y and* conf *differently from the real input garbling algorithm* GInp.

We require them to satisfy the following properties.

1. $\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})})$ *runs in time* $\mathrm{poly}(\lambda)$ *independently of the size of C where* $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$ *and* $\mathsf{R}_i \leftarrow \{0,1\}^\ell$ *for* $i \in [N_{\mathsf{rand}}]$*. Note that the similar property for* $\mathsf{SimGate}_{\mathsf{White}}$ *directly follows from* $\mathsf{SimGate}_{\mathsf{White}} = \mathsf{GGate}$ *and the efficiency requirement for* GGate *in Definition 3.2.*

2. *When the circuit configuration is* $\mathsf{conf} = \mathsf{White}^{|\mathsf{Gates}|}$*, a garbled input simulated with* conf *are generated similarly to the real one.*[13] *That is,* $\mathsf{SimInpConf}\left(\mathsf{st}, x, y, \mathsf{White}^{|\mathsf{Gates}|}\right)$ *does not depend on* y *and is equivalent to* $\mathsf{GInp}(\mathsf{st}, x)$.
3. *When the circuit configuration is* $\mathsf{conf} = \mathsf{Black}^{|\mathsf{Gates}|}$*, a garbled circuit and input simulated with* conf *can be simulated from* $C(x)$ *and* $\mathsf{top}(C)$ *without using* C *or* x*. Formally, the following is satisfied:*
   (a) *For each* $\mathsf{G} \in \mathsf{Gates}$*,* $S_{\mathsf{Black}}(\mathsf{G})$ *is efficiently computable from the index* i_{G} *of* G *and the topology* $\mathsf{top}(C)$*. Moreover,* $\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i \in S_{\mathsf{Black}}(\mathsf{G})})$ *also only uses the topology* $\mathsf{top}(\mathsf{G}) = (i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}})$ *of* G *rather than the full description of* G*. That is, there is a PPT algorithm* $\mathsf{SimGate}'_{\mathsf{Black}}$ *such that* $\mathsf{SimGate}'_{\mathsf{Black}}(\mathsf{pp}, \mathsf{top}(\mathsf{G}), \{\mathsf{R}_i\}_{i \in S_{\mathsf{Black}}(\mathsf{G})})$ *is equivalent to* $\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i \in S_{\mathsf{Black}}(\mathsf{G})})$.
   (b) *There is a deterministic polynomial-time algorithm* SimInp *such that for any fixed* C*,* x*,* $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$*, and* $\{\mathsf{R}_i\}_{i \in \cup_{\mathsf{G} \in \mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G})}$*,*

$$\left\{ \widetilde{x} \;\middle|\; \begin{array}{l} \mathsf{R}_i \leftarrow \{0,1\}^\ell \text{ for } i \in [N_{\mathsf{rand}}] \setminus \cup_{\mathsf{G} \in \mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G}) \\ \mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i \in S_{\mathsf{st}}}) \\ \widetilde{x} \leftarrow \mathsf{SimInpConf}(\mathsf{st}, x, C(x), \mathsf{Black}^{|\mathsf{Gates}|}) \end{array} \right\}$$
$$\equiv \left\{ \widetilde{x} \;\middle|\; \begin{array}{l} \mathsf{R}_i \leftarrow \{0,1\}^\ell \text{ for } i \in [N_{\mathsf{rand}}] \setminus \cup_{\mathsf{G} \in \mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G}) \\ \mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i \in S_{\mathsf{st}}}) \\ \widetilde{x} \leftarrow \mathsf{SimInp}(\mathsf{st}, C(x)) \end{array} \right\}.$$

   That is, $\mathsf{SimInpConf}$ *for the all* Black *configuration is equivalent to* SimInp *even if we fix all the randomness needed for generating garbled gates in the* Black *mode.*

Remark 3.1. We remark that $\mathsf{SimGate}_{\mathsf{White}}$ may take randomness as input additionally to $\{\mathsf{R}_i\}_{i \in S_{\mathsf{White}}(\mathsf{G})}$. The same comment applies to $\mathsf{SimGate}_{\mathsf{White}}$ and $\mathsf{SimGate}_{\mathsf{Black}}$. Namely, in the above, we explicitly write down the randomness that is shared among the sub-algorithms by R_i terms. Each algorithm may take additional randomness that is not used by other sub-algorithms.

Definition 3.5 (Pebbling-based Simulation Security). *Let* GC *and* $\mathsf{Sim}_{\mathsf{GC}}$ *be as in Definition 3.4. We say that* GC *satisfies* (M, T)*-pebbling-based simulation security with respect to the simulator* $\mathsf{Sim}_{\mathsf{GC}}$ *if the following is satisfied. For any circuit* C *of size* size*, there exists a sequence of circuit configurations* $(\mathsf{conf}_1, \ldots, \mathsf{conf}_T)$ *that satisfies the following where we use the notation* $\mathsf{conf}_j = \{\mathsf{mode}_j(\mathsf{G})\}_{\mathsf{G} \in \mathsf{Gates}}$ *and* $I_j = \{\mathsf{G} \in \mathsf{Gates} : \mathsf{mode}_j(\mathsf{G}) = \mathsf{Gray}\}$ *for* $j \in [T]$.[14]

1. $\mathsf{conf}_1 = \mathsf{White}^{|\mathsf{Gates}|}$ *and* $\mathsf{conf}_T = \mathsf{Black}^{|\mathsf{Gates}|}$.
2. $|I_j| \leq M$ *for all* $j \in [T]$.
3. *For all,* $j \in [T-1]$*, there is exactly one* $\mathsf{G} \in \mathsf{Gates}$*, which we denote by* G^*_j*, such that* $\mathsf{mode}_j(\mathsf{G}^*_j) \neq \mathsf{mode}_{j+1}(\mathsf{G}^*_j)$*. Moreover, either* $\mathsf{mode}_j(\mathsf{G}^*_j)$ *or* $\mathsf{mode}_{j+1}(\mathsf{G}^*_j)$ *is* Gray*. Remark that this implies* $\mathsf{G}^*_j \in I_j \cup I_{j+1}$ *and* $I_j \cup I_{j+1} = I_j$ *or* $I_j \cup I_{j+1} = I_{j+1}$.
4. *For every stateful* PPT *adversary* A *and polynomial* $\mathsf{size} = \mathsf{size}(\lambda)$*, the following holds for all* $j \in [T-1]$*:*

$$\left| \Pr[\mathsf{Exp}^{j,0}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}\left(1^\lambda, 1^{\mathsf{size}}\right) = 1] - \Pr[\mathsf{Exp}^{j,1}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}\left(1^\lambda, 1^{\mathsf{size}}\right) = 1] \right| = \mathrm{negl}(\lambda)$$

   where $\mathsf{Exp}^{j,b}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$ *is an experiment defined below where we write* $S_j(\mathsf{G})$ *to mean* $S_{\mathsf{mode}_j(\mathsf{G})}(\mathsf{G})$ *for each* j *and* G*.*

$\mathsf{Exp}^{j,b}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$*: The experiment works as follows where steps that depend on* b *are marked by red underlines.*

- $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$.

[13] Remark that a garbled circuit simulated in White mode is distributed similarly to the real one since we require $\mathsf{SimGate}_{\mathsf{White}}$ is the same algorithm as GGate.

[14] Note that conf_j and I_j depend on C, but we omit this dependency for notational simplicity.

- $\left(C, x, \{\mathsf{R}_i\}_{i\in V_j}\right) \leftarrow \mathsf{A}(1^\lambda, 1^{\mathsf{size}}, \mathsf{pp})$ *where* C *is a circuit of size at most* size *and*

$$V_j := \bigcup_{\mathsf{G}\in\mathsf{Gates}\setminus(I_j\cup I_{j+1})} S_j(\mathsf{G}).$$

  That is, $V_j \subseteq [N_{\mathsf{rand}}]$ *is the subset of indices of randomness that are needed for simulation of gates outside* $I_j \cup I_{j+1}$.
- $\mathsf{R}_i \leftarrow \{0,1\}^\ell$ *for* $i \in [N_{\mathsf{rand}}] \setminus V_j$.
- *For all* $\mathsf{G} \in I_j \cup I_{j+1} \setminus \{\mathsf{G}^*_j\}$, $\widetilde{\mathsf{G}} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in[N_{\mathsf{rand}}]}, C, x)$.
- *Generate* $\widetilde{\mathsf{G}^*_j}$ *in either of the following way:*
  - *If* $\mathsf{mode}_{j+b}(\mathsf{G}^*_j) \in \{\mathsf{White}, \mathsf{Black}\}$, $\widetilde{\mathsf{G}^*_j} \leftarrow \mathsf{SimGate}_{\mathsf{mode}_{j+b}(\mathsf{G}^*_j)}(\mathsf{pp}, \mathsf{G}^*_j, \{\mathsf{R}_i\}_{i\in S_{j+b}(\mathsf{G}^*_j)})$.
  - *If* $\mathsf{mode}_{j+b}(\mathsf{G}^*_j) = \mathsf{Gray}$, $\widetilde{\mathsf{G}^*_j} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}^*_j, \{\mathsf{R}_i\}_{i\in[N_{\mathsf{rand}}]}, C, x)$.

  We note that one case happens when $b = 0$ *and the other case happens when* $b = 1$ *by Item 3.*
- $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{st}}})$.
- $\widetilde{x} \leftarrow \mathsf{SimInpConf}(\mathsf{st}, x, C(x), \mathsf{conf}_{j+b})$
- $b' \leftarrow \mathcal{A}(\{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in I_j\cup I_{j+1}}, \widetilde{x})$.
- *The experiment returns* b'.

Comparison with locally simulatable garbling in [AL18]. The definition of gate-by-gate garbling with pebbling-based simulation security is conceptually similar to *locally simulatable garbling* introduced by Ananth and Lombardi [AL18]. Indeed, both share almost the same instantiation based on [GS18]. However, their formalization is not sufficient for our purpose due to the following reasons (as well as other minor syntactic incompatibility):

1. Their definition does not require circuit privacy, i,e., a simulator is given the full description of the circuit C being garbled. On the other hand, our definition requires that only the topology of C is leaked.
2. They introduce two types of security notions, *semi-adaptive* local simulation security and *strong* local simulation security. On the other hand, we need a security notion that captures both flavors of security.

On the other hand, we remark that their definition also requires some properties that are not needed for our purpose. In principle, we could modify the definition of locally simulatable garbling to one that suffices for our purpose. However, we chose to introduce a new definition rather than resolving all the inconsistencies.

Remark 3.2. We give an LOT-based instantiation in Appendix A based on [GS18]. We note that a similar pebbling-based strategy was also used in [HJO+16] for proving adaptive security of a variant of Yao's garbling based on one-way functions. Unfortunately, however, our definitions do not seem to capture [HJO+16]. It would be possible to generalize our definitions to capture [HJO+16]. (Indeed, locally simulatable garbling in [AL18] captures [HJO+16].) However, a scheme based on [HJO+16] would support only bounded width or bounded depth circuits. Since our purpose is to construct a CPFE scheme with unbounded size without any restriction on depth or width, such an instantiation is not useful for our purpose. For this reason, we choose to give a simpler formalization that only captures [GS18] instead of introducing a more general definition, which would be more complicated.

Instantiation. Our definition of gate-by-gate garbling with pebbling-based simulation security captures the backbone of the proof technique of adaptive garbling in [GS18]. The following lemma is implicit in their work. The lemma is proven in Appendix A for completeness.

Lemma 3.1 (Implicit in [GS18]). *If there exists LOT, which exists assuming either of* CDH, Factoring, *or* LWE, *there exists a gate-by-gate garbling for all polynomial-size circuits that satisfies* (M, T)*-pebbling-based simulation security for* $M = O(\log \mathsf{size})$ *and* $T = \mathrm{poly}(\mathsf{size})$ *where* size *is the size of a circuit being garbled.*

4 AD-SIM CPFE with Dynamic Bounded Collusion

In this section, we construct an AD-SIM secure CPFE scheme CPFE for unbounded polynomial-size circuits with dynamic bounded collusion. CPFE supports the function class $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ for any polynomials $\mathsf{inp} = \mathsf{inp}(\lambda)$ and $\mathsf{out} = \mathsf{out}(\lambda)$ where $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ is the class of circuits with input-length inp and output-length out.

4.1 Multi-Ciphertext Security of FE

4.1.1 Preparation We prepare a definition of symmetric key encryption with pseudorandom ciphertexts that is used as a building block in our construction of multi-ciphertext secure CPFE.

Definition 4.1 (Symmetric key encryption:). *A symmetric key encryption (SKE) scheme with the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ consists of PPT algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{K}$*: The setup algorithm takes as input the security parameter 1^λ and outputs a secret key* K.
$\mathsf{Enc}(\mathsf{K}, m) \to \mathsf{ct}$*: The encryption algorithm takes as input a secret key* K *and a message $m \in \mathcal{M}$ and outputs a ciphertext* $\mathsf{ct} \in \mathcal{C}$.
$\mathsf{Dec}(\mathsf{K}, \mathsf{ct}) \to m$*: The decryption algorithm is a deterministic algorithm that takes as input a secret key* K *and a ciphertext* $\mathsf{ct} \in \mathcal{C}$ *and outputs a message m.*

We require an SKE scheme to satisfy the following properties:
Correctness. *For any $\lambda \in \mathbb{N}$, $m \in \mathcal{M}$, and $\mathsf{K} \leftarrow \mathsf{Setup}(1^\lambda)$, it holds that*

$$\mathsf{Dec}(\mathsf{K}, \mathsf{Enc}(\mathsf{K}, m)) = m.$$

Prseudorandom ciphertext. *For every PPT adversary* A *and messages $m \in \mathcal{M}$, it holds that*

$$|\Pr[\mathsf{A}^{\mathsf{Enc}(\mathsf{K},\cdot)}(\mathsf{ct}) = 1] - \Pr[\mathsf{A}^{\mathsf{Enc}(\mathsf{K},\cdot)}(\mathsf{ct}') = 1]| = \mathrm{negl}(\lambda)$$

where $\mathsf{K} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{K}, m)$, *and* $\mathsf{ct}' \leftarrow \mathcal{C}$.

Such an SKE scheme can be easily constructed based on any (weak) PRF, which exists under the existence of OWF.

4.1.2 Definition We define multi-ciphertext AD-SIM-security for FE since that is needed as a building block of our construction of an AD-SIM secure CPFE scheme for unbounded polynomial-size circuits with dynamic bounded collusion in Section 4. We remark that our multi-ciphertext security is formalized in a different way from the standard one, e.g., [GVW12, Appendix A]. Specifically, we require the indistinguishability between "$K-1$ simulated ciphertexts + 1 real ciphertext" and "K simulated ciphertexts" whereas the standard definition requires the indistinguishability between "K real ciphertext" and "K simulated ciphertexts". These two definitions seem incomparable in general. We define our security notion in this way since this is more convenient in the security proof for the scheme in Section 4. For convenience, we describe the definition in the form of CPFE.

Definition 4.2 (K-CT AD-SIM Security for FE with Dynamic Bounded Collusion). *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a CPFE scheme with dynamic bounded collusion for the circuit class $\mathcal{C}$. For every stateful PPT adversary* A *and a stateful PPT simulator* $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$*, consider the following experiments:*

We emphasize that the adversary A *is stateful, even though we do not explicitly include the internal state of it into the output above for the simplicity of the notation. On the other hand, the above explicitly denotes the internal state of the simulator* Sim *by* st*. The oracles $\mathcal{O}$ and $\mathcal{O}'$ works as follows:*

- *The oracle $\mathcal{O}(\mathsf{st}, \mathsf{msk}, \cdot, \cdot)$ takes as input a tuple $\{y_i^{(q)}\}_{i\in[K'-1]}$ for $q \in [Q_1+1, Q_1+Q_2]$ and the q-th key query $x^{(q)}$ and returns*
  - $\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, \{y_i^{(q)}\}_{i\in[K'-1]}, x^{(q)})$ *if $K' \geq 2$*
  - $\mathsf{KeyGen}(\mathsf{msk}, x^{(q)})$ *if $K' = 1$.*

  where $Q_1 + Q_2 \leq Q$. For the case of $K' = 1$, we often write $\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, \{y_i^{(q)}\}_{i\in[K'-1]}, x^{(q)})$ *to mean* $\mathsf{KeyGen}(\mathsf{msk}, x^{(q)})$ *for convenience.*
- *The oracle $\mathcal{O}'(\mathsf{st}, \mathsf{msk}, \cdot, \cdot)$ takes as input a tuple $\{y_i^{(q)}\}_{i\in[K'-1]}$ for $q \in [Q_1+1, Q_1+Q_2]$ and the q-th key query $x^{(q)}$, computes $y_{K'}^{(q)} := C(x^{(q)})$ and returns* $\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, \{y_i^{(q)}\}_{i\in[K']}, x^{(q)})$.

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{A},\mathsf{Sim}}(1^\lambda)$	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{A},\mathsf{Sim}}(1^\lambda)$
1. $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$	1. $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$
2. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$	2. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$
3. $(\{y_i^{(q)}\}_{i\in[K'-1],q\in[Q_1]}, C, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$ where $1 \le K' \le K$.	3. $(\{y_i^{(q)}\}_{i\in[K'-1],q\in[Q_1]}, C, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$ where $1 \le K' \le K$.
– Let $(x^{(1)}, \dots, x^{(Q_1)})$ be A's oracle queries. – Let $\mathsf{sk}^{(q)}$ be the oracle reply to $x^{(q)}$. – Let $\mathcal{V} := \left\{ \left(\{y_i^{(q)}\}_{i\in[K'-1]}, x^{(q)}, \mathsf{sk}^{(q)} \right) \right\}_{q\in[Q_1]}$.	– Let $(x^{(1)}, \dots, x^{(Q_1)})$ be A's oracle queries. – Let $\mathsf{sk}^{(q)}$ be the oracle reply to $x^{(q)}$. – Let $y^{(q)}_{K'} := C(x^{(q)})$. – Let $\mathcal{V}' := \left\{ \left(\{y_i^{(q)}\}_{i\in[K']}, x^{(q)}, \mathsf{sk}^{(q)} \right) \right\}_{q\in[Q_1]}$.
4. $(\{\mathsf{ct}_i\}_{i\in[K'-1]}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}, 1^{\lvert C\rvert}, 1^Q)$	4. $(\{\mathsf{ct}_i\}_{i\in[K']}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}', 1^{\lvert C\rvert}, 1^Q)$
5. $\mathsf{ct}_{K'} \leftarrow \mathsf{Enc}(\mathsf{mpk}, C, 1^Q)$	5.
6. $b \leftarrow \mathsf{A}^{\mathcal{O}(\mathsf{st},\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[K']})$	6. $b \leftarrow \mathsf{A}^{\mathcal{O}'(\mathsf{st},\mathsf{msk},\cdot,\cdot)}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[K']})$
7. Output b	7. Output b

The FE scheme FE *is then said to be* K*-CT* AD-SIM*-secure against dynamic bounded collusion if there is a* PPT *simulator* Sim *such that for every* PPT *adversary* A*, the following holds:*

$$\left| \Pr[\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{A},\mathsf{Sim}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{A},\mathsf{Sim}}(1^\lambda) = 1] \right| = \mathrm{negl}(\lambda).$$

Intuitively, $y_i^{(q)}$ in the above security games means what is supposed to be $C_i(x^{(q)})$ where C_i is the i-th challenge message and $x^{(q)}$ is the q-th key query. In standard K-challenge security, A should specify C_i for each $i \in [K]$, and $y_i^{(q)}$ is just replaced with $C_i(x^{(q)})$. However, for our purpose, we need to allow an adversary to specify the tuple of outputs $\{y_i^{(q)}\}_{i,q}$ without specifying the corresponding messages $\{C_i\}_i$ for a technical reason. Note that the experiments are still well-defined since we consider two hybrids, in both of which $K-1$ out of K ciphertexts are generated by the simulator, which only needs outputs $\{y_i^q\}_{i,q}$ rather than the messages $\{C_i\}_i$ themselves.

De Caro et al. [CIJ+13] sketched that an FE scheme with indistinguishability-based adaptive security, which is weaker than single-ciphertext AD-SIM-security, can be converted into one with multi-ciphertext AD-SIM-security in a slightly different definition from ours in the setting where the adversary is allowed to make unbounded number of post-challenge key queries. We observe that the same conversion works for the above definition in the dynamic bounded collusion setting. Moreover, a single-ciphertext AD-SIM-secure CPFE with dynamic bounded collusion for bounded polynomial-size circuits is constructed in [AMVY21, Section 3.3] from IBE. Therefore, we have the following lemma.

Lemma 4.1. *If there exists IBE, for any polynomials* $K = K(\lambda)$*,* $\mathsf{inp} = \mathsf{inp}(\lambda)$*,* $\mathsf{out} = \mathsf{out}(\lambda)$*, and* $\mathsf{size} = \mathsf{size}(\lambda)$*, there exists a* K*-CT* AD-SIM*-secure CPFE scheme with dynamic bounded collusion for* $\mathcal{C}_{\mathsf{inp},\mathsf{out},\mathsf{size}}$*, which is the class of circuits with input-length* inp*, output-length* out*, and size* size*.*

We prove the above lemma in the Appendix B.1.

4.2 Construction of AD-SIM CPFE with Dynamic Bounded Collusion

Ingredients. We now describe the underlying building blocks used to obtain CPFE:

1. A gate-by-gate garbling scheme $\mathsf{GC} = (\mathsf{GCkt}, \mathsf{GInp}, \mathsf{GEval})$ for $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ with N_{rand} randomness slots and randomness length ℓ that satisfies (M, T)-pebbling-based simulation security for $M = \mathrm{poly}(\lambda)$ and $T = \mathrm{poly}(\lambda)$. By Lemma 3.1, such a scheme exists under the existence of laconic OT, which in turn exists under either of CDH, Factoring, or LWE. We denote by $\mathcal{R}$ the randomness space of GGate.
2. A PRF $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ from $\{0,1\}^{\mathsf{inp}}$ to $\{0,1\}^\ell$.
3. A PRF $\mathsf{PRF}' = (\mathsf{PRF}'.\mathsf{Setup}, \mathsf{PRF}'.\mathsf{Eval})$ from $\{0,1\}^{\mathsf{inp}}$ to $\mathcal{R}$.

Circuit $\mathsf{GarbleCirc}$

Input: A string $x \in \{0,1\}^{\mathsf{inp}}$ and a key-tag $r \in \{0,1\}^{\lambda}$
Hardwired value: a public parameter pp, gate G, set $S(\mathsf{G})$, tuple of PRF keys $\{\mathsf{K}_i\}_{i\in S(\mathsf{G})}$, and PRF key K'_{G}.

1. Compute $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$ for $i \in S(\mathsf{G})$ and $\mathsf{R}'_{\mathsf{G}} \leftarrow \mathsf{PRF'.Eval}(\mathsf{K}'_{\mathsf{G}}, r)$.
2. Output $\widetilde{\mathsf{G}} := \mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S(\mathsf{G})}; \mathsf{R}'_{\mathsf{G}})$.

Fig. 3 The description of GarbleCirc

Circuit $\mathsf{GarbleInp}$

Input: A string $x \in \{0,1\}^{\mathsf{inp}}$ and a key-tag $r \in \{0,1\}^{\lambda}$
Hardwired value: a public parameter pp and a tuple of PRF keys $\{\mathsf{K}_i\}_{i\in S_{\mathsf{st}}}$.

1. Compute $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$ for $i \in S_{\mathsf{st}}$.
2. Set $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{st}}})$.
3. Output $\widetilde{x} := \mathsf{GInp}(\mathsf{st}, x)$.

Fig. 4 The description of GarbleInp

4. An M-CT AD-SIM secure CPFE scheme with dynamic bounded collusion denoted by

$$\mathsf{BCPFE} = (\mathsf{BCPFE.Setup}, \mathsf{BCPFE.Enc}, \mathsf{BCPFE.KeyGen}, \mathsf{BCPFE.Dec})$$

for bounded polynomial-size circuits. Here, M-CT AD-SIM security roughly means security against adversaries that see M challenge ciphertexts. Due to a technical reason, however, our notion of M-CT AD-SIM security is slightly different from the standard one e.g., [GVW12, Appendix A]. Our definition is given in Definition 4.2 in Section 4.1.2. We construct a CPFE scheme for bounded polynomial-size circuits that satisfies this security notion from IBE in Appendix B.1.
We require BCPFE to support a circuit class $\mathcal{C}_{\mathsf{BCPFE.inp},\mathsf{BCPFE.out},\mathsf{BCPFE.size}}$ consisting of circuits with input length BCPFE.inp, output length BCPFE.out, and size at most BCPFE.size, where $\mathsf{BCPFE.inp} := \mathsf{inp} + \lambda$ and BCPFE.out and BCPFE.size are output-length and the maximum size of the circuit GarbleCirc defined in Figure 3, respectively. By the efficiency requirements of GC (Definition 3.2), $\mathsf{BCPFE.out} = \mathrm{poly}(\lambda, \mathsf{inp})$ and $\mathsf{BCPFE.size} = \mathrm{poly}(\lambda, \mathsf{inp})$ independently of the size of the circuit C being encrypted.
5. A (single-ciphertext) AD-SIM-secure CPFE scheme with dynamic bounded collusion denoted by

$$\mathsf{BCPFE}' = (\mathsf{BCPFE}'.\mathsf{Setup}, \mathsf{BCPFE}'.\mathsf{Enc}, \mathsf{BCPFE}'.\mathsf{KeyGen}, \mathsf{BCPFE}'.\mathsf{Dec})$$

for bounded polynomial-size circuits. We require BCPFE' to support a circuit class $\mathcal{C}_{\mathsf{BCPFE}'.\mathsf{inp},\mathsf{BCPFE}'.\mathsf{out},\mathsf{BCPFE}'.\mathsf{size}}$ consisting of circuits with input length BCPFE'.inp, output length BCPFE'.out, and size at most BCPFE'.size, where $\mathsf{BCPFE}'.\mathsf{inp} := \mathsf{inp} + \lambda$ and BCPFE'.out and BCPFE'.size are the output length and the maximum size of the circuit GarbleInp defined in Figure 4, respectively. By the efficiency requirements of GC (Definition 3.2), $\mathsf{BCPFE}'.\mathsf{out} = \mathrm{poly}(\lambda, \mathsf{inp})$ and $\mathsf{BCPFE}'.\mathsf{size} = \mathrm{poly}(\lambda, \mathsf{inp})$ independently of the size of the circuit C being encrypted.

Construction. In the construction, for a circuit C, we define the universal circuit U_C such that $U_C(x)$=$C(x)$. We define U_C in such a way that the topology of U_C does not reveal anything beyond the size of C.[15] The description of CPFE is given below.

$\mathsf{Setup}(1^{\lambda}, \mathsf{prm})$: On input the security parameter λ and the parameter prm, do the following:

[15] We explain how to construct such U_C in Section 2.1.

1. Run $(\mathsf{BCPFE.mpk}, \mathsf{BCPFE.msk}) \leftarrow \mathsf{BCPFE.Setup}(1^\lambda, \mathsf{BCPFE.prm})$.
2. Run $(\mathsf{BCPFE}'.\mathsf{mpk}, \mathsf{BCPFE}'.\mathsf{msk}) \leftarrow \mathsf{BCPFE}'.\mathsf{Setup}(1^\lambda, \mathsf{BCPFE}'.\mathsf{prm})$.
3. Output $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{BCPFE.mpk}, \mathsf{BCPFE}'.\mathsf{mpk}), (\mathsf{BCPFE.msk}, \mathsf{BCPFE}'.\mathsf{msk}))$.

$\mathsf{Enc}(\mathsf{mpk}, C, 1^Q)$: On input the master public key $\mathsf{mpk} = (\mathsf{BCPFE.mpk}, \mathsf{BCPFE}'.\mathsf{mpk})$, a circuit $C \in \mathcal{C}_{\mathsf{inp},\mathsf{out}}$, and the query bound $1 \le Q \le 2^\lambda$ in unary form, do the following:

1. Compute the universal circuit U_C. In the following, we write Gates to mean the set of gates of U_C rather than C.
2. Run $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(U_C))$.
3. For $i \in [N_{\mathsf{rand}}]$, generate $\mathsf{K}_i \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
4. For $\mathsf{G} \in \mathsf{Gates}$ generate $\mathsf{K}'_\mathsf{G} \leftarrow \mathsf{PRF}'.\mathsf{Setup}(1^\lambda)$.
5. For $\mathsf{G} \in \mathsf{Gates}$, run

$$\mathsf{BCPFE.ct}_\mathsf{G} \leftarrow \mathsf{BCPFE.Enc}(\mathsf{BCPFE.mpk}, \mathsf{GarbleCirc}[\mathsf{pp}, \mathsf{G}, S(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S(\mathsf{G})}, \mathsf{K}'_\mathsf{G}], 1^Q)$$

   where $\mathsf{GarbleCirc}[\mathsf{pp}, \mathsf{G}, S(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S(\mathsf{G})}, \mathsf{K}'_\mathsf{G}]$ is the circuit as defined in Figure 3.
6. Run $\mathsf{BCPFE}'.\mathsf{ct} \leftarrow \mathsf{BCPFE}'.\mathsf{Enc}(\mathsf{BCPFE}'.\mathsf{mpk}, \mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i\in S_\mathsf{st}}])$ where $\mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i\in S_\mathsf{st}}]$ is the circuit as defined in Figure 4.
7. Output $\mathsf{ct} := (\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G}\in\mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$: On input the master secret key $\mathsf{msk} = (\mathsf{BCPFE.msk}, \mathsf{BCPFE}'.\mathsf{msk})$ and an input $x \in \{0,1\}^{\mathsf{inp}}$, do the following:

1. Generate $r \leftarrow \{0,1\}^\lambda$.
2. Run $\mathsf{BCPFE.sk} \leftarrow \mathsf{BCPFE.KeyGen}(\mathsf{BCPFE.msk}, (x, r))$.
3. Run $\mathsf{BCPFE}'.\mathsf{sk} \leftarrow \mathsf{BCPFE}'.\mathsf{KeyGen}(\mathsf{BCPFE}'.\mathsf{msk}, (x, r))$.
4. Output $\mathsf{sk} := (r, \mathsf{BCPFE.sk}, \mathsf{BCPFE}'.\mathsf{sk})$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, 1^Q)$: On input a ciphertext $\mathsf{ct} = (\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G}\in\mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$ and a secret key $\mathsf{sk} = (r, \mathsf{BCPFE.sk}, \mathsf{BCPFE}'.\mathsf{sk})$, do the following:

1. For $\mathsf{G} \in \mathsf{Gates}$, run $\widetilde{\mathsf{G}} \leftarrow \mathsf{BCPFE.Dec}(\mathsf{BCPFE.ct}_\mathsf{G}, \mathsf{BCPFE.sk}, 1^Q)$
2. Run $\widetilde{x} \leftarrow \mathsf{BCPFE}'.\mathsf{Dec}(\mathsf{BCPFE}'.\mathsf{ct}, \mathsf{BCPFE}'.\mathsf{sk}, 1^Q)$.
3. Set $\widetilde{U_C} := (\mathsf{pp}, \{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}})$.
4. Compute and output $\mathsf{GEval}(\widetilde{U_C}, \widetilde{x})$.

Correctness Let $\mathsf{ct} = (\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G}\in\mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$ be an honestly generated ciphertext for a circuit C and $\mathsf{sk} = (r, \mathsf{BCPFE.sk}, \mathsf{BCPFE}'.\mathsf{sk})$ be an honestly generated secret key for an input x. By the correctness of BCPFE, for each $\mathsf{G} \in \mathsf{Gates}$, if we generate $\widetilde{\mathsf{G}} \leftarrow \mathsf{BCPFE.Dec}(\mathsf{BCPFE.ct}_\mathsf{G}, \mathsf{BCPFE.sk}, 1^Q)$, then we have $\widetilde{\mathsf{G}} = \mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S(\mathsf{G})}; \mathsf{R}'_\mathsf{G})$ where $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$ for $i \in S(\mathsf{G})$ and $\mathsf{R}'_\mathsf{G} \leftarrow \mathsf{PRF}'.\mathsf{Eval}(\mathsf{K}'_\mathsf{G}, r)$. Similarly, by the correctness of BCPFE', if we generate $\widetilde{x} \leftarrow \mathsf{BCPFE}'.\mathsf{Dec}(\mathsf{BCPFE}'.\mathsf{ct}, \mathsf{BCPFE}'.\mathsf{sk}, 1^Q)$, then we have $\widetilde{x} = \mathsf{GInp}(\mathsf{st}, x)$ where $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$ for $i \in S_\mathsf{st}$ and $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i\in S_\mathsf{st}})$. Then, by the (perfect) correctnes of GC, $\mathsf{GEval}(\widetilde{U_C}, \widetilde{x}) = U_C(x) = C(x)$ where $\widetilde{U_C} := (\mathsf{pp}, \{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}})$.

Security The following theorem asserts the security of CPFE.

Theorem 4.1. *If* GC *satisfies* (M, T)*-pebbling-based simulation security for* $M = \mathrm{poly}(\lambda)$ *and* $T = \mathrm{poly}(\lambda)$*,* BCPFE *is* M*-CT* AD-SIM*-secure against dynamic bounded collusion,* BCPFE' *is* AD-SIM*-secure against dynamic bounded collusion, and* PRF *and* PRF' *are secure pseudorandom functions, then* CPFE *is* AD-SIM*-secure against dynamic bounded collusion.*

Proof. We prove the CPFE scheme CPFE constructed in Section 4 is AD-SIM-secure against dynamic bounded collusion.

For convenience of the security proof, we redefine the circuit GarbleCirc as in Figure 5. Compared to the definition in Figure 3 in Section 4, it has an additional hardwired value $\mathsf{mode} \in \{\mathsf{White}, \mathsf{Black}\}$ and uses $\mathsf{SimGate}_\mathsf{mode}$ instead of GGate. Since $\mathsf{SimGate}_\mathsf{White}$ is identical to GGate as required in Definition 3.4, $\mathsf{GarbleCirc}[\ldots]$ as defined in Figure 3 is identical to $\mathsf{GarbleCirc}[\mathsf{White}, \ldots]$ as defined in Figure 5. For the circuit to be well-defined, we assume that the randomness spaces of $\mathsf{SimGate}_\mathsf{White}$ and $\mathsf{SimGate}_\mathsf{Black}$ are the same as the range $\mathcal{R}$ of PRF' without loss of generality.

Circuit $\mathsf{GarbleCirc}$

Input: A string $x \in \{0,1\}^{\mathsf{inp}}$ and a key-tag $r \in \{0,1\}^{\lambda}$
Hardwired value: a mode $\mathsf{mode} \in \{\mathsf{White}, \mathsf{Black}\}$, public parameter pp, gate G, set $S_{\mathsf{mode}}(\mathsf{G})$, tuple of PRF keys $\{\mathsf{K}_i\}_{i \in S_{\mathsf{mode}}(\mathsf{G})}$, and PRF key K'_{G}.

1. Compute $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$ for $i \in S_{\mathsf{mode}}(\mathsf{G})$ and $\mathsf{R}'_{\mathsf{G}} \leftarrow \mathsf{PRF}'.\mathsf{Eval}(\mathsf{K}'_{\mathsf{G}}, r)$.
2. Output $\widetilde{\mathsf{G}} := \mathsf{SimGate}_{\mathsf{mode}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i \in S_{\mathsf{mode}}(\mathsf{G})}; \mathsf{R}'_{\mathsf{G}})$.

Fig. 5 The description of GarbleCirc (slighly modified from Figure 3)

Circuit $\mathsf{GarbleInp}$

Input: A string $x \in \{0,1\}^{\mathsf{inp}}$ and a key-tag $r \in \{0,1\}^{\lambda}$
Hardwired value: a public parameter pp and a tuple of PRF keys $\{\mathsf{K}_i\}_{i \in S_{\mathsf{st}}}$.

1. Compute $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$ for $i \in S_{\mathsf{st}}$.
2. Set $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_i\}_{i \in S_{\mathsf{st}}})$.
3. Output $\widetilde{x} := \mathsf{GInp}(\mathsf{st}, x)$.

Fig. 6 The description of GarbleInp (identical to Figure 4)

Moreover, we require that BCPFE supports a circuit class that contains the redefined GarbleCirc as defined in Figure 5. This is possible since the size of GarbleCirc as defined in Figure 5 is independent of the size of the circuit C being encrypted by the efficiency requirements of GC (Definitions 3.2 and 3.4).

For clarity, we rewrite the construction of CPFE based on the redefined GarbleCirc. Though nothing is changed for GarbleInp, we redisplay it in Figure 6 for convenience of the readers.

$\mathsf{Setup}(1^{\lambda}, \mathsf{prm})$: On input the security parameter λ and the parameter prm, do the following:

1. Run $(\mathsf{BCPFE.mpk}, \mathsf{BCPFE.msk}) \leftarrow \mathsf{BCPFE.Setup}(1^{\lambda}, \mathsf{BCPFE.prm})$.
2. Run $(\mathsf{BCPFE}'.\mathsf{mpk}, \mathsf{BCPFE}'.\mathsf{msk}) \leftarrow \mathsf{BCPFE}'.\mathsf{Setup}(1^{\lambda}, \mathsf{BCPFE}'.\mathsf{prm})$.
3. Output $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{BCPFE.mpk}, \mathsf{BCPFE}'.\mathsf{mpk}), (\mathsf{BCPFE.msk}, \mathsf{BCPFE}'.\mathsf{msk}))$.

$\mathsf{Enc}(\mathsf{mpk}, C, 1^{Q})$: On input the master public key $\mathsf{mpk} = (\mathsf{BCPFE.mpk}, \mathsf{BCPFE}'.\mathsf{mpk})$, a circuit $C \in \mathcal{C}_{\mathsf{inp},\mathsf{out}}$, and the query bound $1 \leq Q \leq 2^{\lambda}$ in unary form, do the following:

1. Compute the universal circuit U_C. In the following, we write Gates to mean the set of gates of U_C rather than C.
2. Run $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^{\lambda}, \mathsf{top}(U_C))$.
3. For $i \in [N_{\mathsf{rand}}]$, generate $\mathsf{K}_i \leftarrow \mathsf{PRF.Setup}(1^{\lambda})$
4. For $\mathsf{G} \in \mathsf{Gates}$ generate $\mathsf{K}'_{\mathsf{G}} \leftarrow \mathsf{PRF}'.\mathsf{Setup}(1^{\lambda})$.
5. For $\mathsf{G} \in \mathsf{Gates}$, run

$$\mathsf{BCPFE.ct}_{\mathsf{G}} \leftarrow \mathsf{BCPFE.Enc}(\mathsf{BCPFE.mpk}, \mathsf{GarbleCirc}[\mathsf{White}, \mathsf{pp}, \mathsf{G}, S_{\mathsf{White}}(\mathsf{G}), \{\mathsf{K}_i\}_{i \in S_{\mathsf{White}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}])$$

   where $\mathsf{GarbleCirc}[\mathsf{White}, \mathsf{pp}, \mathsf{G}, S_{\mathsf{White}}(\mathsf{G}), \{\mathsf{K}_i\}_{i \in S_{\mathsf{White}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}]$ is the circuit as defined in Figure 5. We remark that when $\mathsf{mode} = \mathsf{White}$, $\mathsf{SimGate}_{\mathsf{mode}}$, which is run by GarbleCirc, is the same algorithm as GGate as required in Definition 3.4.
6. Run $\mathsf{BCPFE}'.\mathsf{ct} \leftarrow \mathsf{BCPFE}'.\mathsf{Enc}(\mathsf{BCPFE}'.\mathsf{mpk}, \mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i \in S_{\mathsf{st}}}])$ where $\mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i \in S_{\mathsf{st}}}]$ is the circuit as defined in Figure 6.
7. Output $\mathsf{ct} := (\{\mathsf{BCPFE.ct}_{\mathsf{G}}\}_{\mathsf{G} \in \mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$: On input the master secret key $\mathsf{msk} = (\mathsf{BCPFE.msk}, \mathsf{BCPFE}'.\mathsf{msk})$ and an input $x \in \{0,1\}^{\mathsf{inp}}$, do the following:

1. Generate $r \leftarrow \{0,1\}^{\lambda}$.

2. Run $\mathsf{BCPFE.sk} \leftarrow \mathsf{BCPFE.KeyGen}(\mathsf{BCPFE.msk}, (x, r))$.
3. Run $\mathsf{BCPFE}'.\mathsf{sk} \leftarrow \mathsf{BCPFE}'.\mathsf{KeyGen}(\mathsf{BCPFE}'.\mathsf{msk}, (x, r))$.
4. Output $\mathsf{sk} := (r, \mathsf{BCPFE.sk}, \mathsf{BCPFE}'.\mathsf{sk})$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, 1^Q)$: On input a ciphertext $\mathsf{ct} = (\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G}\in\mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$ and a secret key $\mathsf{sk} = (r, \mathsf{BCPFE.sk}, \mathsf{BCPFE}'.\mathsf{sk})$, do the following:

1. For $\mathsf{G} \in \mathsf{Gates}$, run $\widetilde{\mathsf{G}} \leftarrow \mathsf{BCPFE.Dec}(\mathsf{BCPFE.ct}_\mathsf{G}, \mathsf{BCPFE.sk}, 1^Q)$
2. Run $\widetilde{x} \leftarrow \mathsf{BCPFE}'.\mathsf{Dec}(\mathsf{BCPFE}'.\mathsf{ct}, \mathsf{BCPFE}'.\mathsf{sk}, 1^Q)$.
3. Set $\widetilde{U_C} := (\mathsf{pp}, \{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}})$.
4. Compute and output $\mathsf{GEval}(\widetilde{U_C}, \widetilde{x})$.

The security proof is given below:

Proof (Proof of Theorem 4.1). Let $\mathsf{BCPFE.Sim} = (\mathsf{BCPFE.SimEnc}, \mathsf{BCPFE.SimKG})$ and $\mathsf{BCPFE}'.\mathsf{Sim} = (\mathsf{BCPFE}'.\mathsf{SimEnc}, \mathsf{BCPFE}'.\mathsf{SimKG})$ be the simulators for BCPFE and BCPFE', respectively. Let A be a PPT adversary that makes at most Q key queries. We consider the following sequence of games. Let $\mathcal{E}_{\mathrm{xx}}$ be the event that $\mathsf{Game}_{\mathrm{xx}}$ outputs 1.

$\mathbf{Game}_0$: This is the real experiment $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE},\mathsf{A}}(1^\lambda)$ as defined in Definition 2.3. That is, the game works as follows:

Setup:

1. Run $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$.
2. Run $(\mathsf{BCPFE.mpk}, \mathsf{BCPFE.msk}) \leftarrow \mathsf{BCPFE.Setup}(1^\lambda, \mathsf{BCPFE.prm})$.
3. Run $(\mathsf{BCPFE}'.\mathsf{mpk}, \mathsf{BCPFE}'.\mathsf{msk}) \leftarrow \mathsf{BCPFE}'.\mathsf{Setup}(1^\lambda, \mathsf{BCPFE}'.\mathsf{prm})$.
4. Set $(\mathsf{mpk}, \mathsf{msk}) := ((\mathsf{BCPFE.mpk}, \mathsf{BCPFE}'.\mathsf{mpk}), (\mathsf{BCPFE.msk}, \mathsf{BCPFE}'.\mathsf{msk}))$.

Pre-challenge key queries and challenge query: Run $(C, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$.

Encryption:

1. Compute the universal circuit U_C.
2. Run $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(U_C))$.
3. For $i \in [N_{\mathsf{rand}}]$, generate $\mathsf{K}_i \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
4. For $\mathsf{G} \in \mathsf{Gates}$, generate $\mathsf{K}'_\mathsf{G} \leftarrow \mathsf{PRF}'.\mathsf{Setup}(1^\lambda)$.

Encryption by BCPFE: For $\mathsf{G} \in \mathsf{Gates}$, run

$$\mathsf{BCPFE.ct}_\mathsf{G} \leftarrow \mathsf{BCPFE.Enc}(\mathsf{BCPFE.mpk}, \mathsf{GarbleCirc}[\mathsf{White}, \mathsf{pp}, \mathsf{G}, S_{\mathsf{White}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{White}}(\mathsf{G})}, \mathsf{K}'_\mathsf{G}])$$

where $\mathsf{GarbleCirc}[\mathsf{White}, \mathsf{pp}, \mathsf{G}, S_{\mathsf{White}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{White}}(\mathsf{G})}, \mathsf{K}'_\mathsf{G}]$ is the circuit as defined in Figure 5.

Encryption by BCPFE': Run

$$\mathsf{BCPFE}'.\mathsf{ct} \leftarrow \mathsf{BCPFE}'.\mathsf{Enc}(\mathsf{BCPFE}'.\mathsf{mpk}, \mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i\in S_{\mathsf{st}}}])$$

where $\mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i\in S_{\mathsf{st}}}]$ is the circuit as defined in Figure 6.

5. Set $\mathsf{ct} := (\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G}\in\mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$.

Post-challenge key queries: Run $b \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$.

Final output: Output b.

Let Q_1 and Q_2 be the number of pre-challenge and post-challenge key queries, respectively, $x^{(q)}$ be the q-th key query, and $\mathsf{sk}^{(q)} = (r^{(q)}, \mathsf{BCPFE.sk}^{(q)}, \mathsf{BCPFE}'.\mathsf{sk}^{(q)})$ be the secret key answered to the q-th query for $q \in [Q_1 + Q_2]$.

$\mathbf{Game}_1$: This game works similarly to $\mathbf{Game}_0$ except that if there are $q \neq q'$ such that $r^{(q)} = r^{(q')}$, it outputs a uniformly random bit. Since $r^{(q)}$ is uniformly chosen from $\{0,1\}^\lambda$ for each $q \in [Q_1 + Q_2]$ and $Q_1 + Q_2 \leq Q$, the probability that this happens is at most $Q^2 \cdot 2^{-\lambda}$. Therefore, we have

$$|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq Q^2 \cdot 2^{-\lambda} = \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$: This game works similarly to $\mathbf{Game}_1$ except that $\mathsf{BCPFE}'.\mathsf{ct}$ and the post-challenge key generation oracle are simulated by the simulator. That is, "Encryption by BCPFE'" and "Post-challenge key queries" phases are modified as follows where differences from $\mathbf{Game}_1$ in "Post-challenge key queries" phase are marked by red underlines. (We do not put red underlines for "Encryption by BCPFE'" phase since that is completely modified.)

Encryption by BCPFE':

(a) For $q \in [Q_1]$, run $\widetilde{x}^{(q)} \leftarrow \mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i\in S_{\mathsf{st}}}](x^{(q)}, r^{(q)})$. Specifically, do the following:
   i. For $i \in S_{\mathsf{st}}$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.
   ii. $\mathsf{st}^{(q)} := (\mathsf{pp}, \{\mathsf{R}_i^{(q)}\}_{i\in S_{\mathsf{st}}})$.
   iii. Compute $\widetilde{x}^{(q)} \leftarrow \mathsf{GarbleInp}(\mathsf{st}^{(q)}, x^{(q)})$.

(b) Set $\mathcal{V}_{\mathsf{BCPFE}'} := \left\{\left(\widetilde{x}^{(q)}, (x^{(q)}, r^{(q)}), \mathsf{BCPFE}'.\mathsf{sk}^{(q)}\right)\right\}_{q\in[Q_1]}$.

(c) Run

$$(\mathsf{BCPFE}'.\mathsf{ct}, \mathsf{BCPFE}'.\mathsf{st}) \leftarrow \mathsf{BCPFE}'.\mathsf{SimEnc}(\mathsf{BCPFE.mpk}, \mathcal{V}_{\mathsf{BCPFE}'}, 1^{\mathsf{BCPFE}'.\mathsf{size}}, 1^Q).$$

Post-challenge key queries: Run $b \leftarrow \mathsf{A}^{\mathcal{O}_2(\mathsf{st},\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$ where $\mathcal{O}_2$ is an oracle that works as follows:

$\mathcal{O}_2$: On input $x^{(q)}$ for $q \in [Q_1+1, Q_1+Q_2]$, do the following:

(a) Generate $r^{(q)} \leftarrow \{0,1\}^\lambda$.

(b) Run $\mathsf{BCPFE.sk}^{(q)} \leftarrow \mathsf{BCPFE.KeyGen}(\mathsf{BCPFE.msk}, (x^{(q)}, r^{(q)}))$.

(c) Run $\widetilde{x}^{(q)} \leftarrow \mathsf{GarbleInp}[\mathsf{pp}, \{\mathsf{K}_i\}_{i\in S_{\mathsf{st}}}](x^{(q)}, r^{(q)})$. Specifically, do the following:
   i. For $i \in S_{\mathsf{st}}$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.
   ii. $\mathsf{st}^{(q)} := (\mathsf{pp}, \{\mathsf{R}_i^{(q)}\}_{i\in S_{\mathsf{st}}})$.
   iii. Compute $\widetilde{x}^{(q)} \leftarrow \mathsf{GarbleInp}(\mathsf{st}^{(q)}, x^{(q)})$.

(d) Run $\mathsf{BCPFE}'.\mathsf{sk}^{(q)} \leftarrow \mathsf{BCPFE.SimKG}'(\mathsf{BCPFE}'.\mathsf{st}, \mathsf{BCPFE}'.\mathsf{msk}, \widetilde{x}^{(q)}, (x^{(q)}, r^{(q)}))$.

(e) Return $\mathsf{sk}^{(q)} := (r^{(q)}, \mathsf{BCPFE.sk}^{(q)}, \mathsf{BCPFE}'.\mathsf{sk}^{(q)})$.

By a straightforward reduction to AD-SIM security of BCPFE' against dynamic bounded collusion, we obtain

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \le \mathrm{negl}(\lambda).$$

$\mathbf{Game}_{2.j}$: For $j \in [T]$, this game works similarly to $\mathbf{Game}_2$ except that each $\mathsf{BCPFE.ct}_{\mathsf{G}}$ is generated according to the "j-th circuit configuration". Formally, "Encryption with BCPFE", "Encryption by BCPFE'", and "Post-challenge key queries" phases are modified as follows where differences from $\mathbf{Game}_2$ in "Encryption by BCPFE'" and "Post-challenge key queries" phases are marked by red underlines. (We do not put red underlines for "Encryption by BCPFE" phase since that is completely modified.)

Encryption by BCPFE: Let $(\mathsf{conf}_1, \ldots, \mathsf{conf}_T)$ be the sequence that satisfy the requirements of Definition 3.5 for the circuit U_C. Let $\mathsf{conf}_j = \{\mathsf{mode}_j(\mathsf{G})\}_{\mathsf{G}\in\mathsf{Gates}}$, $I_j = \{\mathsf{G} \in \mathsf{Gates} : \mathsf{mode}_j(\mathsf{G}) = \mathsf{Gray}\}$, and $S_j(\mathsf{G}) = S_{\mathsf{mode}_j(\mathsf{G})}(\mathsf{G})$ as in Definition 3.5. Generate $\{\mathsf{BCPFE.ct}_{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}}$ as follows:

(a) For $i \in [N_{\mathsf{rand}}]$ and $q \in [Q_1]$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.

(b) For $\mathsf{G} \in I_j$ and $q \in [Q_1]$, run $\widetilde{\mathsf{G}}^{(q)} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i^{(q)}\}_{i\in[N_{\mathsf{rand}}]}, U_C, x^{(q)})$.

(c) Set $\mathcal{V}_{\mathsf{BCPFE}} := \left\{\left(\{\widetilde{\mathsf{G}}^{(q)}\}_{\mathsf{G}\in I_j}, (x^{(q)}, r^{(q)}), \mathsf{BCPFE.sk}^{(q)}\right)\right\}_{q\in[Q_1]}$.

(d) Run

$$(\{\mathsf{BCPFE.ct}_{\mathsf{G}}\}_{\mathsf{G}\in I_j}, \mathsf{BCPFE.st}) \leftarrow \mathsf{BCPFE.SimEnc}(\mathsf{BCPFE.mpk}, \mathcal{V}_{\mathsf{BCPFE}}, 1^{\mathsf{BCPFE.size}}, 1^Q).$$

(e) For $\mathsf{G} \in \mathsf{Gates} \setminus I_j$, run

$$\mathsf{BCPFE.ct}_{\mathsf{G}} \leftarrow \mathsf{BCPFE.Enc}(\mathsf{BCPFE.mpk}, \mathsf{GarbleCirc}[\mathsf{mode}_j(\mathsf{G}), \mathsf{pp}, \mathsf{G}, S_j(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_j(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}])$$

where $\mathsf{GarbleCirc}[\mathsf{mode}_j(\mathsf{G}), \mathsf{pp}, \mathsf{G}, S_j(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_j(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}]$ is the circuit as defined in Figure 5.

Encryption by BCPFE':

(a) For $q \in [Q_1]$, do the following:[16]

[16] Compared with $\mathbf{Game}_2$, the generation of $\mathsf{R}_i^{(q)}$ for $i \in S_{\mathsf{st}}$ is omitted since that is already generated in the "Encryption by BCPFE" phase as modifed above.

i. $\mathsf{st}^{(q)} := (\mathsf{pp}, \{\mathsf{R}_i^{(q)}\}_{i\in S_{\mathsf{st}}})$.
ii. Compute $\widetilde{x}^{(q)} \leftarrow \mathsf{SimInpConf}(\mathsf{st}^{(q)}, x^{(q)}, C(x^{(q)}), \mathsf{conf}_j)$.

(b) Set $\mathcal{V}_{\mathsf{BCPFE}'} := \left\{\left(\widetilde{x}^{(q)}, (x^{(q)}, r^{(q)}), \mathsf{BCPFE}'.\mathsf{sk}^{(q)}\right)\right\}_{q\in[Q_1]}$.

(c) Run

$$(\mathsf{BCPFE}'.\mathsf{ct}, \mathsf{BCPFE}'.\mathsf{st}) \leftarrow \mathsf{BCPFE}'.\mathsf{SimEnc}(\mathsf{BCPFE}.\mathsf{mpk}, \mathcal{V}_{\mathsf{BCPFE}'}, 1^{\mathsf{BCPFE}'.\mathsf{size}}, 1^{Q}).$$

Post-challenge key queries: Run $b \leftarrow \mathsf{A}^{\mathcal{O}_{2.j}(\mathsf{st},\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$ where $\mathcal{O}_{2.j}$ is an oracle that works as follows:

$\mathcal{O}_{2.j}$: On input $x^{(q)}$ for $q \in [Q_1+1, Q_2]$, do the following:

(a) Generate $r^{(q)} \leftarrow \{0,1\}^{\lambda}$.
(b) For $i \in [N_{\mathsf{rand}}]$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF}.\mathsf{Eval}(\mathsf{K}_i, r^{(q)})$.
(c) For $\mathsf{G} \in I_j$, run $\widetilde{\mathsf{G}}^{(q)} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i^{(q)}\}_{i\in[N_{\mathsf{rand}}]}, U_C, x^{(q)})$.
(d) Run $\mathsf{BCPFE}.\mathsf{sk}^{(q)} \leftarrow \mathsf{BCPFE}.\mathsf{SimKG}(\mathsf{BCPFE}.\mathsf{st}, \mathsf{BCPFE}.\mathsf{msk}, \{\widetilde{\mathsf{G}}^{(q)}\}_{\mathsf{G}\in I_j}, (x^{(q)}, r^{(q)}))$.
(e) Do the following:[17]
i. Set $\mathsf{st}^{(q)} := (\mathsf{pp}, \{\mathsf{R}_i^{(q)}\}_{i\in S_{\mathsf{st}}})$.
ii. Set $\widetilde{x}^{(q)} := \mathsf{SimInpConf}(\mathsf{st}^{(q)}, x^{(q)}, C(x^{(q)}), \mathsf{conf}_j)$.
(f) Run $\mathsf{BCPFE}'.\mathsf{sk}^{(q)} \leftarrow \mathsf{BCPFE}'.\mathsf{SimKG}(\mathsf{BCPFE}'.\mathsf{st}, \mathsf{BCPFE}'.\mathsf{msk}, \widetilde{x}^{(q)}, (x^{(q)}, r^{(q)}))$.
(g) Return $\mathsf{sk}^{(q)} := (r^{(q)}, \mathsf{BCPFE}.\mathsf{sk}^{(q)}, \mathsf{BCPFE}'.\mathsf{sk}^{(q)})$.

By Item 1 of Definition 3.5, $\mathsf{conf}_1 = \mathsf{White}^{|\mathsf{Gates}|}$ and thus $I_1 = \emptyset$. In this case, $\mathsf{BCPFE}.\mathsf{SimKG}(\mathsf{BCPFE}.\mathsf{st}, \mathsf{BCPFE}.\mathsf{msk}, \{\widetilde{\mathsf{G}}^{(q)}\}_{\mathsf{G}\in I_j}, (x^{(q)}, r^{(q)}))$ means $\mathsf{BCPFE}.\mathsf{KeyGen}(\mathsf{BCPFE}.\mathsf{msk}, (x^{(q)}, r^{(q)}))$ as defined in Definition 4.2. Moreover, $\mathsf{SimInpConf}(\mathsf{st}^{(q)}, x^{(q)}, C(x^{(q)}), \mathsf{White}^{|\mathsf{Gates}|})$ is equivalent to $\mathsf{GInp}(\mathsf{st}^{(q)}, x^{(q)})$ as required in Item 2 of Definition 3.4. Therefore, we have

$$\Pr[\mathcal{E}_2] = \Pr[\mathcal{E}_{2.1}].$$

We prove

$$|\Pr[\mathcal{E}_{2.j}] - \Pr[\mathcal{E}_{2.j+1}]| \leq \mathrm{negl}(\lambda).$$

for all $j \in [T-1]$ in Lemma 4.2.

$\mathbf{Game}_3$: This game works similarly to $\mathbf{Game}_{2.T}$ except that whenever $\mathsf{PRF}(\mathsf{K}_i, \cdot)$ for $i \in [N_{\mathsf{rand}}] \setminus \cup_{\mathsf{G}\in\mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G})$ is invoked, the output is replaced with a uniformly random value.
Since $\mathsf{conf}_T = \mathsf{Black}^{|\mathsf{Gates}|}$ by Item 1 of Definition 3.5, K_i for $i \in [N_{\mathsf{rand}}] \setminus \cup_{\mathsf{G}\in\mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G})$ is not used as a hardwired value of a circuit encrypted by BCPFE or BCPFE′, and only used for deriving PRF values. Moreover, by the modification made in $\mathbf{Game}_1$, each PRF is invoked on the same input at most once or otherwise the game just outputs a uniformly random bit. Based on this observation, by a straightforward reduction to the security of PRF, we have

$$|\Pr[\mathcal{E}_{2.T}] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_4$: This game works similarly to $\mathbf{Game}_3$ except that $\widetilde{x}^{(q)}$ is generated as follows both in "Encryption by BCPFE′" and "Post-challenge key queries" phases:

$$\widetilde{x}^{(q)} \leftarrow \mathsf{SimInp}(\mathsf{st}^{(q)}, C(x^{(q)})).$$

Note that $\{\mathsf{R}_i^{(q)}\}_{i\in[N_{\mathsf{rand}}]\setminus\cup_{\mathsf{G}\in\mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G})}$ is freshly sampled for each q and only used for generating $\widetilde{x}^{(q)}$ by the modification made in $\mathbf{Game}_2$. Then, by a straightforward reduction to Item 3 of Definition 3.4, we obtain

$$\Pr[\mathcal{E}_3] = \Pr[\mathcal{E}_4].$$

$\mathbf{Game}_5$: This game works similarly to $\mathbf{Game}_4$ except that for each $\mathsf{G} \in \mathsf{Gates}$, $\mathsf{BCPFE}.\mathsf{ct}_{\mathsf{G}}$ is generated as follows:

$$\mathsf{BCPFE}.\mathsf{ct}_{\mathsf{G}} \leftarrow \mathsf{BCPFE}.\mathsf{Enc}(\mathsf{BCPFE}.\mathsf{mpk}, \mathsf{GarbleCirc}'[\mathsf{Black}, \mathsf{pp}, \mathsf{top}(\mathsf{G}), S_{\mathsf{Black}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}])$$

[17] A similar remark to Footnote 16 applies.

where $\mathsf{GarbleCirc}'[\mathsf{Black}, \mathsf{pp}, \mathsf{top}(\mathsf{G}), S_{\mathsf{Black}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}]$ is identical to $\mathsf{GarbleCirc}[\mathsf{Black}, \mathsf{pp}, \mathsf{G}, S_{\mathsf{Black}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}]$ except that it generates $\widetilde{\mathsf{G}}$ as $\widetilde{\mathsf{G}} := \mathsf{SimGate}'_{\mathsf{Black}}(\mathsf{pp}, \mathsf{top}(\mathsf{G}), \{\mathsf{R}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}; \mathsf{R}'_{\mathsf{G}})$. instead of using $\mathsf{SimGate}_{\mathsf{Black}}$. (See Item 3a of Definition 3.4 for the definition of $\mathsf{SimGate}'_{\mathsf{Black}}$.) We remark that we only have to embed $\mathsf{top}(\mathsf{G})$ instead of G into $\mathsf{GarbleCirc}'$ since $\mathsf{SimGate}'_{\mathsf{Black}}$ takes $\mathsf{top}(\mathsf{G})$ instead of G as an input.
As required in Item 3a of Definition 3.4, $\mathsf{SimGate}'_{\mathsf{Black}}(\mathsf{pp}, \mathsf{top}(\mathsf{G}), \{\mathsf{R}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}; \mathsf{R}'_{\mathsf{G}})$ is equivalent to $\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}; \mathsf{R}'_{\mathsf{G}})$. Therefore, by a straightforward reduction to AD-SIM security of BCPFE against dynamic bounded collusion, we obtain

$$|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_5]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_6$: This game works similarly to $\mathbf{Game}_5$ except that we reverse the modifications made in $\mathbf{Game}_1$ and $\mathbf{Game}_3$. That is, the game does not output a uniformly random bit even if there are $q \neq q'$ such that $r^{(q)} = r^{(q')}$ and every $\mathsf{R}_i^{(q)}$ is generated as $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$ for $i \in [N_{\mathsf{rand}}]$ and $q \in [Q_1 + Q_2]$.
Similarly to the game hops from $\mathbf{Game}_0$ to $\mathbf{Game}_1$ and from $\mathbf{Game}_{2.T}$ to $\mathbf{Game}_3$, by a straightforward reduction to the security of PRF, we obtain

$$|\Pr[\mathcal{E}_5] - \Pr[\mathcal{E}_6]| \leq \mathrm{negl}(\lambda).$$

Moreover, $\mathbf{Game}_6$ is identical to $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{A},\mathsf{Sim}}(1^\lambda)$ for the simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ that works as follows:

$\mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}, 1^{|C|}, 1^Q)$: On input $\mathsf{mpk} = (\mathsf{BCPFE.mpk}, \mathsf{BCPFE}'.\mathsf{mpk})$, $\mathcal{V} = \left\{\left(C(x^{(q)}), x^{(q)}, \mathsf{sk}^{(q)}\right)\right\}_{q\in[Q_1]}$, $1^{|C|}$, and 1^Q, do the following:

1. For $q \in [Q_1]$, parse $\mathsf{sk}^{(q)} = (r^{(q)}, \mathsf{BCPFE.sk}^{(q)}, \mathsf{BCPFE}'.\mathsf{sk}^{(q)})$.
2. Run $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(U_C))$. We note that this can be done because $\mathsf{top}(U_C)$ only depends on $|C|$.
3. For $i \in [N_{\mathsf{rand}}]$, generate $\mathsf{K}_i \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
4. For $\mathsf{G} \in \mathsf{Gates}$, generate $\mathsf{K}'_{\mathsf{G}} \leftarrow \mathsf{PRF}'.\mathsf{Setup}(1^\lambda)$.[18]
5. For $i \in [N_{\mathsf{rand}}]$ and $q \in [Q_1]$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.
6. For $\mathsf{G} \in \mathsf{Gates}$, run

$$\mathsf{BCPFE.ct}_{\mathsf{G}} \leftarrow \mathsf{BCPFE.Enc}(\mathsf{BCPFE.mpk}, \mathsf{GarbleCirc}'[\mathsf{Black}, \mathsf{pp}, \mathsf{top}(\mathsf{G}), S_{\mathsf{Black}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}])$$

   where $\mathsf{GarbleCirc}'[\mathsf{Black}, \mathsf{pp}, \mathsf{top}(\mathsf{G}), S_{\mathsf{Black}}(\mathsf{G}), \{\mathsf{K}_i\}_{i\in S_{\mathsf{Black}}(\mathsf{G})}, \mathsf{K}'_{\mathsf{G}}]$ is the circuit as defined in the description of $\mathbf{Game}_5$. We remark that this can be done without knowing C since $S_{\mathsf{Black}}(\mathsf{G})$ is efficiently computable from the topology of U_C, which does not depend on C, as required in Item 3a of Definition 3.4.
7. For $q \in [Q_1]$, do the following:
   (a) $\mathsf{st}^{(q)} := (\mathsf{pp}, \{\mathsf{R}_i^{(q)}\}_{i\in S_{\mathsf{st}}})$.
   (b) Compute $\widetilde{x}^{(q)} \leftarrow \mathsf{SimInp}(\mathsf{st}^{(q)}, C(x^{(q)}))$.
8. Set $\mathcal{V}_{\mathsf{BCPFE}'} := \left\{\left(\widetilde{x}^{(q)}, (x^{(q)}, r^{(q)}), \mathsf{BCPFE}'.\mathsf{sk}^{(q)}\right)\right\}_{q\in[Q_1]}$.
9. Run

$$(\mathsf{BCPFE}'.\mathsf{ct}, \mathsf{BCPFE}'.\mathsf{st}) \leftarrow \mathsf{BCPFE}'.\mathsf{SimEnc}(\mathsf{BCPFE}'.\mathsf{mpk}, \mathcal{V}_{\mathsf{BCPFE}'}, 1^{\mathsf{BCPFE}'.\mathsf{size}}, 1^Q).$$

10. Output $\mathsf{ct} := (\{\mathsf{BCPFE.ct}_{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}}, \mathsf{BCPFE}'.\mathsf{ct})$ and $\mathsf{st} := (\mathsf{pp}, \{\mathsf{K}_i\}_{i\in[N_{\mathsf{rand}}]}, \mathsf{BCPFE}'.\mathsf{st})$.

$\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, C(x), x)$: On input $\mathsf{st} = (\mathsf{pp}, \{\mathsf{K}_i\}_{i\in[N_{\mathsf{rand}}]}, \mathsf{BCPFE}'.\mathsf{st})$, $\mathsf{msk} = (\mathsf{BCPFE.msk}, \mathsf{BCPFE.msk}')$, $C(x)$, and x, do the following:

1. Generate $r \leftarrow \{0,1\}^\lambda$.
2. For $i \in [N_{\mathsf{rand}}]$, compute $\mathsf{R}_i \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r)$.
3. Run $\mathsf{BCPFE.sk} \leftarrow \mathsf{BCPFE.KeyGen}(\mathsf{BCPFE.msk}, (x, r))$.
4. Set $\mathsf{st} \leftarrow (\mathsf{pp}, \{\mathsf{R}_i\}_{i\in S_{\mathsf{st}}})$.
5. Compute $\widetilde{x} \leftarrow \mathsf{SimInp}(\mathsf{st}, C(x))$.
6. Run $\mathsf{BCPFE}'.\mathsf{sk} \leftarrow \mathsf{BCPFE.SimKG}'(\mathsf{BCPFE}'.\mathsf{st}, \mathsf{BCPFE}'.\mathsf{msk}, \widetilde{x}, (x, r))$.

[18] Strictly speaking, this description is not rigorous since the set Gates cannot be defined without fixing C. By abuse of notation, we write Gates to mean the set of topology of gates of U_C, which can be computed only from $|C|$.

7. Output $\mathsf{sk} := (r, \mathsf{BCPFE.sk}, \mathsf{BCPFE}'.\mathsf{sk})$.

Therefore, what is left is to prove the following lemma.

Lemma 4.2. *For all $j \in [T-1]$, it holds that*

$$|\Pr[\mathcal{E}_{2.j}] - \Pr[\mathcal{E}_{2.j+1}]| \leq \mathrm{negl}(\lambda).$$

Proof. By Item 3 of Definition 3.5, there is exactly one $\mathsf{G} \in \mathsf{Gates}$, which we denote by G^*_j, such that $\mathsf{mode}_j(\mathsf{G}^*_j) \neq \mathsf{mode}_{j+1}(\mathsf{G}^*_j)$. Moreover, either $\mathsf{mode}_j(\mathsf{G}^*_j)$ or $\mathsf{mode}_{j+1}(\mathsf{G}^*_j)$ is Gray. In the following, we prove Lemma 4.2 assuming that $\mathsf{mode}_{j+1}(\mathsf{G}^*) = \mathsf{Gray}$ (and thus $\mathsf{mode}_j(\mathsf{G}^*) \neq \mathsf{Gray}$). Remark that we have $I_{j+1} = I_j \cup \{\mathsf{G}^*_j\}$ and $I_j \cup I_{j+1} = I_{j+1}$ in this case. The proof of the other case can be obtained by simply swapping the roles of conf_j and conf_{j+1}.

For proving the lemma, we consider further hybrid games between $\mathbf{Game}_{2.j}$ and $\mathbf{Game}_{2.j+1}$ as follows.

$\mathbf{Game}_{2.j.a}$: This game is identical to $\mathbf{Game}_{2.j}$ except that $\mathsf{BCPFE.ct}_{\mathsf{G}^*_j}$ is generated by the simulator rather than the real encryption algorithm, but the output values are programmed to be a ciphertext in the original mode. Formally, "Encryption by BCPFE" and "Post-challenge key queries" phases are modified as follows where differences from $\mathbf{Game}_{2.j}$ are marked by red underlines:

Encryption by BCPFE: Let $(\mathsf{conf}_1, \ldots, \mathsf{conf}_T)$ be the sequence that satisfies the requirements of Definition 3.5 for the circuit U_C. Let $\mathsf{conf}_j = \{\mathsf{mode}_j(\mathsf{G})\}_{\mathsf{G} \in \mathsf{Gates}}$, $I_j = \{\mathsf{G} \in \mathsf{Gates} : \mathsf{mode}_j(\mathsf{G}) = \mathsf{Gray}\}$, and $S_j(\mathsf{G}) = S_{\mathsf{mode}_j(\mathsf{G})}(\mathsf{G})$ as in Definition 3.5. Generate $\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G} \in \mathsf{Gates}}$ as follows:

(a) For $i \in [N_{\mathsf{rand}}]$ and $q \in [Q_1]$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.

(b) For $\mathsf{G} \in I_j$ and $q \in [Q_1]$, run $\widetilde{\mathsf{G}}^{(q)} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i^{(q)}\}_{i \in [N_{\mathsf{rand}}]}, U_C, x^{(q)})$.

(c) Run $\widetilde{\mathsf{G}^*_j}^{(q)} \leftarrow \mathsf{GarbleCirc}[\mathsf{mode}_j(\mathsf{G}^*_j), \mathsf{pp}, \mathsf{G}^*_j, S_j(\mathsf{G}^*_j), \{\mathsf{K}_i\}_{i \in S_j(\mathsf{G}^*_j)}, \mathsf{K}'_{\mathsf{G}^*_j}](x^{(q)}, r^{(q)})$. Specifically, do the following:

i. Compute $\mathsf{R}'^{(q)}_{\mathsf{G}^*_j} \leftarrow \mathsf{PRF}'.\mathsf{Eval}(\mathsf{K}'_{\mathsf{G}^*_j}, r^{(q)})$.

ii. Run $\widetilde{\mathsf{G}^*_j}^{(q)} \leftarrow \mathsf{SimGate}_{\mathsf{mode}_j(\mathsf{G}^*_j)}(\mathsf{pp}, \mathsf{G}^*_j, \{\mathsf{R}_i^{(q)}\}_{i \in S_j(\mathsf{G}^*_j)}; \mathsf{R}'^{(q)}_{\mathsf{G}^*_j})$.

(d) Set $\mathcal{V}_{\mathsf{BCPFE}} := \left\{ \left(\{\widetilde{\mathsf{G}}^{(q)}\}_{\mathsf{G} \in I_{j+1}}, (x^{(q)}, r^{(q)}), \mathsf{BCPFE.sk}^{(q)} \right) \right\}_{q \in [Q_1]}$.

(e) Run

$$(\{\mathsf{BCPFE.ct}_\mathsf{G}\}_{\mathsf{G} \in I_{j+1}}, \mathsf{BCPFE.st}) \leftarrow \mathsf{BCPFE.SimEnc}(\mathsf{BCPFE.mpk}, \mathcal{V}_{\mathsf{BCPFE}}, 1^{\mathsf{BCPFE.size}}, 1^Q).$$

(f) For $\mathsf{G} \in \mathsf{Gates} \setminus I_{j+1}$, run

$$\mathsf{BCPFE.ct}_\mathsf{G} \leftarrow \mathsf{BCPFE.Enc}(\mathsf{BCPFE.mpk}, \mathsf{GarbleCirc}[\mathsf{mode}_j(\mathsf{G}), \mathsf{pp}, \mathsf{G}, S_j(\mathsf{G}), \{\mathsf{K}_i\}_{i \in S_j(\mathsf{G})}, \mathsf{K}'_\mathsf{G}])$$

where $\mathsf{GarbleCirc}[\mathsf{mode}_j(\mathsf{G}), \mathsf{pp}, \mathsf{G}, S_{\mathsf{White}}(\mathsf{G}), \{\mathsf{K}_i\}_{i \in S_{\mathsf{White}}(\mathsf{G})}, \mathsf{K}'_\mathsf{G}]$ is the circuit as defined in Figure 5.

Post-challenge key queries: Run $b \leftarrow \mathsf{A}^{\mathcal{O}_{2.j.a}(\mathsf{st}, \mathsf{msk}, \cdot)}(\mathsf{mpk}, \mathsf{ct})$ where $\mathcal{O}_{2.j.a}$ is an oracle that works as follows:

$\mathcal{O}_{2.j.a}$: On input $x^{(q)}$ for $q \in [Q_1 + 1, Q_2]$, do the following:

(a) Generate $r^{(q)} \leftarrow \{0,1\}^\lambda$.

(b) For $i \in [N_{\mathsf{rand}}]$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.

(c) For $\mathsf{G} \in I_j$, run $\widetilde{\mathsf{G}}^{(q)} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_i\}_{i \in [N_{\mathsf{rand}}]}, U_C, x^{(q)})$.

(d) Run $\widetilde{\mathsf{G}^*_j}^{(q)} \leftarrow \mathsf{GarbleCirc}[\mathsf{mode}_j(\mathsf{G}^*_j), \mathsf{pp}, \mathsf{G}^*_j, S_j(\mathsf{G}^*_j), \{\mathsf{K}_i\}_{i \in S_j(\mathsf{G}^*_j)}, \mathsf{K}'_{\mathsf{G}^*_j}](x^{(q)}, r^{(q)})$. Specifically, do the following:

i. Compute $\mathsf{R}'^{(q)}_{\mathsf{G}^*_j} \leftarrow \mathsf{PRF}'.\mathsf{Eval}(\mathsf{K}'_{\mathsf{G}^*_j}, r^{(q)})$.

ii. Run $\widetilde{\mathsf{G}^*_j}^{(q)} \leftarrow \mathsf{SimGate}_{\mathsf{mode}_j(\mathsf{G}^*_j)}(\mathsf{pp}, \mathsf{G}^*_j, \{\mathsf{R}_i^{(q)}\}_{i \in S_j(\mathsf{G}^*_j)}; \mathsf{R}'^{(q)}_{\mathsf{G}^*_j})$.

(e) Run $\mathsf{BCPFE.sk}^{(q)} \leftarrow \mathsf{BCPFE.SimKG}(\mathsf{BCPFE.st}, \mathsf{BCPFE.msk}, \{\widetilde{\mathsf{G}}^{(q)}\}_{\mathsf{G}\in I_{j+1}}, (x^{(q)}, r^{(q)}))$.
(f) Do the following:
  i. For $i \in S_{\mathsf{st}}$, compute $\mathsf{R}_i^{(q)} \leftarrow \mathsf{PRF.Eval}(\mathsf{K}_i, r^{(q)})$.
  ii. Set $\mathsf{st}^{(q)} := (\mathsf{pp}, \{\mathsf{R}_i^{(q)}\}_{i\in S_{\mathsf{st}}})$.
  iii. Set $\widetilde{x}^{(q)} := \mathsf{SimInpConf}(\mathsf{st}^{(q)}, x^{(q)}, C(x^{(q)}), \mathsf{conf}_j)$.
(g) Run $\mathsf{BCPFE'.sk}^{(q)} \leftarrow \mathsf{BCPFE'.SimKG}(\mathsf{BCPFE'.st}, \mathsf{BCPFE'.msk}, \widetilde{x}^{(q)}, (x^{(q)}, r^{(q)}))$.
(h) Return $\mathsf{sk}^{(q)} := (r^{(q)}, \mathsf{BCPFE.sk}^{(q)}, \mathsf{BCPFE'.sk}^{(q)})$.

By a straightforward reduction to M-CT AD-SIM-security of BCPFE against dynamic bounded collusion, we have

$$|\Pr[\mathcal{E}_{2.j}] - \Pr[\mathcal{E}_{2.j.a}]| = \mathrm{negl}(\lambda).$$

$\mathbf{Game}_{2.j.b}$: This game is identical to $\mathbf{Game}_{2.j.a}$ except that whenever $\mathsf{PRF}(\mathsf{K}_i, \cdot)$ for $i \in [N_{\mathsf{rand}}] \setminus V_j$ or $\mathsf{PRF}(\mathsf{K}'_{\mathsf{G}_j^*}, \cdot)$ is invoked, the output is replaced with a uniformly random value where

$$V_j := \bigcup_{\mathsf{G}\in\mathsf{Gates}\setminus I_{j+1}} S_j(\mathsf{G})$$

as in Item 4 of Definition 3.5. (Note that we assume $I_j \cup I_{j+1} = I_{j+1}$.)
By the definition of the game, we can see that K_i for $i \in [N_{\mathsf{rand}}] \setminus V_j$ or $\mathsf{K}'_{\mathsf{G}_j^*}$ is not used as a hardwired value of a circuit encrypted by BCPFE or BCPFE$'$, and only used for deriving PRF values. Moreover, by the modification made in $\mathbf{Game}_1$, each PRF is invoked on the same input at most once or otherwise the game just outputs a uniformly random bit. Based on this observation, by a straightforward reduction to the security of PRF and PRF$'$, we have

$$|\Pr[\mathcal{E}_{2.j.a}] - \Pr[\mathcal{E}_{2.j.b}]| = \mathrm{negl}(\lambda).$$

$\mathbf{Game}_{2.j.c}$: This game is identical to $\mathbf{Game}_{2.j.b}$ except that $\widetilde{\mathsf{G}}_j^{*(q)}$ and $\widetilde{x}^{(q)}$ are generated according to the configuration conf_{j+1}. That is, they are generated as follows:

$$\widetilde{\mathsf{G}}_j^{*(q)} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}_j^*, \{\mathsf{R}_i\}_{i\in[N_{\mathsf{rand}}]}, U_C, x^{(q)}),$$
$$\widetilde{x}^{(q)} := \mathsf{SimInpConf}(\mathsf{st}^{(q)}, x^{(q)}, C(x^{(q)}), \mathsf{conf}_{j+1}).$$

We remark that this modification applies to both cases of $q \in [Q_1]$ and $q \in [Q_1+1, Q_1+Q_2]$.
Due to the modification made in $\mathbf{Game}_{2.j.b}$, $\mathsf{R}_i^{(q)}$ for $i \in [N_{\mathsf{rand}}] \setminus V_j$ and $\mathsf{R}'^{(q)}_{\mathsf{G}_j^*}$ are freshly sampled for each q. Therefore, by a straightforward reduction to Item 4 of Definition 3.5 with a standard hybrid argument, we have

$$|\Pr[\mathcal{E}_{2.j.b}] - \Pr[\mathcal{E}_{2.j.c}]| = \mathrm{negl}(\lambda).$$

Moreover, it is easy to see that $\mathbf{Game}_{2.j.c}$ is identical to $\mathbf{Game}_{2.j+1}$ if we reverse the modification made in $\mathbf{Game}_{2.j.b}$. Therfore, by a straightforward reduction to the security of PRF and PRF$'$, we have

$$|\Pr[\mathcal{E}_{2.j.c}] - \Pr[\mathcal{E}_{2.j+1}]| = \mathrm{negl}(\lambda).$$

This completes the proof of Lemma 4.2, which completes the proof of Theorem 4.1.

Necessity of adaptive garbling. Our CPFE scheme CPFE for unbounded polynomial-size circuits relies on the existence of LOT in addition to IBE. On the other hand, [AMVY21] gave NA-SIM-secure CPFE scheme for unbounded polynomial-size circuits solely based on IBE. Then a natural question is if we can achieve the AD-SIM security only from IBE without relying on additional assumptions. We observe this is difficult since even 1-key AD-SIM CPFE scheme for unbounded polynomial-size circuits immediately implies adaptive garbling whose online-efficiency does not depend on the circuit being garbled (as long as the size is polynomial in the security parameter) by considering the combination of the setup and encryption algorithms as a circuit garbling algorithm and the key generation algorithm as an input garbling algorithm. Therefore, the additional assumption is unavoidable unless we find a construction of adaptive garbling with circuit-independent online-efficiency from IBE, which we believe is quite challenging.

FE for Turing machines. Agrawal et al. [AMVY21] (implicitly) showed that one can construct FE for TM with AD-SIM security against dynamic bounded collusion based on CPFE for unbounded polynomial-size circuits with AD-SIM security against dynamic bounded collusion additionally assuming sub-exponential LWE. Since (even polynomial) LWE implies LOT and IBE, by combining their result and Theorem 4.1, we obtain the following theorem:

Theorem 4.2. *Assuming sub-exponential* LWE*, we have FE for TM with* AD-SIM *security against dynamic bounded collusion.*

This improves one of the main results of [AMVY21] that constructed a similar scheme with NA-SIM security based on the same assumption. Since this is further improved in regard to assumptions in Section 5, we omit the details.

5 TMFE without Succinct FE

In this section, we propose an alternative route to construct FE for Turing machine that does not use succinct FE. We refer to Section 1 for the overview.

5.1 FE for Laconic OT Functionality

Here, we define FE for LOT functionality by specifying the relation $R_{\mathsf{LOTFE}} : \mathcal{X}_{\mathsf{LOTFE}} \times \mathcal{Y}_{\mathsf{LOTFE}} \to \{0,1\}^*$.

FE for Laconic OT Functionality. To define FE for LOT functionality, we set $\mathsf{prm} = \bot$, $\mathcal{X}_{\mathsf{LOTFE}} = \mathbb{N} \times \mathbb{N} \times \{0,1\}^* \times \{0,1\}^*$, and $\mathcal{Y}_{\mathsf{LOTFE}} = \{0,1\}^*$. An element in $\mathcal{X}_{\mathsf{LOTFE}}$ is represented by (N, i, μ_0, μ_1) with $N \in \mathbb{N}$, $i \in [N]$, and $\mu_0, \mu_1 \in \{0,1\}^*$ with $|\mu_0| = |\mu_1|$. We assume that both i and N are represented in binary form. We then define

$$R_{\mathsf{LOTFE}}((N, i, \mu_0, \mu_1), D) = \begin{cases} (N, \mu_{D[i]}) & \text{if } |D| = N \\ (N, 1^{|\mu_0|}) & \text{otherwise} \end{cases},$$

where $|D|$ is the length of D as a binary string and $D[i]$ is the i-th bit of D.

Remark 5.1 (Succinctness). We note that the encryption algorithm should run in fixed polynomial time in λ that is independent from N, since N is input to the encryption algorithm in binary form. This in particular implies that the running time of the encryption algorithm is independent from the size of the database D supported by the scheme. This property can be seen as an analogue of the efficiency requirements for the succinctness of FE [GTKP+13b] or laconic OT [CDG+17].

Remark 5.2. We also note that the above FE has similar functinality to that of LOT [CDG+17]. However, the important difference is that we intend to hide the index i while they do not. This security requirement is captured by the definition of R_{LOTFE} above, where i is not part of the output.

Ingredients. We now describe the underlying building blocks used for our construction of FE for LOT functionality. We need the following ingredients.

1. PIR scheme $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Answer}, \mathsf{PIR.Reconstruct})$ that satisfies the efficiency requirements in Definition 2.9. In particular, we require that PIR.Query and PIR.Reconstruct run in fixed polynomial time for any $N \leq 2^{\log^2 \lambda}$ (even for super-polynomial N). This implies that the lengths of PIR.query, PIR.answer, and PIR.st are bounded by a fixed polynomial in the security parameter that is independent of N. We use the uniform upper bound $\ell_{\mathsf{PIR}} = \mathrm{poly}(\lambda)$ for them and assume that they are represented by binary strings of length ℓ_{PIR}. Additionally, we require that the function PIR.Answer has shallow circuit implementations. As we show in Appendix C, we have the following instantiations:
   (a) PIR constructions from (the polynomial hardness of) LWE [BV11, GSW13] has implementation of the answer function in NC.
   (b) For PIR constructions from DDH/QR [DGI+19], we have implementations of the answer function in NC^1. For DDH based construction, we have to use the multiplicative sub-group of $\mathbb{Z}_q$ for prime q.

2. 1-Sel-IND secure ABE scheme $\mathsf{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ for circuits that can evaluate the answer function of PIR in the above. We consider two types of instantiations:
   (a) We can use general ABE for circuit [GVW13, BGG+14]. If the answer function of PIR is implemented in NC, we can base the security of the scheme on polynomial hardness of LWE with quasi-polynomial approximation factors (i.e., $O(\lambda^{\mathrm{poly}(\log \lambda)})$).
   (b) We can use ABE for NC^1 circuits. In more details, we need the scheme to support circuit with fixed input length and any depth d, where we allow the key generation algorithm and the decryption algorithms to run in time $\mathrm{poly}(\lambda, 2^d)$. This effectively limits the class of the circuits to be NC^1. We can instantiate such an ABE from (the polynomial hardness of) LWE with super-polynomial approximation factor [GV15] (i.e., $O(\lambda^{\omega(1)})$) or various assumptions on pairing groups including DBDH or CBDH (the computational bilinear Diffie-Hellman assumption) [GPSW06].
3. Selectively secure garbled circuit $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Sim})$. We can instantiate it from any one-way function [Yao86].
4. IBE scheme $\mathsf{IBE} = (\mathsf{IBE.Setup}, \mathsf{IBE.KeyGen}, \mathsf{IBE.Enc}, \mathsf{IBE.Dec})$ with AD-IND security. Since IBE with AD-IND security is implied by IBE with Sel-IND [DG17a] and the latter is trivially implied by the ABE for NC^1 circuit, this does not add a new assumption.

We assume that all the ingredients above have perfect correctness for simplicity. We consider correctness error only for PIR, because DDH based instantiation does not have perfect correctness.

Construction. Here, we describe our scheme $\mathsf{LOTFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The construction is similar to that of succinct FE by [GTKP+13b] where FHE is replaced by PIR. In the construction, we assume that $N, |D| \leq 2^{\log^2 \lambda}$. This is sufficient for dealing with unbounded size D, because $2^{\log^2 \lambda} = \lambda^{\log \lambda}$ is super-polynomial.

$\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, do the following:
1. Run $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$.
2. Run $(\mathsf{ABE.mpk}, \mathsf{ABE.msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, \mathsf{prm})$, where $\mathsf{prm} := 1^{\ell_{\mathsf{PIR}}+2\lambda}$. This means that the ABE supports circuit with input length $\ell_{\mathsf{PIR}} + 2\lambda$.
3. Output $\mathsf{mpk} := (\mathsf{ABE.mpk}, \mathsf{IBE.mpk})$ and $\mathsf{msk} := (\mathsf{ABE.msk}, \mathsf{IBE.msk})$.

$\mathsf{Enc}(\mathsf{mpk}, (N, i, \mu_0, \mu_1))$: On input the master public key $\mathsf{mpk} = (\mathsf{ABE.mpk}, \mathsf{IBE.mpk})$ and the message (N, i, μ_0, μ_1), do the following:
1. Run $(\mathsf{PIR.query}, \mathsf{PIR.st}) \leftarrow \mathsf{PIR.Query}(1^\lambda, i, N)$.
2. Pick $\mathsf{lab}_{k,b} \leftarrow \{0,1\}^\lambda$ for $k \in [\ell_{\mathsf{PIR}}]$, $b \in \{0,1\}$.
3. For $k \in [\ell_{\mathsf{PIR}}]$, $b \in \{0,1\}$, compute
$$\mathsf{ABE.ct}_{k,b} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, k, b), \mathsf{lab}_{k,b}).$$
4. Construct circuit $C[N, \mathsf{PIR.st}, \mu_0, \mu_1]$ as Figure 7.
5. Run $\widetilde{C} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, C[N, \mathsf{PIR.st}, \mu_0, \mu_1]\right)$.
6. Set $\mathsf{msg} := \left(\widetilde{C}, \mathsf{PIR.query}, \{\mathsf{ABE.ct}_{k,b}\}_{k\in[\ell_{\mathsf{PIR}}], b\in\{0,1\}}\right)$.
7. Run $\mathsf{IBE.ct} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, N, \mathsf{msg})$.
8. Output $\mathsf{ct} := (N, 1^{|\mu_0|}, \mathsf{IBE.ct})$.

$\mathsf{KeyGen}(\mathsf{msk}, D)$: On input the master secret key $\mathsf{msk} = (\mathsf{ABE.msk}, \mathsf{IBE.msk})$, an input $D \in \{0,1\}^*$ with $|D| \leq 2^{\log^2 \lambda}$, do the following:
1. Construct the circuit $F[D]$ as Figure 8.
2. Run
$$\mathsf{ABE.sk} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.msk}, F[D]).$$
3. Run $\mathsf{IBE.sk} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, |D|)$.
4. Output $\mathsf{sk} := (D, \mathsf{ABE.sk}, \mathsf{IBE.sk})$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$: On input a ciphertext $\mathsf{ct} = (N, 1^{|\mu_0|}, \mathsf{IBE.ct})$, a secret key $\mathsf{sk} = (D, \mathsf{ABE.sk}, \mathsf{IBE.sk})$, do the following:
1. If $|D| \neq N$, output $(N, 1^{|\mu_0|})$.
2. Run $\mathsf{msg} \leftarrow \mathsf{IBE.Dec}(\mathsf{IBE.sk}, \mathsf{IBE.ct})$.

Circuit $C[N, \mathsf{PIR.st}, \mu_0, \mu_1]$

Hardwired constants: The integer N, The PIR secret state $\mathsf{PIR.st}$, the messages μ_0, μ_1.
Input: String $X \in \{0,1\}^{\ell_{\mathsf{PIR}}}$

1. Parse $X \to \mathsf{PIR.answer}$.
2. Run $\mathsf{PIR.Reconstruct}(\mathsf{PIR.st}, \mathsf{PIR.answer}, N) \to b$.
3. Output μ_b.

Fig. 7 Circuit $C[N, \mathsf{PIR.st}, \mu_0, \mu_1]$

Circuit $F[D]$

Hardwired constants: The data $D \in \{0,1\}^{|D|}$.
Input: String $X \in \{0,1\}^{\ell_{\mathsf{PIR}}+\lambda+1}$

1. Parse the input as $X \to (\mathsf{PIR.query}, k, b)$ where $\mathsf{PIR.query} \in \{0,1\}^{\ell_{\mathsf{PIR}}}$, $k \in \{0,1\}^{\lambda}$, and $b \in \{0,1\}$.
2. If $k \notin [\ell_{\mathsf{PIR}}]$, output 0, where k is interpreted as an integer.
3. Run $\mathsf{PIR.Answer}(\mathsf{PIR.query}, D, 1^{|D|}) \to \mathsf{PIR.answer}$.
4. Compute $b' = \mathsf{PIR.answer}_k \oplus b \oplus 1$, where $\mathsf{PIR.answer}_k$ is the k-th bit of $\mathsf{PIR.answer}$.
5. Output b'.

Fig. 8 Circuit $F[D]$

3. Parse $\mathsf{msg} \to (\widetilde{C}, \mathsf{PIR.query}, \{\mathsf{ABE.ct}_{k,b}\}_{k\in[n],b\in\{0,1\}})$.
4. Run $\mathsf{PIR.answer} \leftarrow \mathsf{PIR.Answer}(\mathsf{PIR.query}, D, 1^N)$.
5. Run $\mathsf{lab}_k \leftarrow \mathsf{ABE.Dec}(\mathsf{ABE.sk}, \mathsf{ABE.ct}_{k,\mathsf{PIR.answer}_k})$ for $k \in [\ell_{\mathsf{PIR}}]$.
6. Compute $\mu := \mathsf{GC.Eval}(\widetilde{C}, \{\mathsf{lab}_k\}_k)$.
7. Output (N, μ).

Efficiency. We discuss the efficiency of the scheme. It is not hard to see that Setup and Enc run in polynomial time in its input length. We note that Enc runs in polynomial time in λ and $|\mu|$ even for super-polynomial N as large as $2^{\log^2 \lambda}$ by the efficiency property of PIR.Query and PIR.Reconstruct. For evaluating the efficiency of KeyGen, we consider two settings based on how we instantiate ABE and PIR. The first case is the combination of ABE for circuits and any PIR, whereas the second case is ABE for NC^1 circuits and PIR with answer function in NC^1. We focus on the latter case since the former case is much simpler. Evaluating the efficiency of KeyGen in this case is a bit subtle, because ABE.KeyGen used inside the algorithm runs in exponential time in the depth of the input circuit. In order to bound the running time of the algorithm, we evaluate the depth of $F[D]$ by going over all the computation steps inside the circuit. We observe that only the second and the third steps out of the five steps are non-trivial. The second step can be implemented by a circuit of depth $O(\log \ell_{\mathsf{PIR}}) = O(\log \lambda)$ by checking $k \stackrel{?}{=} i$ for all $i \in [\ell_{\mathsf{PIR}}]$ in parallel and taking OR of all the outcomes. The third step can be implemented by a circuit of depth $O(\log |D|)$ by our assumption on PIR. Overall, the depth of the circuit $F[D]$ can be upper bounded by $O(\log \lambda + \log |D|)$ and thus the key generation algorithm runs in time

$$2^{O(\log \lambda + \log |D|)} = \mathrm{poly}(\lambda, |D|)$$

as desired. We finally observe that the decryption algorithm runs in time polynomial in the length of ct and sk, which are bounded by $\mathrm{poly}(\lambda, |D|)$ by the efficiency of the encryption and key generation algorithms. Therefore, the decryption algorithm runs in time $\mathrm{poly}(\lambda, |D|)$ as well.

Correctness. We focus on the case of $|D| = N$, since otherwise it is trivial. By the correctness of IBE, msg is correctly recovered in the first step of the decryption. Furthermore, since $F[D](\mathsf{PIR.query}, k, \mathsf{PIR.answer}_k) = \mathsf{PIR.answer}_k \oplus \mathsf{PIR.answer}_k \oplus 1 = 1$, we observe that lab_k recovered in the 5-th step of the decryption equals to $\mathsf{lab}_{k,\mathsf{PIR.answer}_k}$ by the

correctness of ABE. Finally, by the correctness of GC and PIR, μ recovered in the 6-th step of the decryption equals to $C[N, \mathsf{PIR.st}, \mu_0, \mu_1](\mathsf{PIR.answer}) = \mu_{D[i]}$ with overwhelming probability as desired.

Security. The following theorem addresses the security of LOTFE, whose proof is similar to that of succinct FE by [GTKP+13b]. However, we need somewhat more careful analysis in order to base the security of the scheme on Sel-IND security of the underlying ABE rather than on AD-IND security. The reason why Sel-IND security suffices for our case is that the reduction algorithm can guess the target attribute the adversary chooses only with polynomial security loss. In more detail, we change ABE ciphertexts encrypting $\mathsf{lab}_{k,1-\mathsf{PIR.answer}_k}$ for attribute $(\mathsf{PIR.query}, k, 1-\mathsf{PIR.answer}_k)$ to be that encrypting $\mathsf{lab}_{k,\mathsf{PIR.answer}_k}$ in the security proof. A naive way of guessing the attribute $(\mathsf{PIR.query}, k, 1-\mathsf{PIR.answer}_k)$ ends up with exponential security loss, since there are exponentially many possible PIR.query. However, the reduction algorithm only has to guess (N, i) that is encoded inside PIR.query, because the randomness used for computing the query is not controlled by the adversary, but by the reduction algorithm. Since there are only polynomial number of possible $(N, i, k, 1-\mathsf{PIR.answer}_k)$, the guessing can be done only with polynomial security loss.

Theorem 5.1. *If* IBE *is* AD-IND *secure,* ABE *is* Sel-IND *secure,* GC *is selectively secure, and* PIR *is private, then the above FE is* 1-NA-SIM *secure.*

Proof. To prove the security, we construct the simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$. Since we are going to prove 1-NA-SIM security, we set $\mathsf{SimKG} = \bot$. Let A be an adversary and GC.Sim be the simulator for GC. We define SimEnc as follows.

$\mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}_{\mathsf{LOTFE}}, 1^{|P|})$: On input $\mathsf{mpk} = (\mathsf{ABE.mpk}, \mathsf{IBE.mpk})$, $\mathcal{V}_{\mathsf{LOTFE}} = \{R_{\mathsf{LOTFE}}((N, i, \mu_0, \mu_1), D), D, \mathsf{sk} = (D, \mathsf{ABE.sk}, \mathsf{IBE.sk})\}$ do the following. There are two cases:

- If $|D| \neq N$, we have $R_{\mathsf{LOTFE}}((N, i, \mu_0, \mu_1), D) = (N, 1^{|\mu_0|})$. In this case, it works as follows.
  1. Compute the length $|\mathsf{msg}|$ of msg that is computed when the honest encryption is run on input $(N, 1, 1^{|\mu_0|}, 1^{|\mu_0|})$.
  2. Run $\mathsf{IBE.Enc}(\mathsf{IBE.mpk}, N, 0^{|\mathsf{msg}|}) \to \mathsf{IBE.ct}$.
  3. Output $\mathsf{ct} := (N, 1^{|\mu_0|}, \mathsf{IBE.ct})$.
- If $|D| = N$, we have $R_{\mathsf{LOTFE}}((N, i, \mu_0, \mu_1), D) = (N, \mu_{D[i]})$. In this case, it works as follows.
  1. Pick $\mathsf{lab}_k \leftarrow \{0,1\}^\lambda$ for $k \in [\ell_{\mathsf{PIR}}]$
  2. Run
  $$\widetilde{C} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{|C|}, \mu_{D[i]}, \{\mathsf{lab}_k\}_k).$$
  where $|C|$ is the size of the circuit $C[N, \mathsf{PIR.st}, \mu_0, \mu_1]$ defined in Figure 7 and $\mu_{D[i]}$ is from the input to the simulator.
  3. Run $\mathsf{PIR.Query}(1^\lambda, 1, N) \to (\mathsf{PIR.query}, \mathsf{PIR.st})$.
  4. Run
  $$\mathsf{ABE.ct}_{k,b} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, k, b), \mathsf{lab}_k)$$
  for $k \in [\ell_{\mathsf{PIR}}]$, $b \in \{0,1\}$.
  5. Set $\mathsf{msg} := (\widetilde{C}, \mathsf{PIR.query}, \{\mathsf{ABE.ct}_{k,b}\}_{k,b})$.
  6. Run $\mathsf{IBE.ct} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, N, \mathsf{msg})$.
  7. Output $\mathsf{ct} := (N, 1^{|\mu_0|}, \mathsf{IBE.ct})$.

To prove that the view of the adversary in the real game is indistinguishable from the simulated game, we introduce the following sequence of hybrid games.

Game_0**:** This is the same as $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{LOTFE},\mathsf{A}}(1^\lambda)$ for 1-NA-SIM security.

Game_1**:** This is the same as Game_0 except that the challenger generates IBE.ct as $\mathsf{IBE.Enc}(\mathsf{IBE.mpk}, N, 0^{|\mathsf{msg}|}) \to \mathsf{IBE.ct}$ in the case of $|D| \neq N$.

$\mathsf{Game}_{2,j}$**:** The game is defined for $j \in [\ell_{\mathsf{PIR}} + 1]$. This game is the same as Game_1 except that we change the way $\{\mathsf{ABE.ct}_{k,b}\}_{k,b}$ are computed in the case of $|D| = N$. In particular, we compute $\mathsf{ABE.ct}_{k,b}$ for $k \in [\ell_{\mathsf{PIR}}]$ and $b \in \{0,1\}$ as

$$\mathsf{ABE.ct}_{k,b} \leftarrow \begin{cases} \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, k, b), \mathsf{lab}_{k,\mathsf{PIR.answer}_k}) & \text{if } k < j \\ \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, k, b), \mathsf{lab}_{k,b}) & \text{if } k \geq j \end{cases}, \tag{5.1}$$

where

$$\mathsf{PIR.answer} := \mathsf{PIR.Answer}(\mathsf{PIR.query}, D, 1^{|D|}).$$

Game_3**:** This is the same as $\mathsf{Game}_{2,\ell_{\mathsf{PIR}}+1}$ except that the challenger generates $\widetilde{C}$ as

$$\widetilde{C} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{|C|}, \mu_{D[i]}, \{\mathsf{lab}_{k,\mathsf{PIR.answer}_k}\}_k).$$

Game_4**:** This is the same as Game_3 except that PIR.query as

$$(\mathsf{PIR.query}, \mathsf{PIR.st}) \leftarrow \mathsf{PIR.Query}(1, N).$$

Game_5**:** This is the same as Game_4 except that the game stops choosing $\{\mathsf{lab}_{k,b}\}_{k,b}$. Instead, it chooses lab_k for $k \in [\ell_{\mathsf{PIR}}]$ and uses the label lab_k in place of $\mathsf{lab}_{k,\mathsf{PIR.answer}_k}$. Note that the game is well-defined because $\{\mathsf{lab}_{k,1-\mathsf{PIR.answer}_k}\}_k$ are not at all used in Game_4.

Let $\mathcal{E}_{\mathrm{xx}}$ be the event that A outputs 1 in $\mathsf{Game}_{\mathrm{xx}}$. Since we can see that $\mathsf{Game}_{2,1} \equiv \mathsf{Game}_1$, $\mathsf{Game}_4 \equiv \mathsf{Game}_5 \equiv \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{LOTFE,A}}(1^\lambda)$ by inspection, it suffices to prove Lemmas 5.1 to 5.4 in the following to complete the proof of Theorem 5.1.

Lemma 5.1. *If* IBE *is* AD-IND *secure, we have* $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda)$.

Proof. In this proof, we introduce a simplifying assumption that $|D| \neq N$ always holds for the secret key query D and the encryption query (N, i, μ_0, μ_1) made by A. This is without loss of generality because otherwise the two games are completely the same and there is no way to distinguish between them. Assume $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]|$ is non-negligible. We then describe an adversary B that breaks AD-IND security of IBE as follows. At the beginning of the game, B is given IBE.mpk from its challenger. Then, B samples (ABE.mpk, ABE.msk) by itself and gives $\mathsf{mpk} = (\mathsf{ABE.mpk}, \mathsf{IBE.mpk})$ to A as the master public key.

Answering the secret key query. When A makes a secret key query for $D \in \{0,1\}^{|D|}$, B makes a key query for identities $|D|$ to the challenger. Then, the challenger returns the corresponding secret key $\mathsf{IBE.sk}_{|D|}$ to $\mathcal{B}$. B then runs $\mathsf{ABE.KeyGen}(\mathsf{LOTFE.msk}, F[D]) \to \mathsf{ABE.sk}$ and returns $\mathsf{sk} := (D, \mathsf{ABE.sk}, \mathsf{IBE.sk})$ to $\mathcal{A}$.

Answering the encryption query. When A submits an encryption query (N, i, μ_0, μ_1), B does the following:

1. It computes msg from (N, i, μ_0, μ_1) as in the honest encryption algorithm.
2. It submits $(m_0, m_1) := (\mathsf{msg}, 0^{|\mathsf{msg}|})$ to the challenger. Then, the challenger runs

   $$\mathsf{IBE.ct} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, |N|, m_{\mathsf{coin}})$$

   and returns IBE.ct to B. Here, coin is the coin chosen by the challenger.
3. It then returns the ciphertext $\mathsf{ct} := (N, 1^{|\mu_0|}, \mathsf{IBE.ct})$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates Game_0 for A if $\mathsf{coin} = 0$ and Game_1 if $\mathsf{coin} = 1$. Furthermore, B does not make a secret key query for an identity that is also submitted to the challenger because of our assumption that $|D| \neq N$ always holds. Therefore, B breaks the security of IBE if A distinguishes the games.

Lemma 5.2. *If* ABE *is* Sel-IND *secure, we have* $|\Pr[\mathcal{E}_{2,j}] - \Pr[\mathcal{E}_{2,j+1}]| \leq \mathrm{negl}(\lambda)$ *for* $j \in [\ell_{\mathsf{PIR}}]$.

Proof. In this proof, we introduce a simplifying assumption that $|D| = N$ always holds for the secret key query D and the encryption query (N, i, μ_0, μ_1) made by A. This is without loss of generality because otherwise the two games are completely the same and there is no way to distinguish between them.

To prove the lemma, we consider slightly modified version of the game defined as follows. Let $t_{\max} := t_{\max}(\lambda)$ be the maximum running time of A.

$\mathsf{Game}_{2',j}$: This game is the same as $\mathsf{Game}_{2,j}$ except that B chooses random integers $N', i' \leftarrow [t_{\max}]$ and random bit $b' \leftarrow \{0,1\}$ at the beginning of the game. Furthermore, we replace the output of A with 0 at the end of the game if the following equations do not hold:

$$N' = N, \quad i' = i, \quad b' \neq 1 - \mathsf{PIR.answer}_j, \tag{5.2}$$

where PIR.answer is computed as specified in $\mathsf{Game}_{2,j}$.

We have

$$\begin{aligned}\Pr\left[\mathsf{E}_{2',j}\right] &= \Pr\left[\mathsf{E}_{2,j} \wedge (N' = N \wedge i' = i \wedge b' = 1 - \mathsf{PIR.answer}_j)\right] \\ &= \Pr\left[\mathsf{E}_{2,j}\right] \cdot \Pr\left[(N' = N \wedge i' = i \wedge b' = 1 - \mathsf{PIR.answer}_j)\right] \\ &= \Pr\left[\mathsf{E}_{2,j}\right] / 2t_{\max}^2,\end{aligned}$$

where the second equality follows from the fact that N', i', and b' are chosen independently from the view of A and the last equality follows from $i \leq N = |D| \leq t_{\max}$ and $|D| \leq t_{\max}$, which follows from the fact that A's running time should be large enough to be able to output D. We therefore have

$$|\Pr[\mathsf{E}_{2',j+1}] - \Pr[\mathsf{E}_{2',j}]| = |\Pr[\mathsf{E}_{2,j+1}] - \Pr[\mathsf{E}_{2,j}]|/2t_{\max}^2.$$

From the above discussion, it suffices to show that $|\Pr[\mathsf{E}_{2',j+1}] - \Pr[\mathsf{E}_{2',j}]|$ is negligible, since $t_{\max}$ is polynomially bounded. For the sake of contradiction, we assume that the quantity is non-negligible and then show that there exists an adversary B that breaks Sel-IND security of ABE. B proceeds as follows.

Setting up. At the beginning of the game, B proceeds as follows.

1. It chooses $N', i' \leftarrow [t_{\max}]$ and $b' \leftarrow \{0,1\}$.
2. It computes $(\mathsf{PIR.query}, \mathsf{PIR.st}) \leftarrow \mathsf{PIR.Query}(1^\lambda, i', N')$.
3. It submits its target $(\mathsf{PIR.query}, j, b')$ and the parameter $\mathsf{prm} = 1^{\ell_{\mathsf{PIR}}+2\lambda}$ to its challenger. Then, the challenger runs $\mathsf{ABE.Setup}(1^\lambda, \mathsf{prm})$ and returns ABE.mpk to B.
4. It runs $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$.
5. It gives $\mathsf{mpk} := (\mathsf{ABE.mpk}, \mathsf{IBE.mpk})$ to A as the master public key.

Answering the secret key query. When A makes a secret key query for $D \in \{0,1\}^{|D|}$, B proceeds as follows.

1. It computes $\mathsf{PIR.answer} := \mathsf{PIR.Answer}(\mathsf{PIR.query}, D, 1^{|D|})$.
2. If $|D| \neq N'$ or $b' \neq 1 - \mathsf{PIR.answer}_j$, it aborts and outputs 0.
3. It submits $F[D]$ to its challenger. Then, the challenger runs $\mathsf{ABE.sk} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{ABE.msk}, F[D])$ and returns ABE.sk to B.
4. It then runs $\mathsf{IBE.sk} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, |D|)$.
5. It gives $\mathsf{sk} := (\mathsf{ABE.sk}, \mathsf{IBE.sk})$ to A as the secret key.

Answering the encryption query. When A submits an encryption query (N, i, μ_0, μ_1), B does the following:

1. If $i \neq i'$, it aborts and outputs 0.
2. Pick $\mathsf{lab}_{k,b} \leftarrow \{0,1\}^\lambda$ for $k \in [\ell_{\mathsf{PIR}}]$, $b \in \{0,1\}$.
3. Compute $\mathsf{ABE.ct}_{k,b}$ for $(k,b) \neq (j,b')$ as Equation (5.1).
4. It then submits the messages $(\mathsf{lab}_{j,b'}, \mathsf{lab}_{j,1-b'})$ to its challenger. Then, the challenger computes

$$\mathsf{ABE.ct} \leftarrow \begin{cases} \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, b'), \mathsf{lab}_{j,b'}) & \text{if coin} = 0 \\ \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, b'), \mathsf{lab}_{j,1-b'}) & \text{if coin} = 1 \end{cases}$$

   and returns ABE.ct to B, where coin is challenger's random coin. B then sets $\mathsf{ABE.ct}_{j,b'} := \mathsf{ABE.ct}$.
5. It then computes ct as in the honest encryption algorithm from $(\mathsf{PIR.query}, \mathsf{PIR.st})$, $\{\mathsf{ABE.ct}_{k,b}\}_{k,b}$, and (μ_0, μ_1) and returns ct to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

We claim that the output distribution of B corresponds to that of A in $\mathsf{Game}_{2,j}$ if $\mathsf{coin} = 0$ and $\mathsf{Game}_{2,j+1}$ if $\mathsf{coin} = 1$. There are two cases to consider. We assume that Equation (5.2) holds because otherwise the claim is trivial. Assuming this, we can see that $\mathsf{ABE.ct}_{j,0}$ and $\mathsf{ABE.ct}_{j,1}$ encrypt $\mathsf{lab}_{j,0}$ and $\mathsf{lab}_{j,1}$ if $\mathsf{coin} = 0$ and both encrypt the same label $\mathsf{lab}_{j,\mathsf{PIR.answer}_k}$ if $\mathsf{coin} = 1$ by recalling that $b' = 1 - \mathsf{PIR.answer}_j$ in this case. Therefore, B breaks the Sel-IND security of ABE if A distinguishes the games.

Lemma 5.3. *If* GC *is selectively secure, we have* $|\Pr[\mathcal{E}_{2,\ell_{\mathsf{PIR}}+1}] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$.

Proof. Assume $|\Pr[\mathcal{E}_{2,\ell_{\mathsf{PIR}}+1}] - \Pr[\mathcal{E}_3]|$ is non-negligle. We then describe an adversary B that breaks the selective security of GC as follows. At the beginning of the game, B runs Setup by itself and gives the master public key mpk to A.

Answering the secret key query. B answers A's secret key query using msk.

Answering the encryption query. When A submits an encryption query (N, i, μ_0, μ_1), B does the following:

1. If $|D| \neq N$, B computes ct as specified in Game_1 and returns ct to A.
2. It computes $\mathsf{PIR.Query}(1^\lambda, i, N) \to (\mathsf{PIR.query}, \mathsf{PIR.st})$.
3. It computes $\mathsf{PIR.answer} := \mathsf{PIR.Answer}(\mathsf{PIR.query}, D, 1^N)$.
4. It samples $\mathsf{lab}_{k,\mathsf{PIR.answer}_k} \leftarrow \{0,1\}^\lambda$ for $k \in [\ell_{\mathsf{PIR}}]$.
5. It computes $\mathsf{ABE.ct}_{k,b} \leftarrow \mathsf{ABE.Enc}(\mathsf{ABE.mpk}, (\mathsf{PIR.query}, k, b), \mathsf{lab}_{k,\mathsf{PIR.answer}_k})$ for $k \in [\ell_{\mathsf{PIR}}]$, $b \in \{0,1\}$.
6. It then submits $C := C[N, \mathsf{PIR.st}, \mu_0, \mu_1]$ and labels $\{\mathsf{lab}_{k,\mathsf{PIR.answer}_k}\}_{k\in[\ell_{\mathsf{PIR}}]}$ to its challenger. Then the challenger chooses $\{\mathsf{lab}_{k,1-\mathsf{PIR.answer}_k}\}_{k\in[\ell_{\mathsf{PIR}}]}$, runs
$$\widetilde{C} \leftarrow \begin{cases} \mathsf{GC.Garble}\left(1^\lambda, C, \{\mathsf{lab}_{k,b}\}_{k,b}\right) & \text{in the real world} \\ \mathsf{GC.Sim}(1^\lambda, 1^{|C|}, C(\mathsf{PIR.answer}), \{\mathsf{lab}_{k,\mathsf{PIR.answer}_k}\}_k) & \text{in the ideal world} \end{cases},$$
and returns $\widetilde{C}$ to A.
7. It runs $\mathsf{IBE.ct} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, N, \mathsf{msg})$, where $\mathsf{msg} := (\widetilde{C}, \mathsf{PIR.query}, \{\mathsf{ABE.ct}_{k,b}\}_{k,b})$.
8. It returns $(N, 1^{|\mu_0|}, \mathsf{IBE.ct})$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathsf{Game}_{2,\ell_{\mathsf{PIR}}+1}$ for A if B is in the real world. It can also be seen that B simulates Game_3 for A if B is in the ideal world by observing
$$C(\mathsf{PIR.answer}) = \mu_{\mathsf{PIR.Reconstruct}(\mathsf{PIR.st},\mathsf{PIR.answer},N)} = \mu_{D[i]}$$
holds with overwhelming probability by the correctness of PIR. Note that we can invoke the correctness of PIR because randomness of PIR.Query is chosen *after* D and i are declared by the adversary, both of which may be maliciously chosen. Therefore, B breaks the security of GC if A distinguishes the games.

Lemma 5.4. *If* PIR *satisfies privacy, we have* $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda)$.

Proof. Assume $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]|$ is non-negligle. We then describe an adversary B that breaks the privacy of PIR as follows. At the beginning of the game, B runs Setup by itself and gives the master public key mpk to A.

Answering the secret key query. B answers A's secret key query using msk.

Answering the encryption query. When A submits an encryption query (N, i, μ_0, μ_1), B does the following:

1. If $|D| \neq N$, B computes ct as specified in Game_1 and returns ct to A.
2. It submits $((i,1), N)$ and the challenger runs
$$(\mathsf{PIR.query}, \mathsf{PIR.st}) \leftarrow \begin{cases} \mathsf{PIR.Query}(1^\lambda, i, N) & \text{if } \mathsf{coin} = 0 \\ \mathsf{PIR.Query}(1^\lambda, 1, N) & \text{if } \mathsf{coin} = 1 \end{cases},$$
where coin is the coin chosen by the challenger. Then B is given PIR.query from the challenger.

3. It then computes ct as specified in Game_3 and returns ct to A. Note that B can do this without $\mathsf{PIR.st}$ due to the change introduced in Game_3.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates Game_3 for A if $\mathsf{coin} = 0$ and Game_4 otherwise. Therefore, B breaks the privacy of PIR if A distinguishes the games.

5.2 CPFE for read only RAM

Here, we define CPFE for read only RAM by specifying the relation $R_{\mathsf{CPRAMFE}} : \mathcal{X}_{\mathsf{CPRAMFE}} \times \mathcal{Y}_{\mathsf{CPRAMFE}} \to \{0,1\}^*$.

CPFE for read only RAM computation. To define CPFE for read only RAM, we set $\mathcal{Y}_{\mathsf{CPRAMFE}} = \{0,1\}^*$ and $\mathcal{X}_{\mathsf{CPRAMFE}}$ to be a set of read only RAM programs of the form $P = \{P^\tau\}_{\tau\in[t]}$. We define $R_{\mathsf{CPRAMFE}}(P, D) \in \{0,1\}^*$ to be the output obtained by executing P with the RAM access to the data D. In our case, we consider RAM programs with some specific structure. To define this, we introduce the parameter $\mathsf{prm} = 1^{\ell_{\mathsf{st}}}$ and the set of RAM programs $\mathcal{P}_{T,\mathsf{size}}$. We then constrain the domain as $\mathcal{X}_{\mathsf{prm}} = \mathcal{P}_{T,\mathsf{size}}$ and $\mathcal{Y}_{\mathsf{prm}} = \{0,1\}^*$. A RAM program in $\mathcal{P}_{T,\mathsf{size}}$ is of the form $P = \{P^\tau\}_\tau$ where the step circuit P^τ is of the form $P^\tau : \{0,1\}^{\ell_{\mathsf{st}}} \to \{0,1\}^{\ell_{\mathsf{st}}-1} \times \{0,1\}^{\log^2\lambda}$. For $D \in \{0,1\}^N$ and $P = \{P^\tau\}_{\tau\in[t]}$, we define $(\mathsf{st}^\tau, L^\tau) \in \{0,1\}^{\ell_{\mathsf{st}}-1} \times \{0,1\}^{\log^2\lambda}$ for $\tau \in [2,t]$ by induction as

$$(\mathsf{st}^\tau, L^\tau) := P^{\tau-1}(\mathsf{st}^{\tau-1}, D[L^{\tau-1}]) \quad \text{where} \quad (\mathsf{st}^1, D[L^1]) := (0^{\ell_{\mathsf{st}}-1}, 0). \tag{5.3}$$

Here, st^τ and $L^\tau \in \{0,1\}^{\log^2\lambda}$ output by $P^{\tau-1}$ represent the state information and the position L^τ to be read from the data D respectively. The state st^τ and the read bit $D[L^\tau]$, which is the L^τ-th bit of D, is then input to the next step circuit P^τ. In the above computation, the initial state st^1 and the initial read bit $D[L^1]$ are defined to be zero strings. Note that the position to be read is assumed to be represented by a binary string of $\{0,1\}^{\log^2\lambda}$, which is interpreted as an integer in $[0, 2^{\log^2\lambda}]$. Since $2^{\log^2\lambda} = \lambda^{\log\lambda}$ is super-polynomial, this is sufficient for pointing a position in any unbounded size data. The output obtained by running P with the RAM access to the data D is denoted by P^D and is defined to be $P^D := \mathsf{st}^{t+1}$.

Remark 5.3 (Succinctness). Similarly to the case of LOTFE, the running time of the encryption algorithm is independent from the size of the database D supported by the scheme. This property can be seen as an analogue of the efficiency requirements for the succinctness of FE [GTKP+13b] or laconic OT [CDG+17].

Remark 5.4 (Efficiency). We note that we do not require the decryption time to be independent of the size of the database D, while the encryption time is required to be so. This is in contrast to ABE/FE for RAM efficiency in the literature [GHRW14b, AFS19], where the decryption time is also required to be sublinear in the size of the database. However, our weaker definition suffices for our purpose of constructing FE for TM.

Ingredients. We now describe the underlying building blocks used for our construction of CPFE for read only RAM.

1. FE with laconic OT functionality $\mathsf{LOTFE} = (\mathsf{LOTFE.Setup}, \mathsf{LOTFE.KeyGen}, \mathsf{LOTFE.Enc}, \mathsf{LOTFE.Dec})$ with 1-$\mathsf{NA\text{-}SIM}$ security. This can be instantiated by the scheme in Section 5.1. For simplicity, we assume that the encryption algorithm of LOTFE only requires randomness of λ bits. This can be achieved by using the randomness as a PRF key to derive longer pseudorandom string for example.
2. IBE scheme $\mathsf{IBE} = (\mathsf{IBE.Setup}, \mathsf{IBE.KeyGen}, \mathsf{IBE.Enc}, \mathsf{IBE.Dec})$ with $\mathsf{AD\text{-}IND}$ security. Since the construction of LOTFE in Section 5.1 already uses IBE, this does not add new assumption.
3. Selectively secure garbled circuit $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Sim})$. We can instantiate it from any one-way function [Yao86].

Construction. Here, we describe our scheme $\mathsf{CPRAMFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The construction is inspired by the garbled RAM construction by [GHRW14a], which in turn is based on [LO14], where a sequence of garbled circuits read the memory stored outside of the circuits via RAM access. Whereas they use the combination of IBE and ORAM to enable the oblivious access to the memory, we use LOTFE for this purpose instead. This change is crucial for us because ORAM is a secret key primitive and is not compatible with our setting of public key FE.

Circuit $\mathsf{SC}[\tau, P^\tau, \mathsf{LOTFE.mpk}, \mathsf{R}^\tau, \{\mathsf{lab}_{k,b}^{\tau+1}\}_{k,b}]$

Hardwired constants: The step number τ, the step circuit P^τ, the master public key $\mathsf{LOTFE.mpk}$, randomness R^τ, and the set of labels $\{\mathsf{lab}_{k,b}^{\tau+1}\}_{k\in[n],b\in\{0,1\}}$.
Input: String $X \in \{0,1\}^n$.

1. Parse the input as $X \to (N, \mathsf{st}, \mathsf{rData})$, where $N \in \{0,1\}^{\log^2\lambda}$, $\mathsf{st} \in \{0,1\}^{\ell_{\mathsf{st}}-1}$, and $\mathsf{rData} \in \{0,1\}$.
2. Run $P^\tau(\mathsf{st}, \mathsf{rData}) = (\mathsf{st}', L)$.
3. Run $\mathsf{LOTFE.Enc}\left(\mathsf{LOTFE.mpk}, (N, L, \mathsf{lab}_{n,0}^{\tau+1}, \mathsf{lab}_{n,1}^{\tau+1}); \mathsf{R}^\tau\right) \to \mathsf{LOTFE.ct}$.
4. Set $Y := (N, \mathsf{st}')$.
5. Output $\begin{cases} \left(\left\{\mathsf{lab}_{k,Y_k}^{\tau+1}\right\}_{k\in[n-1]}, \mathsf{LOTFE.ct}\right) & \text{if } \tau \neq t \\ \mathsf{st}' & \text{if } \tau = t \end{cases}$

Fig. 9 Circuit $\mathsf{SC}^\tau = \mathsf{SC}[\tau, P^\tau, \mathsf{LOTFE.mpk}, \mathsf{R}^\tau, \{\mathsf{lab}_{k,b}^{\tau+1}\}_{k,b}]$

$\mathsf{Setup}(1^\lambda, \mathsf{prm})$: On input the security parameter λ, the parameter $\mathsf{prm} = 1^{\ell_{\mathsf{st}}}$, do the following:

1. Run $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$.
2. Run $(\mathsf{LOTFE.mpk}, \mathsf{LOTFE.msk}) \leftarrow \mathsf{LOTFE.Setup}(1^\lambda)$.
3. Output $\mathsf{mpk} := (\mathsf{LOTFE.mpk}, \mathsf{IBE.mpk})$ and $\mathsf{msk} := (\mathsf{LOTFE.msk}, \mathsf{IBE.msk})$.

$\mathsf{Enc}(\mathsf{mpk}, P)$: On input the master public key $\mathsf{mpk} = (\mathsf{LOTFE.mpk}, \mathsf{IBE.mpk})$, a read only RAM program $P = \{P_\tau\}_{\tau\in[t]} \in \mathcal{P}_{\ell_{\mathsf{st}}}$, do the following:

1. Set $n := \log^2\lambda + \ell_{\mathsf{st}}$.
2. Pick $\mathsf{lab}_{k,b}^\tau \leftarrow \{0,1\}^\lambda$ for $\tau \in [t]$, $k \in [n]$, and $b \in \{0,1\}$.
3. Pick $\mathsf{R}^\tau \leftarrow \{0,1\}^\lambda$ for $\tau \in [t]$.
4. Construct circuit $\mathsf{SC}^\tau := \mathsf{SC}[\tau, P^\tau, \mathsf{LOTFE.mpk}, \mathsf{R}^\tau, \{\mathsf{lab}_{k,b}^{\tau+1}\}_{k,b}]$ for $\tau \in [t]$ as Figure 9, where we define $\mathsf{lab}_{k,b}^{t+1} = \bot$ for $k \in [n]$, $b \in \{0,1\}$.
5. For all $\tau \in [t]$, run
$$\widetilde{\mathsf{SC}}^\tau \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathsf{SC}^\tau, \{\mathsf{lab}_{k,b}^\tau\}_{k,b}\right).$$
6. For all $k \in [n]$, $b \in \{0,1\}$, run
$$\mathsf{IBE.ct}_{k,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, (k,b), \mathsf{lab}_{k,b}^1).$$
7. Output $\mathsf{ct} := \left(\{\widetilde{\mathsf{SC}}^\tau\}_{\tau\in[t]}, \{\mathsf{IBE.ct}_{k,b}\}_{k\in[n],b\in\{0,1\}}\right)$.

$\mathsf{KeyGen}(\mathsf{msk}, D)$: On input the master secret key $\mathsf{msk} = \mathsf{LOTFE.msk}$, an input $D \in \{0,1\}^N$, where $N \leq 2^{\log^2\lambda}$, do the following:

1. Run
$$\mathsf{LOTFE.sk} \leftarrow \mathsf{LOTFE.KeyGen}(\mathsf{LOTFE.msk}, D).$$
2. Set $X = N \| 0^{n-\log^2\lambda}$, where N is represented as a string in $\{0,1\}^{\log^2\lambda}$.
3. For all $k \in [n]$, run
$$\mathsf{IBE.sk}_{k,X_k} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, (k, X_k)).$$
4. Output $\mathsf{sk} := \left(D, \mathsf{LOTFE.sk}, \{\mathsf{IBE.sk}_{k,X_k}\}_{k\in[n]}\right)$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$: On input a ciphertext $\mathsf{ct} = (\{\widetilde{\mathsf{SC}}^\tau\}_\tau, \{\mathsf{IBE.ct}_{k,b}\}_{k,b})$ and a secret key $\mathsf{sk} = (D, \mathsf{LOTFE.sk}, \{\mathsf{IBE.sk}_{k,X_k}\}_k)$, do the following:

1. Run Set $X := N \| 0^{n-\log^2\lambda}$.
2. Run $\overline{\mathsf{lab}_k^1} := \mathsf{IBE.Dec}(\mathsf{IBE.sk}_{k,X_k}, \mathsf{IBE.ct}_{k,X_k})$ for $k \in [n]$.
3. Set $\overline{\mathsf{label}} := \{\mathsf{lab}_k^1\}_{k\in[n]}$.
4. For $\tau = 1, \ldots, t$

(a) Compute gout := $\mathsf{GC.Eval}(\widetilde{\mathsf{SC}}^{\tau}, \overline{\mathsf{label}})$.
(b) If $\tau = t$, set $y :=$ gout and break out of the loop.
(c) Parse gout $\to \left(\{\mathsf{lab}_k\}_{k\in[n-1]}, \mathsf{LOTFE.ct}\right)$.
(d) Compute $(N, \mathsf{lab}_n) := \mathsf{LOTFE.Dec}(\mathsf{LOTFE.sk}, \mathsf{LOTFE.ct})$.
(e) Set $\overline{\mathsf{label}} := \{\mathsf{lab}_k\}_{k\in[n]}$.

5. Output y.

Correctness. The following theorem addresses the correctness of the scheme.

Theorem 5.2. *If* LOTFE*,* GC*, and* IBE *are correct, then* CPRAMFE *above is correct.*

Proof. The proof is by induction. We define $(\mathsf{st}^{\tau}, L^{\tau})$ for $\tau \in [2, t]$ as Equation (5.3) and X^{τ} for $\tau \in [2, t]$ as

$$X^{\tau} = (N, \mathsf{st}^{\tau}, D[L^{\tau}]). \tag{5.4}$$

We also define $X^1 = N \| 0^{n-\log^2 \lambda}$. We then claim that $\overline{\mathsf{label}}$ that the decryption algorithm has at the beginning of the τ-th loop in the 4-th step of the algorithm corresponds to $\mathsf{lab}^{\tau}_{k,X^{\tau}_k}$. By the correctness of IBE, it is easy to see that the statement holds true in the case of $\tau = 1$. Assuming that the statement holds for $\tau = i$ for $i \in [t-1]$, we can see that gout recovered in the i-th loop equals to $\left(\left\{\mathsf{lab}^{i+1}_{k,Y_k}\right\}_{k\in[n-1]}, \mathsf{LOTFE.ct}\right)$ with $Y = (N, \mathsf{st}^{i+1})$ and $\mathsf{LOTFE.ct} = \mathsf{LOTFE.Enc}\left(\mathsf{LOTFE.mpk}, (N, L^{i+1}, \mathsf{lab}^{i+1}_{n,0}, \mathsf{lab}^{i+1}_{n,1}); \mathsf{R}^i\right)$ by the correctness of GC. By the correctness of LOTFE, lab_n recovered in the i-th loop equals to

$$\mathsf{lab}_n = \mathsf{lab}^{i+1}_{n,D[L^{i+1}]}.$$

Therefore, we recover labels for $(Y, D[L^{i+1}]) = (N, \mathsf{st}^{i+1}, D[L^{i+1}]) = X^{i+1}$ at the end of the i-th loop as desired. Having established that $\{\mathsf{lab}^t_{k,X^t_k}\}_k$ is input to $\widetilde{\mathsf{SC}}^t$, it is easy to see that $\mathsf{st}^{t+1} = P^D$ is output by the algorithm as the final decryption result by the correctness of GC.

Efficiency. By the efficiency of LOTFE and IBE, it is easy to see that Setup and KeyGen run in time $\mathrm{poly}(\lambda)$ and $\mathrm{poly}(\lambda, |D|)$, respectively. We can also see that $|\widetilde{\mathsf{SC}}^{\tau}| = \mathrm{poly}(\lambda, |P^{\tau}|)$ and thus the running time of Enc can be bounded by $\mathrm{poly}(\lambda, |P|)$. Finally, Dec runs in polynomial time in its input length by the efficiency of the underlying primitives and thus run in time $\mathrm{poly}(\lambda, |P|, |D|)$.

Security. The following theorem addresses the security of CPRAMFE.

Theorem 5.3. *If* IBE *is* AD-IND *secure,* GC *is selectively secure, and* LOTFE *is* 1-NA-SIM *secure, then* CPRAMFE *is* 1-NA-SIM *secure.*

Proof. To prove the security, we construct the simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$. Since we are going to prove NA-SIM security, we set $\mathsf{SimKG} = \bot$. Let A be an adversary and GC.Sim be the simulator for GC. We define SimEnc as follows.

$\mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}_{\mathsf{CPRAMFE}}, 1^{|P|})$: On input $\mathsf{mpk} = (\mathsf{LOTFE.mpk}, \mathsf{IBE.mpk})$, $\mathcal{V}_{\mathsf{CPRAMFE}} = \{y = P^D, D, \mathsf{sk} = (D, \mathsf{LOTFE.sk}, \{\mathsf{IBE.sk}_k\}_{k\in[n]})\}$ do the following:

1. Pick $\mathsf{lab}^{\tau}_k \leftarrow \{0,1\}^{\lambda}$ for $\tau \in [t]$, $k \in [n]$.
2. Run $\mathsf{IBE.ct}_{k,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, (k, b), \mathsf{lab}^1_k)$ for $k \in [n]$ and $b \in \{0,1\}$.
3. Run
$$\mathsf{LOTFE.ct}^{\tau} \leftarrow \mathsf{LOTFE.Enc}\left(\mathsf{LOTFE.mpk}, (|D|, 1, \mathsf{lab}^{\tau+1}_n, \mathsf{lab}^{\tau+1}_n)\right)$$
for $\tau \in [t-1]$.
4. For $\tau \in [t]$, set gout^{τ} as
$$\mathsf{gout}^{\tau} = \begin{cases} \left(\{\mathsf{lab}^{\tau+1}_k\}_{k\in[n-1]}, \mathsf{LOTFE.ct}^{\tau}\right) & \text{if } \tau \neq t \\ y & \text{if } \tau = t \end{cases}.$$

5. For $\tau \in [t]$, compute $\widetilde{\mathsf{SC}}^{\tau}$ as follows.

$$\widetilde{\mathsf{SC}}^{\tau} \leftarrow \mathsf{GC.Sim}(1^{\lambda}, 1^{|\mathsf{SC}^{\tau}|}, \mathsf{gout}^{\tau}, \{\mathsf{lab}_{k}^{\tau}\}_{k}).$$

6. Output $\mathsf{ct} := \left(\{\widetilde{\mathsf{SC}}^{\tau}\}_{\tau\in[t]}, \{\mathsf{IBE.ct}_{k,b}\}_{k\in[n],b\in\{0,1\}}\right)$.

To prove that the view of the adversary in the real game is indistinguishable from the simulated game, we introduce the following sequence of hybrid games.

Game_0**:** This is the same as $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPRAMFE},\mathsf{A}}(1^{\lambda})$ for 1-NA-SIM security.
Game_1**:** This is the same as Game_0 except that the challenger generates $\mathsf{IBE.ct}_{k,b}$ as

$$\mathsf{IBE.ct}_{k,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, (k,b), \mathsf{lab}^{1}_{k,X^{1}_{k}}) \tag{5.5}$$

for $k \in [n]$ and $b \in \{0,1\}$, where $X^1 := N \| 0^{n-\log^2 \lambda}$ and N is the length of the key query D.

We then define $\mathsf{Game}_{2,i,1}$ for $i \in [t+1]$ and $\mathsf{Game}_{2,i,2}$ and $\mathsf{Game}_{2,i,3}$ for $i \in [t]$ as follows.

$\mathsf{Game}_{2,i,1}$**:** This is the same as Game_1 except that the the way the encryption query is answered. In particular, $\{\widetilde{\mathsf{SC}}^{\tau}\}_{\tau}$ is generated as follows in this game.

1. Compute $(\mathsf{st}^{\tau}, L^{\tau})$ for $\tau \in [2,t]$ as Equation (5.3) and X^{τ} for $\tau \in [2,t]$ as Equation (5.4) from the secret key query D and the encryption query $P = \{P^{\tau}\}_{\tau}$ made by A. Recall that X^1 is already defined in the description of Game_1.
2. Run

$$\mathsf{LOTFE.ct}^{\tau} \leftarrow \mathsf{LOTFE.Enc}\left(\mathsf{LOTFE.mpk}, \left(|D|, 1, \mathsf{lab}^{\tau+1}_{n,X^{\tau+1}_{n}}, \mathsf{lab}^{\tau+1}_{n,X^{\tau+1}_{n}}\right); \mathsf{R}^{\tau}\right) \tag{5.6}$$

for $\tau \in [i-1]$.
3. For $\tau \in [i-1]$, set gout^{τ} as

$$\mathsf{gout}^{\tau} = \begin{cases} \left(\left\{\mathsf{lab}^{\tau+1}_{k,X^{\tau+1}_{k}}\right\}_{k\in[n-1]}, \mathsf{LOTFE.ct}^{\tau}\right) & \text{if } \tau \neq t \\ y & \text{if } \tau = t \end{cases}. \tag{5.7}$$

4. Compute $\widetilde{\mathsf{SC}}^{\tau}$ for $\tau = 1, \ldots, t$ as

$$\widetilde{\mathsf{SC}}^{\tau} \leftarrow \begin{cases} \mathsf{GC.Sim}(1^{\lambda}, 1^{|\mathsf{SC}^{\tau}|}, \mathsf{gout}^{\tau}, \{\mathsf{lab}^{\tau}_{k,X^{\tau}_{k}}\}_{k}). & \text{if } \tau < i \\ \mathsf{GC.Garble}\left(1^{\lambda}, \mathsf{SC}^{\tau}, \{\mathsf{lab}^{\tau}_{k,b}\}_{k,b}\right) & \text{if } \tau \geq i \end{cases}, \tag{5.8}$$

where $\mathsf{SC}^{\tau} := \mathsf{SC}\left[\tau, P^{\tau}, \mathsf{LOTFE.mpk}, \mathsf{R}^{\tau}, \{\mathsf{lab}^{\tau+1}_{k,b}\}_{k,b}\right]$.

$\mathsf{Game}_{2,i,2}$**:** This game is the same as $\mathsf{Game}_{2,i,1}$ except that we compute $\widetilde{\mathsf{SC}}^{i}$ as follows.

1. If $i \neq t$, compute $\mathsf{LOTFE.ct}^{i}$ as

$$\mathsf{LOTFE.ct}^{i} \leftarrow \mathsf{LOTFE.Enc}\left(\mathsf{LOTFE.mpk}, \left(|D|, 1, \mathsf{lab}^{i+1}_{n,0}, \mathsf{lab}^{i+1}_{n,1}\right); \mathsf{R}^{i}\right).$$

2. Set gout^{i} as

$$\mathsf{gout}^{i} = \begin{cases} \left(\left\{\mathsf{lab}^{i+1}_{k,X^{i+1}_{k}}\right\}_{k\in[n-1]}, \mathsf{LOTFE.ct}^{i}\right) & \text{if } i \neq t \\ y & \text{if } i = t \end{cases}.$$

3. Compute $\widetilde{\mathsf{SC}}^{i}$ as

$$\widetilde{\mathsf{SC}}^{i} \leftarrow \mathsf{GC.Sim}\left(1^{\lambda}, 1^{|\mathsf{SC}|}, \mathsf{gout}^{i}, \{\mathsf{lab}^{i}_{k,X^{\tau}_{k}}\}_{k}\right). \tag{5.9}$$

$\mathsf{Game}_{2,i,3}$**:** This game is the same as $\mathsf{Game}_{2,i,2}$ except that we compute $\mathsf{LOTFE.ct}^i$ as

$$(\mathsf{LOTFE.ct}^i, \mathsf{LOTFE.st}) \leftarrow \mathsf{LOTFE.SimEnc}\left(\mathsf{LOTFE.mpk}, \mathcal{V}_{\mathsf{LOTFE}}, 1^\lambda\right),$$

where $\mathcal{V}_{\mathsf{LOTFE}} := \{(N, \mathsf{lab}^{i+1}_{n,X_n^{i+1}}), D, \mathsf{LOTFE.sk}\}$.

Game_3**:** This is the same as $\mathsf{Game}_{2,t+1,1}$ except that the game stops choosing $\{\mathsf{lab}^\tau_{k,b}\}_{\tau,k,b}$. Instead, it chooses lab^τ_k for $\tau \in [t]$ and $k \in [n]$ and uses the label lab^τ_k in place of $\mathsf{lab}^\tau_{k,X^\tau_k}$. Note that the game is well-defined since $\mathsf{lab}^\tau_{k,1-X^\tau_k}$ is not at all in $\mathsf{Game}_{2,t+1,1}$. Since we only change the names of the labels, this is a conceptual change.

Let $\mathcal{E}_{\mathrm{xx}}$ be the event that A outputs 1 in $\mathsf{Game}_{\mathrm{xx}}$. Since we can see that $\mathsf{Game}_1 \equiv \mathsf{Game}_{2,1,1}$ and $\mathsf{Game}_{2,t+1,1} \equiv \mathsf{Game}_3 \equiv \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPRAMFE},\mathsf{A}}(1^\lambda)$ by inspection, it suffices to prove Lemmas 5.5 to 5.8 in the following to complete the proof of Theorem 5.3.

Lemma 5.5. *If* IBE *is* AD-IND *secure, we have* $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda)$.

Proof. Assume $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]|$ is non-negligle. We then describe an adversary B that breaks multi-challenge AD-IND security of IBE as follows. Note that as we remarked in Remark 2.4, the multi-challenge security notion is equivalent to more standard single-challenge security notion. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = 1^{\ell_{\mathsf{st}}}$. Then, B is given IBE.mpk from its challenger. It then runs $\mathsf{LOTFE.Setup}(1^\lambda) \to (\mathsf{LOTFE.mpk}, \mathsf{LOTFE.msk})$ and gives $\mathsf{mpk} = (\mathsf{IBE.mpk}, \mathsf{LOTFE.mpk})$ to A as master public key.

Answering the secret key query. When A makes a secret key query for $D \in \{0,1\}^{|D|}$, B proceeds as follows.

1. It first sets $X := N \| 0^{n - \log^2 \lambda}$ where $N = |D|$.
2. It makes key queries for identities (k, X_k) to the challenger for $k \in [n]$. For each query, the challenger returns $\mathsf{IBE.sk}_{k,X_k}$ to $\mathcal{B}$.
3. It runs $\mathsf{LOTFE.sk} \leftarrow \mathsf{LOTFE.KeyGen}(\mathsf{LOTFE.msk}, D)$.
4. It returns $\mathsf{sk} := (D, \mathsf{LOTFE.sk}, \{\mathsf{IBE.sk}_{k,X_k}\}_k)$ to A.

Answering the encryption query. When A submits an encryption query $P = \{P^\tau\}_{\tau \in [t]}$, B does the following:

1. It samples $\{\mathsf{lab}^\tau_{k,b}\}_{\tau,k,b}$ and computes $\{\widetilde{\mathsf{SC}}^\tau\}_\tau$ as in the honest encryption algorithm.
2. It computes $\{\mathsf{IBE.ct}_{k,b}\}_{k\in[n],b\in\{0,1\}}$ as follows. There are two cases:
   - If $b = 1 - X_k$, it submits the identity (k, b) and two messages $(\mu_0, \mu_1) := (\mathsf{lab}^1_{k,1-X_k}, \mathsf{lab}^1_{1,X_k})$ to the challenger. Then, the challenger runs
   $$\mathsf{IBE.ct}_{k,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, (k,b), \mu_{\mathsf{coin}}).$$
   and returns it to $\mathcal{B}$, where coin is the coin chosen by the challenger.
   - Otherwise, it computes $\mathsf{IBE.ct}_{k,b}$ as in the honest encryption algorithm. Namely, it computes
   $$\mathsf{IBE.ct}_{k,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{IBE.mpk}, (k,b), \mathsf{lab}^1_{k,b}).$$
3. It then returns the ciphertext $\mathsf{ct} := \left(\{\widetilde{\mathsf{SC}}^\tau\}_{\tau\in[t]}, \{\mathsf{IBE.ct}_{k,b}\}_{k\in[n],b\in\{0,1\}}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates Game_0 for A if $\mathsf{coin} = 0$ and Game_1 if $\mathsf{coin} = 1$. Furthermore, it is also easy to see that B does not make a secret key query for an identity that is also submitted to the challenger for an encryption query and vice versa. This means that B breaks the AD-IND security of IBE without violating the query restriction of the game.

Lemma 5.6. *If* GC *is selectively secure, we have* $|\Pr[\mathcal{E}_{2,i,1}] - \Pr[\mathcal{E}_{2,i,2}]| \leq \mathrm{negl}(\lambda)$ *for* $i \in [t]$.

Proof. Assume $|\Pr[\mathcal{E}_{2,i,1}] - \Pr[\mathcal{E}_{2,i,2}]|$ is non-negligle. We then describe an adversary B that breaks selective security of GC as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = 1^{\ell_{\mathsf{st}}}$. B then runs $\mathsf{Setup}(1^\lambda, \mathsf{prm})$ to obtain mpk and msk and then sends mpk to A as the master public key.

Answering the secret key query. B answers secret key queries using msk.

Answering the encryption query. When A submits an encryption query $P = \{P^\tau\}_\tau$, B proceeds as follows. On input the master public key $\mathsf{mpk} = (\mathsf{LOTFE.mpk}, \mathsf{IBE.mpk})$, a read only RAM program $P = \{P_\tau\}_{\tau\in[t]}$, do the following:

1. It samples $\mathsf{lab}^\tau_{k,b} \leftarrow \{0,1\}^\lambda$ for $\tau \in [t]\backslash\{i\}$, $k \in [n]$, and $b \in \{0,1\}$.
2. It samples lab^i_{k,X^i_k} for $k \in [n]$, where X^i is computed from P and D.
3. It computes $\mathsf{IBE.ct}_{k,b}$ for $k \in [n], b \in \{0,1\}$ as Equation (5.5). Note that it suffices to have the labels $\{\mathsf{lab}^1_{k,X^1_k}\}_{k\in[n]}$ for doing this step. In particular, the labels $\{\mathsf{lab}^1_{k,1-X^1_k}\}_{k\in[n]}$, which are not defined in the case of $i = 1$, are not necessary.
4. It samples $\mathsf{R}^\tau \leftarrow \{0,1\}^\lambda$ for $\tau \in [t]$.
5. It computes $\mathsf{LOTFE.ct}^\tau$ for $\tau \in [i-1]$ as Equation (5.6). Note that for the computation of $\mathsf{LOTFE.ct}^i$, only lab^i_{n,X^i_n} is required and the other label $\mathsf{lab}^i_{n,1-X^i_n}$, which is not defined, is not necessary.
6. It sets gout^τ as Equation (5.7) for $\tau \in [i-1]$. Note that for setting gout^{i-1}, only $\{\mathsf{lab}^i_{k,X^i_k}\}_{k\in[n-1]}$ are required and the other labels $\{\mathsf{lab}^i_{n,1-X^i_n}\}_{k\in[n-1]}$, which are not defined, are not necessary.
7. It computes $\widetilde{\mathsf{SC}}^\tau$ for $\tau \neq i$ as Equation (5.8) using $\{\mathsf{gout}^\tau\}_{\tau\in[i-1]}$, $\{\mathsf{R}^\tau\}_{\tau\in[t]}$, $\{\mathsf{lab}^\tau_{k,b}\}_{\tau\neq i,k,b}$.
8. To compute $\widetilde{\mathsf{SC}}^i$, $\mathcal{B}$ submits the circuit $\mathsf{SC}^i{:=}\mathsf{SC}[i, P^i, \mathsf{LOTFE.mpk}, \mathsf{R}^i, \{\mathsf{lab}^{i+1}_{k,b}\}_{k,b}]$ and labels $\{\mathsf{lab}^i_{k,X^i_k}\}_{k\in[n]}$. Then the challenger chooses $\{\mathsf{lab}^i_{k,1-X^i_k}\}_{k\in[n]}$, runs
$$\widetilde{\mathsf{SC}}^i \leftarrow \begin{cases} \mathsf{GC.Garble}\left(1^\lambda, \mathsf{SC}^i, \{\mathsf{lab}^i_{k,b}\}_{k,b}\right) & \text{in the real world} \\ \mathsf{GC.Sim}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{SC}^i(X^i), \{\mathsf{lab}^i_{k,X^i_k}\}_k) & \text{in the ideal world} \end{cases},$$
and returns $\widetilde{\mathsf{SC}}^i$ to A.
9. Finally, $\mathcal{B}$ returns $\mathsf{ct}{:=}(\{\widetilde{\mathsf{SC}}^\tau\}_\tau, \{\mathsf{IBE.ct}_{k,b}\}_{k,b})$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

By observing $\mathsf{SC}^i(X^i) = \mathsf{gout}^i$, we can see that B simulates $\mathsf{Game}_{2,i,1}$ for A if it is in the real world and $\mathsf{Game}_{2,i,2}$ if it is in the ideal world. Thus, B breaks the selective security of GC if A distinguishes between the two games.

Lemma 5.7. *If* LOTFE *is* 1-NA-SIM *secure, we have* $|\Pr[\mathcal{E}_{2,i,2}] - \Pr[\mathcal{E}_{2,i,3}]| \leq \mathrm{negl}(\lambda)$ *for* $i \in [t]$.

Proof. Assume $|\Pr[\mathcal{E}_{2,i,2}] - \Pr[\mathcal{E}_{2,i,3}]|$ is non-negligle. We then describe an adversary B that breaks the 1-NA-SIM security of LOTFE as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = 1^{\ell_{\mathsf{st}}}$. The challenger then sends $\mathsf{LOTFE.mpk}$ to B. B then samples $(\mathsf{IBE.mpk}, \mathsf{IBE.msk})$ by itself and sends $\mathsf{mpk} := (\mathsf{LOTFE.mpk}, \mathsf{IBE.mpk})$ to A.

Answering the secret key query. When A submits a key query for a data $D \in \{0,1\}^N$, B does the following:

1. It computes $\{\mathsf{IBE.sk}_{k,X_k}\}_k$ using $\mathsf{IBE.msk}$.
2. It makes a secret key query for D to its challenger to obtain $\mathsf{LOTFE.sk} \leftarrow \mathsf{LOTFE.KeyGen}(\mathsf{LOTFE.msk}, D)$.
3. Finally, it sends $\mathsf{sk} := (\mathsf{LOTFE.sk}, \{\mathsf{IBE.sk}_{k,X_k}\}_k)$ to A.

When A submits an encryption query $P = \{P^\tau\}_\tau$, B does the following:

1. Pick $\{\mathsf{lab}^\tau_{k,b}\}_{\tau,k,b}$ as in honest encryption.
2. Pick $\mathsf{R}^\tau \leftarrow \{0,1\}^\lambda$ for $\tau \in [t]\backslash\{i\}$.
3. It computes $\mathsf{IBE.ct}_{k,b}$ for $k \in [n], b \in \{0,1\}$ as Equation (5.5).
4. It computes $\mathsf{LOTFE.ct}^\tau$ for $\tau \in [i-1]$ as Equation (5.6).

5. It computes $\mathsf{LOTFE.ct}^i$ as follows. To do so, it submits $x{:=}(N, L^{i+1}, \mathsf{lab}^{i+1}_{n,0}, \mathsf{lab}^{i+1}_{n,1})$ to the challenger. Then,

$$\mathsf{LOTFE.ct}^i \leftarrow \begin{cases} \mathsf{LOTFE.Enc}(\mathsf{LOTFE.mpk}, x), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{LOTFE,B}}(1^\lambda) \\ \mathsf{LOTFE.SimEnc}(\mathsf{LOTFE.mpk}, \mathcal{V}_{\mathsf{LOTFE}}, 1^\lambda), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{LOTFE,LOTFE.Sim}}(1^\lambda) \end{cases}$$

   where $\mathcal{V}_{\mathsf{LOTFE}} := \{R_{\mathsf{LOTFE}}(x, D), D, \mathsf{LOTFE.sk}\}$ is computed and returned to B.
6. It sets gout^τ as Equation (5.7) for $\tau \in [i]$.
7. It computes $\widetilde{\mathsf{SC}}^\tau$ for $\tau \neq i$ as Equation (5.8) and $\widetilde{\mathsf{SC}}^i$ as Equation (5.9) using $\{\mathsf{gout}^\tau\}_{\tau\in[i]}$, $\{\mathsf{R}^\tau\}_{\tau\neq i}$, $\{\mathsf{lab}^\tau_{k,b}\}_{\tau,k,b}$.
8. It returns $\mathsf{ct}{:=}(\{\widetilde{\mathsf{SC}}^\tau\}_\tau, \{\mathsf{IBE.ct}_{k,b}\}_{k,b})$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

We can see that B simulates $\mathsf{Game}_{2,i,2}$ for A if it is in the real world and $\mathsf{Game}_{2,i,3}$ if it is in the ideal world by observing that

$$R_{\mathsf{LOTFE}}(x, D) = R_{\mathsf{LOTFE}}((N, L^{i+1}, \mathsf{lab}^{i+1}_{n,0}, \mathsf{lab}^{i+1}_{n,1}), D) = (N, \mathsf{lab}^{i+1}_{n,D[L^{i+1}]}) = (N, \mathsf{lab}^{i+1}_{n,X^{i+1}_n}).$$

Thus, B breaks the 1-NA-SIM security of LOTFE if A distinguishes between the two games.

Lemma 5.8. *If* LOTFE *is* 1-NA-SIM *secure, we have* $|\Pr[\mathcal{E}_{2,i,3}] - \Pr[\mathcal{E}_{2,i+1,1}]| \leq \mathrm{negl}(\lambda)$ *for* $i \in [t]$.

Proof. We observe that these two games are different only in how $\mathsf{LOTFE.ct}^i$ is computed. In particular, it is computed as

$$(\mathsf{LOTFE.ct}^i, \mathsf{LOTFE.st}) \leftarrow \mathsf{LOTFE.SimEnc}\left(\mathsf{LOTFE.mpk}, \mathcal{V}_{\mathsf{LOTFE}}, 1^\lambda\right),$$

with $\mathcal{V}_{\mathsf{LOTFE}}{:=}\{(N, \mathsf{lab}^{i+1}_{n,X^{i+1}_n}), \mathsf{LOTFE.mpk}, \mathsf{LOTFE.sk}\}$ in $\mathsf{Game}_{2,i,3}$ whereas it is computed as

$$\mathsf{LOTFE.ct}^i \leftarrow \mathsf{LOTFE.Enc}\left(\mathsf{LOTFE.mpk}, \left(|D|, 1, \mathsf{lab}^{i+1}_{n,X^{i+1}_n}, \mathsf{lab}^{i+1}_{n,X^{i+1}_n}\right); \mathsf{R}^i\right)$$

in $\mathsf{Game}_{2,i+1,1}$. We can prove that this change is unnoticed by A by almost the same proof as that for Lemma 5.7, where we change B so that it submits $x{:=}(N, 1, \mathsf{lab}^{i+1}_{n,X^{i+1}_n}, \mathsf{lab}^{i+1}_{n,X^{i+1}_n})$ instead of $x = (N, L^{i+1}, \mathsf{lab}^{i+1}_{n,0}, \mathsf{lab}^{i+1}_{n,1})$ to the challenger.

5.3 FE for Turing Machines with Fixed Input Length

Here, we show that CPRAMFE we constructed in Section 5.2 can easily be converted into FE for TM. The resulting construction can handle TM of unbounded size, but it is only 1-NA-SIM secure and can only handle the case where the length of $(x, 1^t)$ is bounded. These limitations will be removed in the next subsection.

RAM Programs reading multi-bit at once. To simplify the description, we assume that each step of RAM computation reads a block consisting of $B(\lambda) = \mathrm{poly}(\lambda)$ bits at once instead of reading a single bit. Correspondingly, we assume that the database contains B bits of data at a single location. This is without loss of generality because a RAM program that reads single bit at once can be converted into that reads B bits at once by making the length of step circuits B times longer and increasing the size of each step circuit so that it can keep B bits inside it.

Representing Turing machine computation as RAM computation. In order to represent the computation executed by a Turing machine as a computation by RAM program, we introduce the following mappings:

f : This mapping takes as input $(x \in \{0,1\}^n, 1^t)$ and then convert it into a read only RAM program $P_{(x,1^t)} = \{P^\tau_{(x,1^t)}\}_{\tau\in[t]}$ defined as in Figure 10. Here, we set $B(\lambda) = 3\lambda$. In the circuit, q, q'_0, and q'_1 are represented by strings in $\{0,1\}^\lambda$. In particular, this means that the circuit can handle any size of Turing machines because we can assume $q \leq Q < 2^\lambda$ without loss of generality.

$P^{\tau}_{(x,1^t)}$

Hardwired constants: The step number τ, the input x to the TM, and the running time t.
Input: $(\mathsf{st}, \mathsf{rData}) \in \{0,1\}^{t+2\lambda} \times \{0,1\}^{B}$.

1. Parse the input as $\mathsf{st} \to (i, W, q)$, where $i, q \in \{0,1\}^{\lambda}$ and $W \in \{0,1\}^{t}$ and $\mathsf{rData} \to ((q'_0, b'_0, \Delta i_0), (q'_1, b'_1, \Delta i_1), \mathsf{pre}_0, \mathsf{pre}_1)$.
2. If $\tau = 1$, replace W with $W = x \| 0^{t-n}$.
3. Set $i' \coloneqq i + \Delta i_{W[i]}$, $q' = q'_{W[i]}$, $\mathsf{pre}' = \mathsf{pre}_{W[i]}$, and $W'[j] = \begin{cases} W[j] & \text{if } j \neq i \\ b'_{W[i]} & \text{if } j = i \end{cases}$ for $j \in [t]$.
4. Set $(\mathsf{st}', L) = ((i', W', q'), q')$.
5. Output $\begin{cases} (\mathsf{st}', L) & \text{if } \tau \neq t \\ \mathsf{pre}' & \text{if } \tau = t \end{cases}$

Fig. 10 Circuit $P^{\tau}_{(x,1^t)}$

g : This mapping takes as input description of a Turing machine $M = (Q, \delta, F)$ and outputs a database D_M that contains Q blocks each consisting of B bits. At its q-th block, D_M contains

$$D_M[q] \coloneqq (\delta(q,0), \delta(q,1), \mathsf{pre}_0, \mathsf{pre}_1),$$

where $\mathsf{pre}_c \in \{0,1\}$ for $c \in \{0,1\}$ indicates whether q'_c defined by $\delta(q,c) = (q'_c, b'_c, \Delta i_c)$ is in the set of accepting states F or not. Since $\delta(q,b) \in [Q] \times \{0,1\} \times \{0, \pm 1\}$ and $Q < 2^{\lambda}$, each block can be represented by a binary string of length at most $B(\lambda) = 3\lambda$.

We observe that the output of $P^{D_M}_{(x,1^t)}$ is the same as that obtained by running the Turing machine M on input x for t steps. This is because each step circuit $P^{\tau}_{(x,1^t)}$ of $P_{(x,1^t)}$ is designed to emulate τ-th step of the computation done by the machine. This means that by applying the above mappings, we can convert CPRAMFE into an FE scheme for Turing machine with fixed input length. It is easy to see that the security and correctness of the scheme are preserved. In particular, the resulting scheme inherits the 1-NA-SIM security. We also observe that the size of the program $P = \{P^{\tau}_{(x,1^t)}\}_{\tau}$ is bounded by a fixed polynomial in $|(x, 1^t)|$.

5.4 Getting the Full-Fledged Construction

Here, we remove the restrictions from the consturction in Section 5.3 and obtain full-fledged FE scheme for TM.

Removing the non-adaptive and single key restriction. In the first step, we apply the conversion in Section 6.4, which is essentially the same as the conversion given by Agrawal et. al [AMVY21, Section 4], to the scheme to upgrade the security. The resulting scheme is AD-SIM secure against bounded dynamic collusion. We refer to Section 6.4 for the details.

Removing the fixed input length restriction. Our goal in the second step is constructing an FE scheme for $R^{\leq} : \mathcal{A} \times \mathcal{B} \to \mathcal{M} \cup \{\bot\}$, where $\mathcal{A} = \{0,1\}^*$, $\mathcal{B}$ is the set of all Turing machines, and

$$R^{\leq}((x,1^t), M) = \begin{cases} 1 & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge (|(x,1^t)| \leq |M|) \\ 0 & \text{otherwise.} \end{cases}$$

This step is done in essentially the same manner as [AMVY21, Section 6.2.2] using Theorem 2.1. We observe that the FE scheme that we obtained above can be seen as an FE scheme for $\mathsf{prm} = 1^i$, $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\}$ where $\mathcal{X}_i = \{0,1\}^i$ and $\mathcal{Y}_i$ is the set of all Turing machines, and

$$R_i((x,1^t), M) = \begin{cases} 1 & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \\ 0 & \text{otherwise.} \end{cases}$$

That is, $|(x, 1^t)|$ is a-priori bounded by i. We set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = i, \quad \mathcal{T}(i) = \{1, \ldots, i\}, \quad f(x, 1^t) = (x, 1^t), \quad g(M) = \{M\}_{i \in [|M|]}.$$

Here, we crucially rely on $|(x, 1^t)| \leq |M|$.

Recall that

$$R^{\mathsf{bndl}}(x, y) = \{R_i(f(x)_i, g(y)_i)\}_{i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y|)}, \tag{5.10}$$

where $f(x)_i \in \mathcal{X}_i$, and $g(y)_i \in \mathcal{Y}_i$ are the i-th entries of $f(x)$ and $g(x)$, respectively.

It is easy to see that R^{bndl} is equivalent to $R^{\leq}$ except for the case $|(x, 1^t)| > |M|$. In this case, the decryption outputs an empty set $\emptyset$. However, the output should be 0 in FE for $R^{>}$. This issue can be easily fixed as observed by Agrawal et al. [AMVY21, Section 6.2.2]. Namely, we modify the decryption algorithm so that it outputs 0 if the decryption result is $\emptyset$. We note that the resulting scheme inherits AD-SIM security, which is guaranteed by Theorem 2.1.

Removing the shorter input length restriction. In the above construction, there is a restriction that the decryption is possible only when $|(x, 1^t)| \leq |M|$. To remove the restriction, we first construct an AD-SIM secure FE scheme for TM such that the decryption is possible only when $|(x, 1^t)| > |M|$. Such a scheme can be obtained by applying the conversion by Agrawal et al. [AMVY21, Section 6.1 and 6.2.1] to AD-SIM secure CPFE with dynamic bounded collusion for $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ obtained in Section 4. We then combine these two schemes to obtain the full-fledged scheme without the restriction by applying the conversion by Agrawal et al. [AMVY21, Section 6.2.3]. Then, we obtain AD-SIM secure FE for TM. Based on the discussion above, we obtain the following theorem:

Theorem 5.4. *Assuming IBE with* AD-IND *security, ABE for circuits with circuit class $\mathcal{C}$ with* Sel-IND *security, updatable LOT (as per Definition A.1), and PIR (as per Definition 2.9) whose answer function is in $\mathcal{C}$, we have FE for TM with* AD-SIM *security against dynamic bounded collusion.*

6 AD-IND ABE for Turing Machines with Dynamic Bounded Collusion

In this section, we construct an ABE scheme for Turing machines with dynamic bounded collusion. Our scheme achieves AD-IND security and based on IBE and LOT. Previously, AD-IND security was possible only in the random oracle model [GSW21]. Recall that in ABE for TM, a ciphertext encrypting a message m is associated with an attribute x and a time bound t and a secret key is with a TM M and the decryption is possible iff M accepts x in t steps. Similarly to the case of FE, we construct two schemes that handle the case of $|(x, 1^t)| > |M|$ and $|(x, 1^t)| \leq |M|$ respectively and then combine them to obtain the full-fledged scheme.

6.1 Dealing with longer input case

We first consider the case where $|(x, 1^t)| > |M|$. We call the case longer input case. In Section 4, we obtain AD-SIM secure CPFE with dynamic bounded collusion for $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ with parameter $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{out}})$ for arbitrary polynomials $\mathsf{inp} = \mathsf{inp}(\lambda)$ and $\mathsf{out} = \mathsf{out}(\lambda)$ where $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ is the set of all polynomial size circuits with input length inp and output length out. As shown by Agrawal et al. [AMVY21, Section 6.1 and 6.2.1], we can obtain FE for TM in the case of $|(x, 1^t)| > |M|$ from the CPFE scheme for $\mathcal{C}_{\mathsf{inp},\mathsf{out}}$ above. By specializing the FE to the setting of ABE, this trivially implies ABE for TM for the case of $|(x, 1^t)| > |M|$. The resulting scheme is AD-SIM secure against dynamic bounded collusion, since so is the CPFE scheme.

6.2 Dealing with shorter input case

We then consider the case where $|(x, 1^t)| \leq |M|$. We call the case shorter input case. Our starting point is the following theorem by Goyal et al. [GSW21], which shows that we can construct single-key non-adaptive ABE for TM from IBE in the standard model by using the ideas from garbled RAM scheme [GHRW14a, LO14, GOS18].

Theorem 6.1 ([GSW21]). *If there exists* AD-IND *secure IBE, there exists* 1-NA-IND *ABE for TM.*

As we observed in Remark 2.5, 1-NA-IND security for ABE immediately implies 1-NA-SIM security. We therefore have ABE for TM with 1-NA-SIM security from the same assumption. However, the security is still weak since it only supports single key and is in the non-adaptive setting. To upgrade the security, we apply two conversions in the following.

Removing the non-adaptive and single key restriction. In the first step, we upgrade the security of the scheme to be AD-SIM security against dynamic bounded collusion. This can be done by using essentially the same conversion as that given by Agrawal et. al [AMVY21, Section 4], whose description is deferred to Section 6.4 and skipped here. However, the conversion requires that the messages to be encrypted have of fixed polynomial length, which is not the case for the above 1-NA-SIM secure FE scheme when we consider x and t with unbounded length. Therefore, we restrict the above 1-NA-SIM secure FE scheme to the setting where $|(x, 1^t)|$ is of fixed polynomial length and then apply the conversion.[19] After applying the conversion, we have ABE for TM with AD-SIM security against dynamic collusion. However, the resulting scheme inherits the property that $|(x, 1^t)|$ is bounded by some fixed polynomial chosen at the setup. This restriction will be removed in the next step.

Removing the bounded input length restriction. Here, we apply the conversions by Agrawal et. al [AMVY21, Section 6.1 and 6.2.2], which is summarized in Theorem 2.1, to the ABE scheme obtained above. As a result, we obtain ABE for TM with AD-SIM security against dynamic bounded collusion for the case of $|(x, 1^t)| \leq |M|$. Note that $(x, 1^t)$ is *not* a-priori bounded here.

More specifically, our goal here is to construct an FE scheme for $R^{\leq} : \mathcal{A} \times \mathcal{B} \to \mathcal{M} \cup \{\bot\}$, where $\mathcal{A} = \{0,1\}^*$, $\mathcal{B}$ is the set of all Turing machines, and

$$R^{\leq}((x, 1^t, m), M) = \begin{cases} (x, 1^t, m) & \text{(if } M \text{ accepts } x \text{ in } t \text{ steps)} \wedge (|(x, 1^t)| \leq |M|) \\ (x, 1^t, \bot) & \text{otherwise.} \end{cases}$$

We observe that we can use the ABE scheme obtained above as an FE scheme for $\mathsf{prm} = 1^i$, $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \mathcal{M} \cup \{\bot\}$ where $\mathcal{X}_i = \{0,1\}^i \times \{0,1\}$ and $\mathcal{Y}_i$ is the set of all Turing machines, and

$$R_i((x, 1^t, m), M) = \begin{cases} (x, 1^t, m) & \text{(if } M \text{ accepts } x \text{ in } t \text{ steps)} \\ (x, 1^t, \bot) & \text{otherwise.} \end{cases}$$

That is, $|(x, 1^t)|$ is a-priori bounded by i. We set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = i, \quad \mathcal{T}(i) = \{1, \ldots, i\}, \quad f(x, 1^t, m) = (x, 1^t, m), \quad g(M) = \{M\}_{i \in [|M|]}.$$

Here, we crucially rely on $|(x, 1^t)| \leq |M|$.

Recall that

$$R^{\mathsf{bndl}}(x, y) = \{R_i(f(x)_i, g(y)_i)\}_{i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y|)}, \tag{6.1}$$

where $f(x)_i \in \mathcal{X}_i$ and $g(y)_i \in \mathcal{Y}_i$ are the i-th entries of $f(x)$ and $g(x)$, respectively. It is easy to see that R^{bndl} is equivalent to $R^{\leq}$ except for the case $|(x, 1^t)| > |M|$. In this case, the decryption outputs an empty set $\emptyset$. However, the output should be $(x, 1^t, \bot)$ in FE for $R^{>}$. This issue can be easily fixed as observed by Agrawal et al. [AMVY21, Section 6.2.2]. Namely, we modify the decryption algorithm so that it outputs $(x, 1^t, \bot)$ if the decryption result is $\emptyset$.

By Theorem 2.1, the resulting scheme is an ABE scheme for TM with shorter input and is AD-SIM secure against dynamic bounded collusion.

6.3 Getting the Full-Fledged Construction

Finally, we present out full-fledged construction of ABE for TM with dynamic bounded collusion. We can obtain scheme with AD-SIM security and scheme with AD-IND security. To do so, we combine the two schemes described in Sections 6.1 and 6.2 using Theorem 2.1 again.

[19] Recall that we assume that the plaintext is a single bit. Therefore, the length of $(x, 1^t, m)$ is of fixed length as well. If we consider plaintext m of unbounded length, the conversion here does not work.

We set $\mathcal{A} = \{0,1\}^*$ and $\mathcal{B}$ to be the set of all Turing machines. We also set $R_1 = R^{>}$, $R_2 = R^{\leq}$, $\mathcal{X}_i = \mathcal{A}$, $\mathcal{Y}_i = \mathcal{B}$ for $i = 1, 2$, and

$$\mathcal{S}(i) = \{1,2\}, \quad \mathcal{T}(i) = \{1,2\}, \quad f(x, 1^t, m) = \{(x, 1^t, m), (x, 1^t, m)\}, \quad g(M) = \{M, M\}.$$

We consider the following relation.

$$R^{\mathsf{bndl}}((x, 1^t, m), M) = \begin{cases} ((x, 1^t, m), (x, 1^t, \bot)) & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge (|(x, 1^t)| > |M|) \\ ((x, 1^t, \bot), (x, 1^t, m)) & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge (|(x, 1^t)| \leq |M|) \\ ((x, 1^t, \bot), (x, 1^t, \bot)) & \text{otherwise.} \end{cases}$$

The scheme in Sections 6.1 and 6.2 can handle $R_1 = R^{>}$ and $R_2 = R^{\leq}$, respectively.

The relation R^{bndl} is not exactly the same as ABE for Turing machines. However, we can easily modify it into the standard one by using the trick by Agrawal et al. [AMVY21, Section 6.2.3]. We add an extra step to the decryption algorithm where it checks whether there is $m \in \{0,1\}$ in the left or right slot of the decryption results and outputs m if there is. Otherwise (both slots are $\bot$), it outputs $\bot$.

By Theorem 2.1, we can obtain AD-SIM secure ABE for TM with dynamic bounded collusion for single-bit messages since so are the underlying schemes. To expand the message space to $\{0,1\}^\ell$, we run the scheme for ℓ times and encrypt each bit of a message in parallel with the same attribute. In order to encrypt messages with unbounded length, we switch to the indistinguishability setting and use the hybrid encryption technique. Namely, we encrypt a random secret key of secret key encryption scheme with fixed polynomial length and then use it to encrypt longer plaintext. Because AD-SIM security trivially implies AD-IND security, the resulting scheme satisfies AD-IND security assuming the security of the secret key encryption.

6.4 Conversion from 1-NA-SIM to AD-SIM with dynamic bounded collusion for FE with bounded message length

Here, we show that 1-NA-SIM secure FE with "bounded message length" can be converted into FE with AD-SIM security against dynamic bounded collusion while preserving the relation being supported. Here, "bounded message length" means that the length of messages to be encrypted is bounded by some fixed polynomial. The conversion is essentially the same as that of Agrawal et. al [AMVY21, Section 4], but the description here is a bit more general so that we can capture the applications of the conversion both in Sections 5.3 and 6.2. More formal description follows.

Let us consider an 1-NA-SIM secure FE $\mathsf{1FE} = (\mathsf{1FE.Setup}, \mathsf{1FE.KeyGen}, \mathsf{1FE.Enc}, \mathsf{1FE.Dec})$ for relation $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$. We require that all messages x in $x \in \mathcal{X}_{\mathsf{prm}}$ satisfies $|x| \leq \mathrm{poly}(|\mathsf{prm}|)$ for some fixed polynomial. This in particular implies that the encryption algorithm runs in time $\mathrm{poly}(\lambda, |\mathsf{prm}|)$, since it is required to run in polynomial time in its input length by the syntax. It is easy to check that the schemes in Sections 5.3 and 6.2 satisfy the requirement:

- In Section 6.2, we set $\mathsf{prm} = 1^i$ and $\mathcal{X}_{\mathsf{prm}}$ consists of $(x, 1^t, m)$ with $|(x, 1^t)| \leq i$ and $m \in \{0,1\}$. We therefore have $|(x, 1^t, m)| \leq i + 1$.
- In Section 5.3, we set $\mathsf{prm} = 1^i$ and $\mathcal{X}_{\mathsf{prm}}$ consists of $(x, 1^t)$ with $|(x, 1^t)| \leq i$.

We can obtain an FE scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for the same relation $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$ with AD-SIM security against dynamic bounded collusion by combining 1FE with the following ingredients.

Ingredients.

1. A CPFE scheme denoted by $\mathsf{CPFE} = (\mathsf{CPFE.Setup}, \mathsf{CPFE.KeyGen}, \mathsf{CPFE.Enc}, \mathsf{CPFE.Dec})$ for bounded polynomial sized circuits that supports delayed collusion bounded property and satisfies AD-SIM security. Formally, we use $\mathsf{prm}' = (1^{\mathsf{inp}'}, 1^{\mathsf{size}'})$, $\mathcal{X}_{\mathsf{prm}'} = \mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$, $\mathcal{Y}_{\mathsf{prm}'} = \{0,1\}^{\mathsf{inp}'}$ and $R_{\mathsf{prm}'} : \mathcal{X}_{\mathsf{prm}'} \times \mathcal{Y}_{\mathsf{prm}'} \to \{0,1\}^*$, where $R(C, x) = C(x)$, for all $C \in \mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$ and $x \in \{0,1\}^{\mathsf{inp}'}$. We can obtain CPFE from IBE by the result of Agrawal et al. [AMVY21, Section 3.3 and 3.4].

Circuit $\mathsf{E}_{[x,\mathsf{K}]}$

Hardwired constants: Message x, and PRF key K.
Input: $\mathsf{str} \in \{0,1\}^{\mathsf{inp}'}$

(a) Compute $\mathsf{R} = \mathsf{PRF.Eval}(\mathsf{K}, \mathsf{str})$.
(b) Output $\mathsf{1FE.ct} = \mathsf{1FE.Enc}(\mathsf{str}, x; \mathsf{R})$, where R is used as a randomness for the encryption algorithm.

Fig. 11 Circuit $\mathsf{E}_{[x,\mathsf{K}]}$

2. A PRF $(\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ with the domain $\{0,1\}^{\mathsf{inp}'}$ and range $\mathcal{R}_{\mathsf{1FE.Enc}}$, where $\mathcal{R}_{\mathsf{1FE.Enc}}$ is the randomness space of $\mathsf{1FE.Enc}$.

The description of our ABE scheme FE is given below. In this construction, $|x|$ is a-priori bounded by some fixed polynomial $\mathrm{poly}(|\mathsf{prm}|)$.

$\mathsf{Setup}(1^\lambda, \mathsf{prm})$: On input the security parameter λ and a parameter prm, do the following:
1. Compute $\mathsf{inp}' := \mathsf{inp}'(\lambda, \mathsf{prm})$, which is the length of the master public key for $\mathsf{1FE}$ output by $\mathsf{1FE.Setup}(1^\lambda, \mathsf{prm})$, and $\mathsf{size}' := \mathsf{size}'(\lambda)$, which is the size of the circuit $\mathsf{E}_{[x,\mathsf{K}]}$ described in Figure 11, and set $\mathsf{prm}' := (1^{\mathsf{inp}'}, 1^{\mathsf{size}'})$. Note that inp' and size' are bounded by fixed polynomials since the running rime of $\mathsf{1FE.Setup}$ depends only on λ and prm, and $|x|$ is bounded by $\mathrm{poly}(\mathsf{prm})$.
2. Run $(\mathsf{cpfe.mpk}, \mathsf{cpfe.msk}) \leftarrow \mathsf{CPFE.Setup}(1^\lambda, \mathsf{prm}')$.
3. Output $(\mathsf{mpk}, \mathsf{msk}) := (\mathsf{cpfe.mpk}, \mathsf{cpfe.msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, y)$: On input the master secret key $\mathsf{msk} = \mathsf{cpfe.msk}$, a key attribute $y \in \mathcal{Y}_{\mathsf{prm}}$, do the following:
1. Compute $(\mathsf{1FE.mpk}, \mathsf{1FE.msk}) \leftarrow \mathsf{1FE.Setup}(1^\lambda, \mathsf{prm})$.
2. Generate $\mathsf{1FE.sk} \leftarrow \mathsf{1FE.KeyGen}(\mathsf{1FE.msk}, y)$.
3. Generate $\mathsf{cpfe.sk} \leftarrow \mathsf{CPFE.KeyGen}(\mathsf{cpfe.msk}, \mathsf{1FE.mpk})$.
4. Output $\mathsf{sk} := (\mathsf{1FE.sk}, \mathsf{cpfe.sk})$.

$\mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$: On input the master public key $\mathsf{mpk} = \mathsf{cpfe.mpk}$, a message $x \in \mathcal{X}_{\mathsf{prm}}$, and the upper bound for the collusion $1 \leq Q \leq 2^\lambda$ in unary form, do the following:
1. Sample a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
2. Construct the circuit $\mathsf{E}_{[x,\mathsf{K}]}$ defined as Figure 11.
3. Compute $\mathsf{cpfe.ct} \leftarrow \mathsf{CPFE.Enc}(\mathsf{cpfe.mpk}, \mathsf{E}_{[x,\mathsf{K}]}, 1^Q)$.
4. Output $\mathsf{ct} := \mathsf{cpfe.ct}$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$: On input a ciphertext $\mathsf{ct} = \mathsf{cpfe.ct}$ and a secret key $\mathsf{sk} = (\mathsf{1FE.sk}, \mathsf{cpfe.sk})$, do the following:
1. Compute $\alpha \leftarrow \mathsf{CPFE.Dec}(\mathsf{cpfe.sk}, \mathsf{cpfe.ct})$.
2. Compute and output $z := \mathsf{1FE.Dec}(\mathsf{1FE.sk}, \alpha)$.

Correctness.

Theorem 6.2. *Assume* $|x| \leq \mathrm{poly}(|\mathsf{prm}|)$, $\mathsf{1FE}$ *supports the relation* $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$, *and* CPFE *supports the circuit class* $\mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$. *Then, the FE scheme* FE *satisfies the correctness.*

It is easy to see this holds by correcntess of CPFE and $\mathsf{1FE}$.

Security.

Theorem 6.3. *If* PRF *is a secure pseudorandom function,* $\mathsf{1FE}$ *is* 1-NA-SIM *secure, and* CPFE *is* Q-AD-SIM *secure with dynamic bounded collusion,* FE *is* Q-AD-SIM *secure with dynamic bounded collusion.*

We can prove this theorem in the same way as the proof by Agrawal et al. [AMVY21, Theorem C.3 in the full version]. The only difference between the transformatin and theirs is that we use a general form of FE instead of a succinct KPFE. The security level is the same (1-NA-SIM). Thus, we omit the proof.

References

AFS19. P. Ananth, X. Fan, and E. Shi. Towards Attribute-Based Encryption for RAMs from LWE: Sub-linear Decryption, and More. In *ASIACRYPT*, 2019.

Agr17. S. Agrawal. Stronger Security for Reusable Garbled Circuits, New Definitions and Attacks. In *CRYPTO*, 2017.

AGVW13. S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional Encryption: New Perspectives and Lower Bounds. In *CRYPTO*, 2013.

AJ15. P. Ananth and A. Jain. Indistinguishability Obfuscation from Compact Functional Encryption. In *CRYPTO*, 2015.

AL18. P. Ananth and A. Lombardi. Succinct Garbling Schemes from Functional Encryption Through a Local Simulation Paradigm. In *TCC*, 2018.

AM18. S. Agrawal and M. Maitra. FE and iO for Turing Machines from Minimal Assumptions. In *TCC*, 2018.

AMVY21. S. Agrawal, M. Maitra, N. S. Vempati, and S. Yamada. Functional Encryption for Turing Machines with Dynamic Bounded Collusion from LWE. In *CRYPTO*, 2021.

AMY19. S. Agrawal, M. Maitra, and S. Yamada. Attribute based encryption (and more) for nondeterministic finite automata from LWE. In *CRYPTO*, 2019.

AS16. P. Ananth and A. Sahai. Functional Encryption for Turing Machines. In *TCC*, 2016.

AS17. S. Agrawal and I. P. Singh. Reusable Garbled Deterministic Finite Automata from Learning With Errors. In *ICALP*, 2017.

AV19. P. Ananth and V. Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In *TCC*, 2019.

BCH84. P. Beame, S. Cook, and H. Hoover. Log Depth Circuits For Division And Related Problems. In *FOCS*, 1984.

BGG+14. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits. In *EUROCRYPT*, 2014.

BGI16. E. Boyle, N. Gilboa, and Y. Ishai. Breaking the Circuit Size Barrier for Secure Computation Under DDH. In *CRYPTO*, 2016.

BLSV18. Z. Brakerski, A. Lombardi, G. Segev, and V. Vaikuntanathan. Anonymous IBE, Leakage Resilience and Circular Security from New Assumptions. In *EUROCRYPT*, 2018.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional Encryption: Definitions and Challenges. In *TCC*, 2011.

BV11. Z. Brakerski and V. Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *FOCS*, 2011.

BV18. N. Bitansky and V. Vaikuntanathan. Indistinguishability Obfuscation from Functional Encryption. *J. ACM*, 65(6):39:1–39:37, 2018.

CDG+17. C. Cho, N. Döttling, S. Garg, D. Gupta, P. Miao, and A. Polychroniadou. Laconic Oblivious Transfer and Its Applications. In *CRYPTO*, 2017.

CIJ+13. A. D. Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano. On the Achievability of Simulation-Based Security for Functional Encryption. In *CRYPTO*, 2013.

DG17a. N. Döttling and S. Garg. From selective IBE to full IBE and selective HIBE. In *TCC*, 2017.

DG17b. N. Döttling and S. Garg. Identity-based encryption from the Diffie-Hellman assumption. In *CRYPTO*, 2017.

DGHM18. N. Döttling, S. Garg, M. Hajiabadi, and D. Masny. New Constructions of Identity-Based and Key-Dependent Message Secure Encryption Schemes. In *PKC*, 2018.

DGI+19. N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor Hash Functions and Their Applications. In *CRYPTO*, 2019.

GGLW21. R. Garg, R. Goyal, G. Lu, and B. Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In Eprint 2021/847, 2021. To appear in Eurocrypt'22.

GHRW14a. C. Gentry, S. Halevi, M. Raykova, and D. Wichs. Garbled RAM Revisited, Part I. In *EUROCRYPT*, 2014.

GHRW14b. C. Gentry, S. Halevi, M. Raykova, and D. Wichs. Outsourcing Private RAM Computation. In *FOCS*, 2014.

GKW16. R. Goyal, V. Koppula, and B. Waters. Semi-Adaptive Security and Bundling Functionalities Made Generic and Easy. In *TCC*, 2016.

Gol08. O. Goldreich. Computational complexity: a conceptual perspective. *In ACM Sigact News*, 2008.

GOS18. S. Garg, R. Ostrovsky, and A. Srinivasan. Adaptive Garbled RAM from Laconic Oblivious Transfer. In *CRYPTO*, 2018.

GPSW06. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, 2006.

GS18. S. Garg and A. Srinivasan. Adaptively Secure Garbling with Near Optimal Online Complexity. In *EUROCRYPT*, 2018.

GSW13. C. Gentry, A. Sahai, and B. Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO*, 2013.

GSW21. R. Goyal, R. Syed, and B. Waters. Bounded Collusion ABE for TMs from IBE. In *ASIACRYPT*, 2021.

GTKP+13a. S. Goldwasser, Y. Tauman Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. How to Run Turing Machines on Encrypted Data. In *CRYPTO*, 2013.

GTKP+13b. S. Goldwasser, Y. Tauman Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

GV15. S. Gorbunov and D. Vinayagamurthy. Riding on Asymmetry: Efficient ABE for Branching Programs. In *ASIACRYPT*, 2015.

GVW12. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional Encryption with Bounded Collusions from Multiparty Computation. In *CRYPTO*, 2012.

GVW13. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute Based Encryption for circuits. In *STOC*, 2013.

HJO+16. B. Hemenway, Z. Jafargholi, R. Ostrovsky, A. Scafuro, and D. Wichs. Adaptively Secure Garbled Circuits from One-Way Functions. In *CRYPTO*, 2016.

KNTY19. F. Kitagawa, R. Nishimaki, K. Tanaka, and T. Yamakawa. Adaptively Secure and Succinct Functional Encryption: Improving Security and Efficiency, Simultaneously. In *CRYPTO*, 2019.

KO97. E. Kushilevitz and R. Ostrovsky. Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In *FOCS*, 1997.

LL20. H. Lin and J. Luo. Compact Adaptively Secure ABE from k-Lin: Beyond NC1 and Towards NL. In *EUROCRYPT*, 2020.

LO14. S. Lu and R. Ostrovsky. How to Garble RAM Programs. In *EUROCRYPT*, 2014.

PRSD17. C. Peikert, O. Regev, and N. Stephens-Davidowitz. Pseudorandomness of Ring-LWE for Any Ring and Modulus. In *STOC*, 2017.

Reg09. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *In Journal of the ACM*, 2009.

SW05. A. Sahai and B. Waters. Fuzzy Identity-Based Encryption. In *EUROCRYPT*, 2005.

Yao86. A. C. Yao. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*, 1986.

APPENDICES

A Gate-by-Gate Garbling from LOT

In this section, we prove Lemma 3.1. That is, we construct gate-by-gate garbling with pebbling-based simulation security from updatable LOT.

A.1 Updatable Laconic OT

We introduce updatable laconic OT, which will be used as a building block in our construction.

Definition A.1 (Updatable Laconic OT). *An updatable laconic OT (LOT) protocol with message-length $p = p(\lambda)$ consists of six PPT algorithms* $\mathsf{LOT} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Send}, \mathsf{Receive}, \mathsf{SendWrite}, \mathsf{ReceiveWrite})$.

$\mathsf{Gen}(1^\lambda) \to \mathsf{crs}$**:** *The generation algorithm takes as input the security parameter and outputs a common reference string* crs.

$\mathsf{Hash}(\mathsf{crs}, D) \to (d, \widehat{D})$**:** *The hashing algorithm is a deterministic algorithm that takes as input* crs *and a database $D \in \{0,1\}^*$ and outputs a digest d of D and a state $\widehat{D}$. We assume that $\widehat{D}$ also includes D.*

$\mathsf{Send}(\mathsf{crs}, d, L, m_0, m_1) \to e$**:** *The sending algorithm takes as input* crs*, a digest d, database location $L \in \mathbb{N}$, and two messages m_0 and m_1 of bit-length p, and outputs a ciphertext e.*

$\mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, e, L) \to m$**:** *The receiving algorithm is a RAM algorithm with random read access to $\widehat{D}$. It takes as input* crs*, e, and $L \in \mathbb{N}$, and outputs a message m.*

$\mathsf{SendWrite}(\mathsf{crs}, d, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{|d|}) \to e_w$**:** *The send-write algorithm takes as input* crs*, d, L, a bit $b \in \{0,1\}$ to be written in the database, and $|d|$ pairs of messages $\{m_{j,0}, m_{j,1}\}_{j=1}^{|d|}$, and outputs a ciphertext e_w.*

$\mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, L, b, e_w) \to \{m_j\}_{j\in[|d|]}$**:** *The receive-write algorithm is a RAM algorithm with random read and write access to $\widehat{D}$. It takes as input* crs*, $L \in \mathbb{N}$, $b \in \{0,1\}$, and e_w, updates the state $\widehat{D}$, and outputs messages $\{m_j\}_{j\in[|d|]}$.*

These algorithms satisfy the following three properties where we write $D[L]$ to mean the L-th bit of D and $d[j]$ to mean j-th bit of d.

Correctness. *For any database D of size at most $M = \mathrm{poly}(\lambda)$, any memory location $L \in [M]$, any pair of messages $(m_0, m_1) \in \{0,1\}^p \times \{0,1\}^p$, it holds that*

$$\Pr\left[m = m_{D[L]} \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \\ (d, \widehat{D}) := \mathsf{Hash}(\mathsf{crs}, D), \\ e \leftarrow \mathsf{Send}(\mathsf{crs}, d, L, m_0, m_1), \\ m \leftarrow \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, e, L) \end{array} \right] = 1.$$

Correctness with regard to Write. *For any database D of size at most $M = \mathrm{poly}(\lambda)$, any memory location $L \in [M]$, any bit $b \in \{0,1\}$, any D^* such that D^* is identical to D except that $D^*[L] = b$, any messages $\{(m_{j,0}, m_{j,1})\}_{j=1}^{|d|}$ where $m_{j,b} \in \{0,1\}^p$, it holds that*

$$\Pr\left[\forall j\ m'_j = m_{j,d^*[j]} \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \\ (d, \widehat{D}) := \mathsf{Hash}(\mathsf{crs}, D), \\ (d^*, \widehat{D}^*) := \mathsf{Hash}(\mathsf{crs}, D^*), \\ e_w \leftarrow \mathsf{SendWrite}(\mathsf{crs}, d, L, b, \{(m_{j,0}, m_{j,1})\}_{j=1}^{|d|}), \\ \{m'_j\}_{j=1}^{|d|} \leftarrow \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, L, b, e_w) \end{array} \right] = 1.$$

Moreover, $\widehat{D}$ is updated to $\widehat{D}^*$ after the above execution.

Sender Privacy against Semi-Honest Receivers. *There exists a PPT simulator* Sim *such that for any PPT stateful adversary* A*,*

$$|\Pr[\mathsf{Exp}^{\mathsf{real\text{-}sen}}_{\mathsf{LOT},\mathsf{A}}(1^\lambda)=1]-\Pr[\mathsf{Exp}^{\mathsf{ideal\text{-}sen}}_{\mathsf{LOT},\mathsf{Sim}}(1^\lambda)=1]|\leq \mathrm{negl}(\lambda),$$

where the experiments $\mathsf{Exp}^{\mathsf{real\text{-}sen}}_{\mathsf{LOT},\mathsf{A}}(1^\lambda)$ *and* $\mathsf{Exp}^{\mathsf{ideal\text{-}sen}}_{\mathsf{LOT},\mathsf{Sim}}(1^\lambda)$ *are defined as follows where* $|D|=M=\mathrm{poly}(\lambda)$*,* $L\in[M]$*, and* $m_0,m_1\in\{0,1\}^{p(\lambda)}$*.*

$\mathsf{Exp}^{\mathsf{real\text{-}sen}}_{\mathsf{LOT},\mathsf{A}}(1^\lambda)$	$\mathsf{Exp}^{\mathsf{ideal\text{-}sen}}_{\mathsf{LOT},\mathsf{Sim}}(1^\lambda)$
1. $\mathsf{crs}\leftarrow\mathsf{Gen}(1^\lambda)$	1. $\mathsf{crs}\leftarrow\mathsf{Gen}(1^\lambda)$
2. $(D,L,m_0,m_1)\leftarrow\mathsf{A}(\mathsf{crs})$	2. $(D,L,m_0,m_1)\leftarrow\mathsf{A}(\mathsf{crs})$
3. $(d,\widehat{D}):=\mathsf{Hash}(\mathsf{crs},D)$	3. $(d,\widehat{D}):=\mathsf{Hash}(\mathsf{crs},D)$
4. $e\leftarrow\mathsf{Send}(\mathsf{crs},d,L,m_0,m_1)$	4. $e\leftarrow\mathsf{Sim}(\mathsf{crs},D,L,m_{D[L]})$
5. $b\leftarrow\mathsf{A}(e)$	5. $b\leftarrow\mathsf{A}(e)$
6. Output b	6. Output b

Sender Privacy against Semi-Honest Receivers with regard to Write. *There exists a PPT simulator* $\mathsf{SimWrite}$ *such that for any stateful PPT algorithm* A*,*

$$|\Pr[\mathsf{Exp}^{\mathsf{real\text{-}wr\text{-}sen}}_{\mathsf{LOT},\mathsf{A}}(1^\lambda)=1]-\Pr[\mathsf{Exp}^{\mathsf{ideal\text{-}wr\text{-}sen}}_{\mathsf{LOT},\mathsf{Sim}}(1^\lambda)=1]|\leq \mathrm{negl}(\lambda),$$

where the experiments $\mathsf{Exp}^{\mathsf{real\text{-}wr\text{-}sen}}_{\mathsf{LOT},\mathsf{A}}(1^\lambda)$ *and* $\mathsf{Exp}^{\mathsf{ideal\text{-}wr\text{-}sen}}_{\mathsf{LOT},\mathsf{Sim}}(1^\lambda)$ *are defined as follows where* $|D|=M=\mathrm{poly}(\lambda)$*,* $L\in[M]$*, and* $m_{j,0},m_{j,1}\in\{0,1\}^{p(\lambda)}$ *for all* $j\in[|d|]$*.*

$\mathsf{Exp}^{\mathsf{real\text{-}wr\text{-}sen}}_{\mathsf{LOT},\mathsf{A}}(1^\lambda)$	$\mathsf{Exp}^{\mathsf{ideal\text{-}wr\text{-}sen}}_{\mathsf{LOT},\mathsf{Sim}}(1^\lambda)$
1. $\mathsf{crs}\leftarrow\mathsf{Gen}(1^\lambda)$	1. $\mathsf{crs}\leftarrow\mathsf{Gen}(1^\lambda)$
2. $(D,L,b,\{m_{j,0},m_{j,1}\}_{j=1}^{\lvert d\rvert})\leftarrow\mathsf{A}(\mathsf{crs})$	2. $(D,L,b,\{m_{j,0},m_{j,1}\}_{j=1}^{\lvert d\rvert})\leftarrow\mathsf{A}(\mathsf{crs})$
3. $(d,\widehat{D}):=\mathsf{Hash}(\mathsf{crs},D)$	3. $(d,\widehat{D}):=\mathsf{Hash}(\mathsf{crs},D)$
4.	4. $(d^*,\widehat{D}^*):=\mathsf{Hash}(\mathsf{crs},D^*)$ where D^* is identical to D except that $D^*[L]=b$
5. $e\leftarrow\mathsf{SendWrite}(\mathsf{crs},d,L,b,\{m_{j,0},m_{j,1}\}_{j\in[\lvert d\rvert]})$	5. $e\leftarrow\mathsf{SimWrite}(\mathsf{crs},D,L,b,\{m_{j,d^*[j]}\}_{j\in[\lvert d\rvert]})$
6. $b'\leftarrow\mathsf{A}(e)$	6. $b'\leftarrow\mathsf{A}(e)$
7. Output b'	7. Output b'

Efficiency. *We require that* $|d|$ *is bounded by a fixed polynomial in* λ *independent of* $|D|$ *and the running time of* Send*,* $\mathsf{Receive}$*,* $\mathsf{SendWrite}$*, and* $\mathsf{ReceiveWrite}$ *are* $\mathrm{poly}(\log|D|,\lambda)$*.*

Moreover, Hash *satisfies the online-offline efficiency in the following sense: There are deterministic polynomial time algorithms* $\mathsf{HashOff}$ *and* HashOn *such that for any* $D=D_1\|D_2$*, we have*

$$\Pr\left[d_{\mathsf{on}}=d\;\middle|\;\begin{array}{c}\mathsf{crs}\leftarrow\mathsf{Gen}(1^\lambda),\\ d_{\mathsf{off}}:=\mathsf{HashOff}(\mathsf{crs},D_2),\\ d_{\mathsf{on}}:=\mathsf{HashOn}(\mathsf{crs},D_1,d_{\mathsf{off}}),\\ (d,\widehat{D}):=\mathsf{Hash}(\mathsf{crs},D)\end{array}\right]=1$$

Moreover, $|d_{\mathsf{off}}|$ *is bounded by a fixed polynomial in* λ *independent of* $|D_2|$ *and the running time of* HashOn *is* $\mathrm{poly}(\log|D|,|D_1|,\lambda)$

Choi et al. [CDG+17] constructed an LOT (that does not need to satisfy correctness with regard to write and sender privacy against semi-honest receivers with regard to write) based on the DDH assumption and gave a generic conversion

from LOT to updatable LOT. Later, the assumption was improved/generalized to either of CDH, Factoring, or LWE assumptions [DG17b, DGHM18, BLSV18].

Thus, we have the following lemma.

Lemma A.1. *Assuming either of CDH, Factoring, or LWE assumptions, there exists updatable LOT.*

A.2 Construction

In this section, we prove Lemma 3.1. That is, we construct a gate-by-gate garbling scheme with (M, T)-pebbling-based simulation security for $M = \mathrm{poly}(\lambda)$ and $T = \mathrm{poly}(\lambda, \mathsf{size})$ where size is the size of a circuit being garbled assuming the existence of updatable LOT.

Remark A.1. Though [GS18] only considered circuits consisting only of NAND gates, we consider general circuits that may contain other gates. This is because we consider to garble a universal circuit $U(C, \cdot)$ in which C is embedded to hide the topology of C in our construction of a CPFE scheme in Section 4. To do so, we have to define $U(C, \cdot)$ so that the topology of $U(C, \cdot)$ does not reveal anything about C, which cannot be done only with NAND gates. Indeed, if a circuit only has NAND gates, the topology reveals the whole functionality of the circuit. Fortunately, we observe that the construction of [GS18] can be easily extended to circuits with general gates.

Ingredients. We now describe the underlying building blocks used to obtain our construction:

- A selectively secure garbling scheme $\mathsf{GC} := (\mathsf{GC.Garble}, \mathsf{GC.Eval})$.
- An updatable LOT protocol $\mathsf{LOT} := (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Send}, \mathsf{Receive}, \mathsf{SendWrite}, \mathsf{ReceiveWrite})$.

Our gate-by-gate garbling scheme $\mathsf{GC}_{\mathsf{LOT}}$ is described below where we use circuit notations defined in Section 3:

The number of randomness slots N_{rand}: For a circuit C, the number N_{rand} of randomness slots is set to be $\mathsf{inp} + 2|\mathsf{Gates}| + 1$. In the following, we identify $[N_{\mathsf{rand}}]$ with the set $\{i, \mathrm{pad}\}_{i\in[N]} \cup \{i, \mathrm{label}\}_{i\in[\mathsf{inp}+1, N+1]}$. (Recall that we define $N := \mathsf{inp} + |\mathsf{Gates}|$ and thus $N + (N + 1 - \mathsf{inp}) = \mathsf{inp} + 2|\mathsf{Gates}| + 1 = N_{\mathsf{rand}}$.)
The randomness length ℓ: The randomness length ℓ is $2\lambda|d|$.
The definition of $S(\mathsf{G})$: For a gate $\mathsf{G} = (g_{\mathsf{G}}, i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}}) \in \mathsf{Gates}$, the subset $S(\mathsf{G})$ is defined as $S(\mathsf{G}) := \{(i_{\mathsf{G}}, \mathrm{pad}), (A_{\mathsf{G}}, \mathrm{pad}), (B_{\mathsf{G}}, \mathrm{pad}), (i_{\mathsf{G}}, \mathrm{label}), (i_{\mathsf{G}} + 1, \mathrm{label})\}$.
The definition of S_{st}: The subset S_{st} is defined as $S_{\mathsf{st}} := \{(i, \mathrm{pad})\}_{i\in[\mathsf{inp}]\cup[N-\mathsf{out}+1, N]} \cup \{(\mathsf{inp} + 1, \mathrm{label})\}$.

$\mathsf{GCkt}(1^\lambda, C)$: On input the security parameter λ and a circuit C, do the following:

1. Run $\mathsf{pp} \leftarrow \mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$ where GSetup is defined below:
   $\mathsf{GSetup}(1^\lambda, \mathsf{top}(C))$ On input the security parameter λ and the topology $\mathsf{top}(C)$, do the following:[20]
   (a) Run $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$.
   (b) Compute $d_{\mathsf{off}} := \mathsf{HashOff}(\mathsf{crs}, 0^{N-\mathsf{inp}})$.
   (c) Output $\mathsf{pp} := (\mathsf{crs}, d_{\mathsf{off}})$.
2. For $(i, \mathrm{type}) \in \{i, \mathrm{pad}\}_{i\in[N]} \cup \{i, \mathrm{label}\}_{i\in[\mathsf{inp}+1, N]}$, generate $\mathsf{R}_{(i,\mathrm{type})} \leftarrow \{0,1\}^\ell$. We introduce the following notations for convenience:
   - For (i, label), we parse $\mathsf{R}_{(i,\mathrm{label})}$ as $\{\mathsf{label}^i_{k,b}\}_{k\in[|d|],b\in\{0,1\}}$ where $\mathsf{label}^i_{k,b} \in \{0,1\}^\lambda$ for each $k \in [|d|]$ and $b \in \{0,1\}$. We often just write $\{\mathsf{label}^i_{k,b}\}$ to mean $\{\mathsf{label}^i_{k,b}\}_{k\in[|d|],b\in\{0,1\}}$ for notational simplicity.
   - For (i, pad), we denote by r_i to mean the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.[21]
3. For $\mathsf{G} \in \mathsf{Gates}$, run $\widetilde{\mathsf{G}} \leftarrow \mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S(\mathsf{G})})$ where GGate is defined below:
   $\mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S(\mathsf{G})})$**:** On input the public parameter $\mathsf{pp} = (\mathsf{crs}, d_{\mathsf{off}})$, a gate $\mathsf{G} = (g_{\mathsf{G}}, i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}}) \in \mathsf{Gates}$, and a tuple of randomness $\{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S(\mathsf{G})}$, do the following:
   (a) For $i \in \{i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}}\}$, set r_i to be the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.

[20] Remark that this algorithm only need to know N and inp rather than the full description of $\mathsf{top}(C)$.
[21] The second to ℓ-th bits of $\mathsf{R}_{(i,\mathrm{pad})}$ are not used at all in the scheme. We choose $\mathsf{R}_{(i,\mathrm{pad})}$ from ℓ-bit strings just for making the scheme be consistent to the syntax defined in Definition 3.2.

Step Circuit SC

Input: A digest d
Hardwired value: crs, $(r_{A_\mathsf{G}}, r_{B_\mathsf{G}}, r_{i_\mathsf{G}})$, $\mathsf{G} = (g_\mathsf{G}, i_\mathsf{G}, A_\mathsf{G}, B_\mathsf{G})$, $\{\mathsf{label}_{k,b}\}$, and $\mathsf{flag} \in \{0,1\}$.

1. Generate $e_b \leftarrow \mathsf{SendWrite}(\mathsf{crs}, d, i_\mathsf{G}, b, \{\mathsf{label}_{k,0}, \mathsf{label}_{k,1}\}_{k\in[|d|]})$ for $b \in \{0,1\}$.
2. If $\mathsf{flag} = 0$, then $\gamma(\alpha,\beta) := g_\mathsf{G}(\alpha \oplus r_{A_\mathsf{G}}, \beta \oplus r_{B_\mathsf{G}}) \oplus r_{i_\mathsf{G}}$ for all $\alpha,\beta \in \{0,1\}$.
3. If $\mathsf{flag} = 1$, then $\gamma(\alpha,\beta) := r_{i_\mathsf{G}}$ for all $\alpha,\beta \in \{0,1\}$.
4. Generates

$$f_0 \leftarrow \mathsf{Send}(\mathsf{crs}, d, B_\mathsf{G}, (\gamma(0,0), e_{\gamma(0,0)}), (\gamma(0,1), e_{\gamma(0,1)}))$$
$$f_1 \leftarrow \mathsf{Send}(\mathsf{crs}, d, B_\mathsf{G}, (\gamma(1,0), e_{\gamma(1,0)}), (\gamma(1,1), e_{\gamma(1,1)}))$$

5. Output $\mathsf{Send}(\mathsf{crs}, d, A_\mathsf{G}, f_0, f_1)$.

Fig. 12 The description of SC.

(b) For $i \in \{i_\mathsf{G}, i_\mathsf{G}+1\}$, parse $\mathsf{R}_{(i,\mathrm{label})} = \{\mathsf{label}^i_{k,b}\}$.
(c) Compute

$$\widetilde{\mathsf{G}} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{SC}[\mathsf{crs}, (r_{A_\mathsf{G}}, r_{B_\mathsf{G}}, r_{i_\mathsf{G}}), \mathsf{G}, \{\mathsf{label}^{i_\mathsf{G}+1}_{k,b}\}, 0], \{\mathsf{label}^{i_\mathsf{G}}_{k,b}\})$$

where $\mathsf{SC}[\mathsf{crs}, (r_{A_\mathsf{G}}, r_{B_\mathsf{G}}, r_{i_\mathsf{G}}), \mathsf{G}, \{\mathsf{label}^{i_\mathsf{G}+1}_{k,b}\}, 0]$ is defined as in Figure 12. We assume that SC is of a fixed size denoted by $|\mathsf{SC}|$ independently of the hardwired values, which can be assumed without loss of generality by appropriate padding.

4. Output $\widetilde{C} = (\mathsf{crs}, \{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}})$ and $\mathsf{st} = (\mathsf{pp}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_\mathsf{st}})$.

$\mathsf{GInp}(\mathsf{st}, x)$: On input $\mathsf{st} = (\mathsf{pp}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_\mathsf{st}})$, do the following:

1. Parse $\mathsf{pp} = (\mathsf{crs}, d_\mathsf{off})$ and $\mathsf{R}_{(\mathsf{inp}+1,\mathrm{label})} = \{\mathsf{label}^{\mathsf{inp}+1}_{k,b}\}$.
2. For $i \in [\mathsf{inp}] \cup [N-\mathsf{out}+1, N]$, set r_i to be the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.
3. Compute $d := \mathsf{HashOn}(\mathsf{crs}, r_1 \oplus x_1 \| \cdots \| r_\mathsf{inp} \oplus x_\mathsf{inp}, d_\mathsf{off})$.
4. Output $\widetilde{x} := \left(\{\mathsf{label}^{\mathsf{inp}+1}_{k,d[k]}\}_{k\in[|d|]}, \{r_i \oplus x_i\}_{i\in[\mathsf{inp}]}, \{r_i\}_{i\in[N-\mathsf{out}+1,N]}\right)$.

$\mathsf{GEval}(\widetilde{C}, \widetilde{x})$: On input $\widetilde{C} = (\mathsf{crs}, \{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in\mathsf{Gates}})$ and $\widetilde{x} = \left(\{\mathsf{label}^{\mathsf{inp}+1}_{k,d[k]}\}_{k\in[|d|]}, \{r_i \oplus x_i\}_{i\in[\mathsf{inp}]}, \{r_i\}_{i\in[N-\mathsf{out}+1,N]}\right)$, do the following:

1. Set $D := r_1 \oplus x_1 \| \cdots \| r_\mathsf{inp} \oplus x_\mathsf{inp} \| 0^{N-\mathsf{inp}}$.
2. Compute $(d, \widehat{D}) := \mathsf{Hash}(\mathsf{crs}, D)$.
3. Set $\overline{\mathsf{label}} := \{\mathsf{label}^{\mathsf{inp}+1}_{k,d[k]}\}_{k\in[|d|]}$.
4. For $\mathsf{G} \in \mathsf{Gates}$, do the following in the order of $i_\mathsf{G} = \mathsf{inp}+1, \ldots, N$:
   (a) Compute $\mathsf{gout} := \mathsf{GC.Eval}(\widetilde{\mathsf{G}}, \overline{\mathsf{label}})$.
   (b) Compute $(\gamma, e) := \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{gout}, A_\mathsf{G}), B_\mathsf{G})$.
   (c) Set $\overline{\mathsf{label}} := \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, i_\mathsf{G}, \gamma, e)$.
5. Read D from $\widehat{D}$.
6. Output $D[N-\mathsf{out}+1] \oplus r_{N-\mathsf{out}+1} \| \cdots \| D[N] \oplus r_N$.

Correctness and Efficiency. It is easy to see that the above construction satisfies the syntactical and efficiency requirements in Definition 3.2.

Correctness can be proven similarly to [GS18] because the above scheme is identical to their scheme except that we remove an additional layer of somewhere equivocal encryption and we consider general gates instead of limiting to NAND (see Remark A.1). Therefore, we only give a proof sketch for correctness. By an inductive argument,

we can prove that at the point of starting the loop for $\mathsf{G} = \mathsf{G}^*$ in Step 4 of GEval, $\widehat{D}$ corresponds to the database $D_{i_{\mathsf{G}^*}} = D_{i_{\mathsf{G}^*}}[1] \| \dots \| D_{i_{\mathsf{G}^*}}[N]$ where

$$D_{i_{\mathsf{G}^*}}[i] := \begin{cases} x_i \oplus r_i & i \le \mathsf{inp} \\ E_i \oplus r_i & \mathsf{inp}+1 \le i < i_{\mathsf{G}^*} \\ 0 & \text{otherwise,} \end{cases}$$

where E_i is the bit assigned to the output wire of the gate indexed by i when we evaluate C on the input x and $\overline{\mathsf{label}}$ is a tuple of labels corresponding to the digest of $D_{i_{\mathsf{G}^*}}$. Especially, after finishing the loop for the all gates, $\widehat{D}$ corresponds to the database

$$D_{N+1}[i] := \begin{cases} x_i \oplus r_i & i \le \mathsf{inp} \\ E_i \oplus r_i & \mathsf{inp}+1 \le i \le N \end{cases}.$$

By the definition of E_i, $E_{N-\mathsf{out}+j}$ is the j-th bit of $C(x)$ for $j \in [\mathsf{out}]$. This immediately implies the correctness.

Pebbling-based Simulation Security. We construct a pebbling-based simulator $\mathsf{Sim} = (\{\mathsf{SimGate}_{\mathsf{mode}}\}_{\mathsf{mode}\in\{\mathsf{White},\mathsf{Gray},\mathsf{Black}\}}, \mathsf{SimInpConf})$ as follows:

Definitions of $S_{\mathsf{Black}}(\mathsf{G})$ and $S_{\mathsf{White}}(\mathsf{G})$: For a gate $\mathsf{G} = (g_{\mathsf{G}}, i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}}) \in \mathsf{Gates}$, we define $S_{\mathsf{Black}}(\mathsf{G}) := \{(i_{\mathsf{G}}, \mathrm{pad}), (i_{\mathsf{G}}, \mathrm{label}), (i_{\mathsf{G}}+1, \mathrm{label})\}$. Note that $S_{\mathsf{White}}(\mathsf{G}) := S(\mathsf{G}) = \{(i_{\mathsf{G}}, \mathrm{pad}), (A_{\mathsf{G}}, \mathrm{pad}), (B_{\mathsf{G}}, \mathrm{pad}), (i_{\mathsf{G}}, \mathrm{label}), (i_{\mathsf{G}}+1, \mathrm{label})\}$ as required in Definition 3.4.

$\mathsf{SimGate}_{\mathsf{White}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{White}}(\mathsf{G})})$: This is exactly the same algorithm as $\mathsf{GGate}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S(\mathsf{G})})$.

$\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{Black}}(\mathsf{G})})$: On input the public parameter $\mathsf{pp} = (\mathsf{crs}, d_{\mathsf{off}})$, a gate $\mathsf{G} = (g_{\mathsf{G}}, i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}}) \in \mathsf{Gates}$, and a tuple of randomness $\{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{Black}}(\mathsf{G})}$, do the following:

1. Set $r_{i_{\mathsf{G}}}$ to be the first bit of $\mathsf{R}_{(i_{\mathsf{G}},\mathrm{pad})}$.
2. For $i \in \{i_{\mathsf{G}}, i_{\mathsf{G}}+1\}$, parse $\mathsf{R}_{(i,\mathrm{label})} = \{\mathsf{label}^i_{k,b}\}$.
3. Compute and output

$$\widetilde{\mathsf{G}} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{SC}[\mathsf{crs}, (0,0,r_{i_{\mathsf{G}}}), \mathsf{G}, \{\mathsf{label}^{i_{\mathsf{G}}+1}_{k,b}\}, 1], \{\mathsf{label}^{i_{\mathsf{G}}}_{k,b}\})$$

where $\mathsf{SC}[\mathsf{crs}, (0,0,r_{i_{\mathsf{G}}}), \mathsf{G}, \{\mathsf{label}^{i_{\mathsf{G}}+1}_{k,b}\}, 1]$ is defined as in Figure 12.

$\mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in[N_{\mathsf{rand}}]}, C, x)$: On input the public parameter $\mathsf{pp} = (\mathsf{crs}, d_{\mathsf{off}})$, a gate $\mathsf{G} = (g_{\mathsf{G}}, i_{\mathsf{G}}, A_{\mathsf{G}}, B_{\mathsf{G}}) \in \mathsf{Gates}$, and a tuple of randomness $\{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in[N_{\mathsf{rand}}]}$, do the following:

1. For $i \in [N]$, set r_i to be the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.
2. For $i \in [\mathsf{inp}+1, N+1]$, parse $\mathsf{R}_{(i,\mathrm{label})} = \{\mathsf{label}^i_{k,b}\}$.
3. For $\beta \in \{0,1\}$ set $D_{i_{\mathsf{G}}+\beta}$ as

$$D_{i_{\mathsf{G}}+\beta}[i] := \begin{cases} x_i \oplus r_i & i \le \mathsf{inp} \\ E_i \oplus r_i & \mathsf{inp}+1 \le i < i_{\mathsf{G}}+\beta \\ 0 & \text{otherwise,} \end{cases}$$

where E_i is the bit assigned to the output wire of the gate indexed by i when we evaluate C on the input x.

4. For $\beta \in \{0,1\}$, compute $(d_{i_{\mathsf{G}}+\beta}, \widehat{D}_{i_{\mathsf{G}}+\beta}) := \mathsf{Hash}(\mathsf{crs}, D_{i_{\mathsf{G}}+\beta})$.
5. Compute $e \leftarrow \mathsf{SimWrite}(\mathsf{crs}, D_{i_{\mathsf{G}}}, i_{\mathsf{G}}, D_{i_{\mathsf{G}}+1}[i_{\mathsf{G}}], \{\mathsf{label}^{i_{\mathsf{G}}+1}_{k,d_{i_{\mathsf{G}}+1}[k]}\}_{k\in[|d|]})$.
6. Compute $\mathsf{gout} \leftarrow \mathsf{Sim}(\mathsf{crs}, D_{i_{\mathsf{G}}}, A_{\mathsf{G}}, \mathsf{Sim}(\mathsf{crs}, D_{i_{\mathsf{G}}}, B_{\mathsf{G}}, (D_{i_{\mathsf{G}}+1}[i_{\mathsf{G}}], e)))$.
7. Compute and output

$$\widetilde{\mathsf{G}} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{gout}, \{\mathsf{label}^{i_{\mathsf{G}}}_{k,d_{i_{\mathsf{G}}}[k]}\}_{k\in[|d|]}).$$

$\mathsf{SimInpConf}(\mathsf{st}, x, y, \mathsf{conf})$: On input $\mathsf{st} = (\mathsf{pp}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{st}}})$, $x \in \{0,1\}^{\mathsf{inp}}$, $y \in \{0,1\}^{\mathsf{out}}$, and $\mathsf{conf} = \{\mathsf{mode}(\mathsf{G})\}_{\mathsf{G}\in\mathsf{Gates}}$, do the following:

1. Parse $\mathsf{pp} = (\mathsf{crs}, d_{\mathsf{off}})$ and $\mathsf{R}_{(\mathsf{inp}+1,\mathrm{label})} = \{\mathsf{label}^{\mathsf{inp}+1}_{k,b}\}$.
2. For $i \in [\mathsf{inp}] \cup [N-\mathsf{out}+1, N]$, set r_i to be the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.
3. For $\mathsf{G} \in \mathsf{Gates}$ such that $i_{\mathsf{G}} \in [N-\mathsf{out}+1, N]$, do either of the following:

(a) If $\mathsf{mode}(\mathsf{G}) = \mathsf{Black}$, then set $r'_{i_\mathsf{G}} := r_{i_\mathsf{G}} \oplus y_{i_\mathsf{G}-(N-\mathsf{out})}$.
(b) Else, $r'_{i_\mathsf{G}} := r_{i_\mathsf{G}}$.

4. Compute $d := \mathsf{HashOn}(\mathsf{crs}, r_1 \oplus x_1 \| \cdots \| r_{\mathsf{inp}} \oplus x_{\mathsf{inp}}, d_{\mathsf{off}})$.
5. Output $\widetilde{x} := \left(\{\mathsf{label}^{\mathsf{inp}+1}_{k,d[k]}\}_{k\in[|d|]}, \{r_i \oplus x_i\}_{i\in[\mathsf{inp}]}, \{r'_i\}_{i\in[N-\mathsf{out}+1,N]}\right)$.

First, we prove that the above simulator satisfies the requirements of Definition 3.4.

1. It is easy to see that $\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{Black}}(\mathsf{G})})$ runs in time $\mathrm{poly}(\lambda)$ independently of the size of C from the construction.
2. It is easy to see that $\mathsf{SimInpConf}(\mathsf{st}, x, y, \mathsf{White}^{N-n})$ does not use y at all and works in exactly the same way as $\mathsf{GInp}(\mathsf{st}, x)$.
3. The requirements for the case of $\mathsf{conf} = \mathsf{Black}^{N-n}$ can be seen as follows:
   (a) Clearly, $S_{\mathsf{Black}}(\mathsf{G}) := \{(i_\mathsf{G}, \mathrm{pad}), (i_\mathsf{G}, \mathrm{label}), (i_\mathsf{G}+1, \mathrm{label})\}$ can be computed from i_G and $\mathsf{top}(C)$. It is also easy to see that $\mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{Black}}(\mathsf{G})})$ only needs $\mathsf{top}(\mathsf{G}) = \{i_\mathsf{G}, A_\mathsf{G}, B_\mathsf{G}\}$ and does not depend on g_G since it only garbles a step circuit with $\mathsf{flag} = 1$, which does not need g_G.
   (b) We define SimInp as follows:
   $\mathsf{SimInp}(\mathsf{st}, y)$: On input $\mathsf{st} = (\mathsf{pp}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{st}}})$ and $y \in \{0,1\}^{\mathsf{out}}$, do the following:
      i. Parse $\mathsf{pp} = (\mathsf{crs}, d_{\mathsf{off}})$ and $\mathsf{R}_{(\mathsf{inp}+1,\mathrm{label})} = \{\mathsf{label}^{\mathsf{inp}+1}_{k,b}\}$.
      ii. For $i \in [\mathsf{inp}] \cup [N-\mathsf{out}+1, N]$, set r_i to be the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.
      iii. For $i \in [N-\mathsf{out}+1, N]$, set $r'_i := r_i \oplus y_{i-(N-\mathsf{out})}$.
      iv. Compute $d := \mathsf{HashOn}(\mathsf{crs}, r_1 \| \cdots \| r_{\mathsf{inp}}, d_{\mathsf{off}})$.
      v. Output $\widetilde{x} := \left(\{\mathsf{label}^{\mathsf{inp}+1}_{k,d[k]}\}_{k\in[|d|]}, \{r_i\}_{i\in[\mathsf{inp}]}, \{r'_i\}_{i\in[N-\mathsf{out}+1,N]}\right)$.
   The only difference of $\mathsf{SimInp}(\mathsf{st}, y)$ from $\mathsf{SimInpConf}(\mathsf{st}, x, y, \mathsf{Black}^{N-\mathsf{out}})$ is that it uses r_i instead of $x_i \oplus r_i$ for $i \in [\mathsf{inp}]$. It is easy to see that $\{(i, \mathrm{pad})\}_{i\in[\mathsf{inp}]} \cap (\cup_{\mathsf{G}\in\mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G})) = \emptyset$. Therefore, both r_i and $x_i \oplus r_i$ are uniformly distributed conditioned on any $\{\mathsf{R}_{(i,\mathrm{type})}\}_{((i,\mathrm{type})\in\cup_{\mathsf{G}\in\mathsf{Gates}} S_{\mathsf{Black}}(\mathsf{G}))}$. This means that the requirement is satisfied.

Next, we prove that the above simulator satisfies the requirements of Definition 3.5. For proving this, we rely on the following lemma proven in [GS18].

Lemma A.2. *There is a sequence* $(\mathsf{conf}_1, \ldots, \mathsf{conf}_T) \in \{\mathsf{White}, \mathsf{Gray}, \mathsf{Black}\}^T$ *for* $T = \mathrm{poly}(|C|)$ *that satisfies the following where we use the notation* $\mathsf{conf}_j = \{\mathsf{mode}_j(\mathsf{G})\}_{\mathsf{G}\in\mathsf{Gates}}$ *and* $I_j = \{\mathsf{G} \in \mathsf{Gates} : \mathsf{mode}_j(\mathsf{G}) = \mathsf{Gray}\}$ *for* $j \in [T]$*, and* $\mathsf{Pre}(\mathsf{G})$ *denotes* $\mathsf{G}' \in \mathsf{Gates}$ *such that* $i_{\mathsf{G}'} = i_\mathsf{G} - 1$*:*[22]

1. $\mathsf{conf}_1 = \mathsf{White}^{|\mathsf{Gates}|}$ *and* $\mathsf{conf}_T = \mathsf{Black}^{|\mathsf{Gates}|}$.
2. $|I_j| \leq M$ *for all* $j \in [T]$ *where* $M = O(\log |C|)$.
3. *For all* $j \in [T-1]$, conf_j *and* conf_{j+1} *satisfy either of the following:*
   Rule A*: There is* $\mathsf{G}^*_j \in \mathsf{Gates}$ *such that:*
   - $\mathsf{conf}_j(\mathsf{G}) = \mathsf{conf}_{j+1}(\mathsf{G})$ *for all* $\mathsf{G} \neq \mathsf{G}^*_j$,
   - $i_{\mathsf{G}^*_j} = \mathsf{inp} + 1$ *or* $\mathsf{mode}_j(\mathsf{Pre}(\mathsf{G}^*_j)) = \mathsf{Gray}$,
   - $\mathsf{mode}_j(\mathsf{G}^*_j) = \mathsf{White}$ *and* $\mathsf{mode}_{j+1}(\mathsf{G}^*_j) = \mathsf{Gray}$.

   Rule $\mathbf{A}^{-1}$*: This is the same as Rule A except that the roles of* conf_j *and* conf_{j+1} *are reversed. Formally, there is* $\mathsf{G}^*_j \in \mathsf{Gates}$ *such that:*
   - $\mathsf{conf}_j(\mathsf{G}) = \mathsf{conf}_{j+1}(\mathsf{G})$ *for all* $\mathsf{G} \neq \mathsf{G}^*_j$,
   - $i_{\mathsf{G}^*_j} = \mathsf{inp} + 1$ *or* $\mathsf{mode}_j(\mathsf{Pre}(\mathsf{G}^*_j)) = \mathsf{Gray}$,
   - $\mathsf{mode}_j(\mathsf{G}^*_j) = \mathsf{Gray}$ *and* $\mathsf{mode}_{j+1}(\mathsf{G}^*_j) = \mathsf{White}$.

   Rule B*: There is* $\mathsf{G}^*_j \in \mathsf{Gates}$ *such that:*
   - $\mathsf{conf}_j(\mathsf{G}) = \mathsf{conf}_{j+1}(\mathsf{G})$ *for all* $\mathsf{G} \neq \mathsf{G}^*_j$,
   - $i_{\mathsf{G}^*_j} = \mathsf{inp} + 1$ *or* $\mathsf{mode}_j(\mathsf{Pre}(\mathsf{G}^*_j)) = \mathsf{Gray}$,
   - *For* $\mathsf{G} \in \mathsf{Gates}$, $\mathsf{mode}_j(\mathsf{G}) = \mathsf{Black}$ *if and only if* $i_\mathsf{G} > i_{\mathsf{G}^*_j}$.

[22] $\mathsf{Pre}(\mathsf{G})$ is undefined when $i_\mathsf{G} = \mathsf{inp} + 1$.

- $\mathsf{mode}_j(\mathsf{G}^*_j) = \mathsf{Gray}$ *and* $\mathsf{mode}_{j+1}(\mathsf{G}^*_j) = \mathsf{Black}$.

Items 1 and 2 of Lemma A.2 directly imply Items 1 and 2 of Definition 3.5. Moreover, it is also easy to see that Item 3 of Lemma A.2 implies Item 3 of Definition 3.5 since there is only one gate in different modes in either rule. What is left is to prove that the requirement of Item 4 of Definition 3.5 is satisfied assuming that neighboring configurations satisfy either of Rule A, Rule A^{-1} or Rule B. We prove this below.

Lemma A.3. *If* conf_j *and* conf_{j+1} *satisfy Rule A, Rule* A^{-1} *or Rule B, then for every stateful* PPT *adversary* A *and polynomial* $\mathsf{size} = \mathsf{size}(\lambda)$,

$$\left|\Pr[\mathsf{Exp}^{j,0}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}\left(1^\lambda, 1^{\mathsf{size}}\right) = 1] - \Pr[\mathsf{Exp}^{j,1}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}\left(1^\lambda, 1^{\mathsf{size}}\right) = 1]\right| = \mathrm{negl}(\lambda)$$

where $\mathsf{Exp}^{j,b}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$ *is as defined in Item 4 of Definition 3.5.*

Proof. We prove this for each case where we denote by G^*, A^*, B^*, i^*, $S_j(\mathsf{G})$ to mean G^*_j, $A_{\mathsf{G}^*_j}$, $B_{\mathsf{G}^*_j}$, $i_{\mathsf{G}^*_j}$, and $S_{\mathsf{mode}_j(\mathsf{G})}(\mathsf{G})$, respectively for notational simplicity.

Rule A Suppose that conf_j and conf_{j+1} satisfy Rule A. We consider the following sequence of games. Let $\mathcal{E}_{\mathrm{xx}}$ be the event that $\mathsf{Game}_{\mathrm{xx}}$ outputs 1.

$\mathbf{Game}_0$: This game works similarly to $\mathsf{Exp}^{j,0}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$. Noting that $I_j \cup I_{j+1} = I_{j+1}$ and $I_j \cup I_{j+1} \setminus \{\mathsf{G}^*\} = I_j$ by Rule A, it works as follows:

1. $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, $d_{\mathsf{off}} := \mathsf{HashOff}(\mathsf{crs}, 0^{N-\mathsf{inp}})$, $\mathsf{pp} := (\mathsf{crs}, d_{\mathsf{off}})$.
2. $\left(C, x, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in V_j}\right) \leftarrow \mathsf{A}(1^\lambda, 1^{\mathsf{size}}, \mathsf{pp})$ where C is a circuit of size at most size and
$$V_j := \bigcup_{\mathsf{G}\in\mathsf{Gates}\setminus I_{j+1}} S_j(\mathsf{G}).$$
3. $\mathsf{R}_{(i,\mathrm{type})} \leftarrow \{0,1\}^\ell$ for $(i,\mathrm{type}) \in [N_{\mathsf{rand}}] \setminus V_j$. We use the following notations as in the description of the scheme.
   - For (i, label), we parse $\mathsf{R}_{(i,\mathrm{label})}$ as $\{\mathsf{label}^i_{k,b}\}$.
   - For (i, pad), we denote by r_i to mean the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.
4. For all $\mathsf{G} \in I_j$, $\widetilde{\mathsf{G}} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{i,\mathrm{type}}\}_{(i,\mathrm{type})\in[N_{\mathsf{rand}}]}, C, x)$.
5. $\widetilde{\mathsf{G}^*} \leftarrow \mathsf{SimGate}_{\mathsf{White}}(\mathsf{pp}, \mathsf{G}^*, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{White}}(\mathsf{G}^*)})$. Specifically,
$$\widetilde{\mathsf{G}^*} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{SC}[\mathsf{crs}, (r_{A^*}, r_{B^*}, r_{i^*}), \mathsf{G}^*, \{\mathsf{label}^{i^*+1}_{k,b}\}, 0], \{\mathsf{label}^{i^*}_{k,b}\}).$$
6. $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{st}}})$.
7. $\widetilde{x} \leftarrow \mathsf{SimInpConf}(\mathsf{st}, x, C(x), \mathsf{conf}_j)$.
8. $b' \leftarrow \mathcal{A}(\{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in I_{j+1}}, \widetilde{x})$.
9. The game outputs b'.

$\mathbf{Game}_1$: This game is identical to the previous game except that Item 5 is replaced with the following:

5(a). $\mathsf{gout}^* := \mathsf{SC}[\mathsf{crs}, (r_{A^*}, r_{B^*}, r_{i^*}), \mathsf{G}^*, \{\mathsf{label}^{i^*+1}_{k,b}\}, 0](d_{i^*})$ where d_{i^*} is as in the description of $\mathsf{SimGate}_{\mathsf{Gray}}$.

5(b). $\widetilde{\mathsf{G}} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{gout}^*, \{\mathsf{label}^{i^*}_{k,d_{i^*}[k]}\}_{k\in[|d|]})$.

By Rule A, G^* is the "first gate" (i.e., $i^* = \mathsf{inp}+1$) or the previous gate is in the Gray mode (i.e., $\mathsf{mode}_j(\mathsf{Pre}(\mathsf{G}^*)) = \mathsf{Gray}$). Therefore, we have $(i^*, \mathrm{label}) \notin V_j$. Moreover, $\{\mathsf{label}^{i^*}_{k,1-d_{i^*}[k]}\}_{k\in[|d|]}$ is not used in the game at all. Therefore, by a straightforward reduction to the selective security of GC,

$$|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \le \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$: This game is identical to the previous game except that gout^* is generated as follows:

$$\mathsf{gout}^* \leftarrow \mathsf{Sim}(\mathsf{crs}, D_{i^*}, A^*, f_{D_{i^*}[A^*]})$$

where D_{i^*} is as in the description of $\mathsf{SimGate}_{\mathsf{Gray}}$ and $f_{D_{i^*}[A^*]}$ is generated as in the description of SC, i.e.,

- $e_b \leftarrow \mathsf{SendWrite}(\mathsf{crs}, d_{i^*}, i^*, b, \{\mathsf{label}_{k,0}^{i^*+1}, \mathsf{label}_{k,1}^{i^*+1}\}_{k\in[|d|]})$ for $b \in \{0,1\}$.
- $f_{D_{i^*}[A^*]} := \mathsf{Send}(\mathsf{crs}, d_{i^*}, B^*, (\gamma(D_{i^*}[A^*], 0), e_{\gamma(D_{i^*}[A^*],0)}), (\gamma(D_{i^*}[A^*], 1), e_{\gamma(D_{i^*}[A^*],1)}))$ where γ is defined as in the $\mathsf{flag} = 0$ branch in the description of SC.

By a straightforward reduction to the sender privacy of Π_{LOT},

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$: This game is identical to the previous game except that $f_{D_{i^*}[A^*]}$ is generated as follows:

$$f_{D_{i^*}[A^*]} := \mathsf{Sim}(\mathsf{crs}, D_{i^*}, B^*, (D_{i^*+1}[i^*], e_{D_{i^*+1}[i^*]})).$$

Noting that $\gamma(D_{i^*}[A^*], D_{i^*}[B^*]) = D_{i^*+1}[i^*]$, by a straightforward reduction to the sender privacy of Π_{LOT},

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_4$: This game is identical to the previous game except that $e_{D_{i^*+1}[i^*]}$ is generated as follows:

$$e_{D_{i^*+1}[i^*]} \leftarrow \mathsf{SimWrite}(\mathsf{crs}, D_{i^*}, i^*, D_{i^*+1}[i^*], \{\mathsf{label}_{k,d_{i^*+1}[k]}^{i^*+1}\}_{k\in[|d|]})$$

By a straightforward reduction to the sender privacy with regard to write of Π_{LOT},

$$|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda).$$

Moreover, one can see that the way of generating $\widetilde{\mathsf{G}^*}$ in $\mathbf{Game}_4$ is identical to that by $\mathsf{SimGate}_{\mathsf{Gray}}$, and thus $\mathbf{Game}_4$ is identical to $\mathsf{Exp}^{j,1}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$. This completes the proof of Lemma A.3 for the case of Rule A.

Rule $\mathbf{A}^{-1}$ For this case, we can just consider similar game hops to those for the case of Rule A in the reversed order.

Rule B Suppose that conf_j and conf_{j+1} satisfy Rule B. The proof of this case is very similar to the case of Rule A except for the final step. We consider the following sequence of games. Let $\mathcal{E}_{\mathrm{xx}}$ be the event that $\mathsf{Game}_{\mathrm{xx}}$ outputs 1.

$\mathbf{Game}_0$: This game works similarly to $\mathsf{Exp}^{j,1}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$. Remark that we start from $\mathsf{Exp}^{j,1}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$ rather than $\mathsf{Exp}^{j,0}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$ differently from the case of Rule A. Noting that $I_j \cup I_{j+1} = I_j$ and $I_j \cup I_{j+1} \setminus \{\mathsf{G}^*\} = I_{j+1}$ by Rule B, it works as follows:

1. $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, $d_{\mathsf{off}} := \mathsf{HashOff}(\mathsf{crs}, 0^{N-\mathsf{inp}})$, $\mathsf{pp} := (\mathsf{crs}, d_{\mathsf{off}})$.
2. $\left(C, x, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in V_j}\right) \leftarrow \mathsf{A}(1^\lambda, 1^{\mathsf{size}}, \mathsf{pp})$ where C is a circuit of size at most size and
$$V_j := \bigcup_{\mathsf{G}\in\mathsf{Gates}\setminus I_j} S_j(\mathsf{G}).$$
3. $\mathsf{R}_{(i,\mathrm{type})} \leftarrow \{0,1\}^\ell$ for $(i,\mathrm{type}) \in [N_{\mathsf{rand}}] \setminus V_j$. We use the following notations as in the description of the scheme.
   - For (i, label), we parse $\mathsf{R}_{(i,\mathrm{label})}$ as $\{\mathsf{label}_{k,b}^i\}$.
   - For (i, pad), we denote by r_i to mean the first bit of $\mathsf{R}_{(i,\mathrm{pad})}$.
4. For all $\mathsf{G} \in I_{j+1}$, $\widetilde{\mathsf{G}} \leftarrow \mathsf{SimGate}_{\mathsf{Gray}}(\mathsf{pp}, \mathsf{G}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in[N_{\mathsf{rand}}]}, C, x)$.
5. $\widetilde{\mathsf{G}^*} \leftarrow \mathsf{SimGate}_{\mathsf{Black}}(\mathsf{pp}, \mathsf{G}^*, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{Black}}(\mathsf{G}^*)})$. Specifically,
$$\widetilde{\mathsf{G}^*} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{SC}[\mathsf{crs}, (0,0,r_{i^*}), \mathsf{G}^*, \{\mathsf{label}_{k,b}^{i^*+1}\}, 1], \{\mathsf{label}_{k,b}^{i^*}\}).$$
6. $\mathsf{st} := (\mathsf{pp}, \{\mathsf{R}_{(i,\mathrm{type})}\}_{(i,\mathrm{type})\in S_{\mathsf{st}}})$.
7. $\widetilde{x} \leftarrow \mathsf{SimInpConf}(\mathsf{st}, x, C(x), \mathsf{conf}_{j+1})$.
8. $b' \leftarrow \mathcal{A}(\{\widetilde{\mathsf{G}}\}_{\mathsf{G}\in I_j}, \widetilde{x})$.
9. The game outputs b'.

$\mathbf{Game}_1$: This game is identical to the previous game except that Item 5 is replaced with the following:

5(a). $\mathsf{gout}^* := \mathsf{SC}[\mathsf{crs}, (0,0,r_{i^*}), \mathsf{G}^*, \{\mathsf{label}_{k,b}^{i^*+1}\}, 1](d_{i^*})$ where d_{i^*} is as in the description of $\mathsf{SimGate}_{\mathsf{Gray}}$.

5(b). $\widetilde{\mathsf{G}} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{gout}^*, \{\mathsf{label}^{i^*}_{k,d_{i^*}[k]}\}_{k\in[|d|]})$.
By Rule B, G^* is the "first gate" (i.e., $i^* = \mathsf{inp}+1$) or the previous gate is in the Gray mode (i.e., $\mathsf{mode}_j(\mathsf{Pre}(\mathsf{G}^*)) = \mathsf{Gray}$). Therefore, we have $(i^*, \mathrm{label}) \notin V_j$. Moreover. $\{\mathsf{label}^{i^*}_{k,1-d_{i^*}[k]}\}_{k\in[|d|]}$ is not used in the game at all. Therefore, by a straightforward reduction to the selective security of GC,

$$|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \le \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$: This game is identical to the previous game except that gout^* is generated as follows:

$$\mathsf{gout}^* \leftarrow \mathsf{Sim}(\mathsf{crs}, D_{i^*}, A^*, f_{D_{i^*}[A^*]})$$

where D_{i^*} is as in the description of $\mathsf{SimGate}_{\mathsf{Gray}}$ and $f_{D_{i^*}[A^*]}$ is generated as in the description of SC, i.e.,

- $e_{r^*_i} \leftarrow \mathsf{SendWrite}(\mathsf{crs}, d_{i^*}, i^*, r_{i^*}, \{\mathsf{label}^{i^*+1}_{k,0}, \mathsf{label}^{i^*+1}_{k,1}\}_{k\in[|d|]})$.
- $f_{D_{i^*}[A^*]} := \mathsf{Send}(\mathsf{crs}, d_{i^*}, B^*, (r_{i^*}, e_{r_{i^*}}), (r_{i^*}, e_{r_{i^*}}))$.

By a straightforward reduction to the sender privacy of Π_{LOT},

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \le \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$: This game is identical to the previous game except that $f_{D_{i^*}[A^*]}$ is generated as follows:

$$f_{D_{i^*}[A^*]} := \mathsf{Sim}(\mathsf{crs}, D_{i^*}, B^*, (r_{i^*}, e_{r_{i^*}})).$$

By a straightforward reduction to the sender privacy of Π_{LOT},

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \le \mathrm{negl}(\lambda).$$

$\mathbf{Game}_4$: This game is identical to the previous game except that $e_{r_{i^*}}$ is generated as follows:

$$e_{r_{i^*}} \leftarrow \mathsf{SimWrite}(\mathsf{crs}, D_{i^*}, i^*, r_{i^*}, \{\mathsf{label}^{i^*+1}_{k,d'_{i^*+1}[k]}\}_{k\in[|d|]})$$

where d'_{i^*+1} is the digest corresponding to the database D'_{i^*+1} defined as

$$D'_{i^*+1}[i] := \begin{cases} D_{i^*+1}[i] & i \neq i^*, \\ r_{i^*} & i = i^* \end{cases}$$

By a straightforward reduction to the sender privacy with regard to write of Π_{LOT},

$$|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \le \mathrm{negl}(\lambda).$$

One can see that $\mathbf{Game}_4$ is identical to $\mathsf{Exp}^{j,0}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$ except for the following differences:

1. When generating $\widetilde{\mathsf{G}^*}$, D'_{i^*+1} is used instead of D_{i^*+1}.
2. When generating $\widetilde{x}$, r'_{i^*} is set to be $r_{i^*} \oplus C_{i^*-(N-\mathsf{out})}(x)$ instead of r_{i^*} where $C_{i^*-(N-\mathsf{out})}(x)$ denotes the $(i^* - (N - \mathsf{out}))$-th bit of $C(x)$ if $i^* \ge N - \mathsf{out} + 1$.

We show that they make no difference from the view of A by considering the following two cases:

If $i^* < N - \mathsf{out} + 1$: In this case, the second difference is irrelevant. Moreover, r_{i^*} is only used for setting $D'_{i^*+1}[i^*]$, and thus the distribution is identical even if we replace it with $E_{i^*} \oplus r_{i^*}$ because all the succeeding gates are in Black mode, i.e., $\mathsf{mode}_j(\mathsf{G}) = \mathsf{Black}$ for all G such that $i_{\mathsf{G}} > i^*$. This means that the adversary cannot notice the difference even if we replace D'_{i^*+1} with D_{i^*+1} in the generation of $\widetilde{\mathsf{G}^*}$.

If $i^* \ge N - \mathsf{out} + 1$: The joint distribution of $(D'_{i^*+1}[i^*] = r_{i^*}, r'_{i^*} = r_{i^*} \oplus C_{i^*-(N-\mathsf{out})}(x))$ is equal to that of $(D_{i^*+1}[i^*] = C_{i^*-(N-\mathsf{out})}(x) \oplus r_{i^*}, r_{i^*})$. Therefore, the adversary cannot notice the difference even if we replace D'_{i^*+1} with D_{i^*+1} in the generation of $\widetilde{\mathsf{G}^*}$ and replace $C_{i^*-(N-\mathsf{out})}(x) \oplus r_{i^*}$ with r_{i^*} in the garbled input.

Based on the above observations, in either case, $\mathbf{Game}_4$ is identical to $\mathsf{Exp}^{j,0}_{\mathsf{GC},\mathsf{Sim}_{\mathsf{GC}},\mathsf{A}}(1^\lambda, 1^{\mathsf{size}})$ from the view of A. This completes the proof of Lemma A.3 for the case of Rule B.

Circuit $\mathsf{Circuit}[C, i, \mathsf{K}, \{r^{(q)}, y_i^{(q)}\}_{q\in[Q]}, \mathsf{flag}]$

Input: an input $x \in \{0,1\}^{\mathsf{inp}}$, key-tag $r \in \{0,1\}^{\lambda}$, and tuple of SKE ciphertexts $\{\mathsf{SKE.ct}_i\}_{i\in[K]}$.

Hardwired value: a circuit C, ciphertext-tag $i \in [K]\cup\{\bot\}$, SKE key K, tuple $\{r^{(q)}, y^{(q)}\}_{q\in[Q]}$ of key-tags and outputs, and flag $\mathsf{flag} \in \{0,1\}$.

1. If $\mathsf{flag} = 0$, output $C(x)$.
2. Otherwise, do the following:
   (a) If $r = r^{(q)}$ for some $q \in [Q]$, let q^* be the smallest such q and output $y_i^{(q^*)}$.
   (b) Otherwise, output $\mathsf{SKE.Dec}(\mathsf{K}, \mathsf{SKE.ct}_i)$.

Fig. 13 The description of Circuit

B Multi-Ciphertext AD-SIM-secure FE

B.1 Construction

In this section, we prove Lemma 4.1. That is, assuming the existence of IBE, we construct a K-CT AD-SIM-secure CPFE scheme K-CPFE with dynamic bounded collusion (as defined in Definition 4.2) for the circuit class $\mathcal{C}_{\mathsf{inp,out,size}}$ for any polynomials $K = K(\lambda)$, $\mathsf{inp} = \mathsf{inp}(\lambda)$, $\mathsf{out} = \mathsf{out}(\lambda)$ and $\mathsf{size} = \mathsf{size}(\lambda)$ where $\mathcal{C}_{\mathsf{inp,out,size}}$ is the class of circuits consisting of all polynomial-size circuits of input-length inp, output-length out, and size size.

Ingredients. We now describe the underlying building blocks used to obtain K-CPFE:

1. A SKE scheme $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ with the message space $\mathcal{M} = \{0,1\}^{\mathsf{out}}$ and ciphertext space $\mathcal{C}$ that satisfies the pseudorandom ciphertext property.
2. A (single-ciphertext) AD-SIM-secure CPFE scheme with dynamic bounded collusion (as defined in Definition 2.3) denoted by
$$\text{1-CPFE} = (\text{1-CPFE.Setup}, \text{1-CPFE.Enc}, \text{1-CPFE.KeyGen}, \text{1-CPFE.Dec})$$
for bounded polynomial-size circuits. Namely, 1-CPFE supports a circuit class $\mathcal{C}_{\mathsf{inp,size}'}$ consisting of circuits with input length inp and size at most size', where size' is the maximum size of the circuit Circuit defined in Figure 13. Formally, 1-CPFE is FE with $\text{1-CPFE.prm} = (1^{\mathsf{inp}}, 1^{\mathsf{size}'})$, $\mathcal{X}_{\text{1-CPFE.prm}} = \mathcal{C}_{\mathsf{inp,size}'}$, $\mathcal{Y}_{\text{1-CPFE.prm}} = \{0,1\}^{\mathsf{inp}}$, and $R_{\mathsf{prm}'} : \mathcal{X}_{\text{1-CPFE.prm}} \times \mathcal{Y}_{\text{1-CPFE.prm}} \to \{0,1\}^*$ with $R(C,x) = C(x)$ for all $C \in \mathcal{C}_{\mathsf{inp,size}'}$ and $x \in \{0,1\}^{\mathsf{inp}}$. Such a CPFE scheme is constructed in [AMVY21, Section 3.3] from IBE.

Construction. The description of K-CPFE is given below.

$\mathsf{Setup}(1^{\lambda}, \mathsf{prm})$: On input the security parameter λ and the parameter $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{out}}, 1^{\mathsf{size}})$, do the following:
1. Set $\mathsf{prm}' := (1^{\mathsf{inp}}, 1^{\mathsf{size}'})$ as described above.
2. Run $(\text{1-CPFE.mpk}, \text{1-CPFE.msk}) \leftarrow \text{1-CPFE.Setup}(1^{\lambda}, \mathsf{prm}')$.
3. Output $(\mathsf{mpk}, \mathsf{msk}) := (\text{1-CPFE.mpk}, \text{1-CPFE.msk})$.

$\mathsf{Enc}(\mathsf{mpk}, C, 1^Q)$: On input the master public key $\mathsf{mpk} = \text{1-CPFE.mpk}$, a circuit $C \in \mathcal{C}_{\mathsf{inp,out,size}}$, and the query bound $1 \le Q \le 2^{\lambda}$ in unary form, compute
$$\text{1-CPFE.ct} \leftarrow \text{1-CPFE.Enc}(\text{1-CPFE.mpk}, \mathsf{Circuit}[C, \bot, \bot, \{\bot, \bot\}_{q\in[Q]}, 0], 1^Q)$$
where $\mathsf{Circuit}[C, \bot, \bot, \{\bot, \bot\}_{i\in[K]}, 0]$ is the circuit defined in Figure 13 and output $\mathsf{ct} := \text{1-CPFE.ct}$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$: On input the master secret key $\mathsf{msk} = \text{1-CPFE.msk}$ and an input $x \in \{0,1\}^{\mathsf{inp}}$, do the following:
1. Choose $r \leftarrow \{0,1\}^{\lambda}$.
2. For $i \in [K]$, choose $\mathsf{SKE.ct}_i \leftarrow \mathcal{C}$.
3. Run $\text{1-CPFE.sk} \leftarrow \text{1-CPFE.KeyGen}(\text{1-CPFE.msk}, (x, r, \{\mathsf{SKE.ct}_i\}_{i\in[K]}))$.
4. Output $\mathsf{sk} := (r, \{\mathsf{SKE.ct}_i\}_{i\in[K]}, \text{1-CPFE.sk})$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, 1^Q)$: On input a ciphertext $\mathsf{ct} = \text{1-CPFE.ct}$ and a secret key $\mathsf{sk} = (r, \{\mathsf{SKE.ct}_i\}_{i\in[K]}, \text{1-CPFE.sk})$, output $\text{1-CPFE.Dec}(\text{1-CPFE.ct}, \text{1-CPFE.sk}, 1^Q)$.

The correctness of the above scheme is clear.

Security The following theorem asserts the security of K-CPFE.

Theorem B.1. *If* SKE *satisfies the pseudorandom ciphertext property and* 1-CPFE *is* AD-SIM*-secure against dynamic bounded collusion, then* K-CPFE *is* K*-CT* AD-SIM*-secure against dynamic bounded collusion.*

Proof. We first construct a simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ for K-CPFE as follows:

$\mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}, 1^{\mathsf{size}}, 1^Q)$: Do the following:

1. Parse $\mathsf{mpk} = \text{1-CPFE.mpk}$, $\mathcal{V} = \left\{\left(\{y_i^{(q)}\}_{i\in[\hat{K}]}, x^{(q)}, \mathsf{sk}^{(q)}\right)\right\}_{q\in[Q_1]}$ for $1 \le \hat{K} \le K$ and $1 \le Q_1 \le Q$, and $\mathsf{sk}^{(q)} = (r^{(q)}, \{\mathsf{SKE.ct}_i^{(q)}\}_{i\in[K]}, \text{1-CPFE.sk}^{(q)})$ for $q \in [Q_1]$.
2. For $i \in [\hat{K}]$, run $\mathsf{K}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
3. For $i \in [\hat{K}]$, run
$$\text{1-CPFE.ct}_i \leftarrow \text{1-CPFE.Enc}(\text{1-CPFE.mpk}, \mathsf{Circuit}[\perp, i, \mathsf{K}_i, \{r^{(q)}, y_i^{(q)}\}_{q\in[Q]}, 1], 1^Q)$$
where $r^{(q)}$ and $y_i^{(q)}$ are set to be $\perp$ for $Q_1 + 1 \le q \le Q$ and $\mathsf{Circuit}[\perp, i, \mathsf{K}_i, \{r^{(q)}, y_i^{(q)}\}_{q\in[Q]}, 1]$ is the circuit defined in Figure 13.
4. Output $\{\mathsf{ct}_i\}_{i\in[\hat{K}]} := \{\text{1-CPFE.ct}_i\}_{i\in[\hat{K}]}$ and $\mathsf{st} := \{\mathsf{K}_i\}_{i\in[\hat{K}]}$.

$\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, \{y_i^{(q)}\}_{i\in[\hat{K}]}, x^{(q)})$: Do the following:

1. Parse $\mathsf{st} = \{\mathsf{K}_i\}_{i\in[\hat{K}]}$ and $\mathsf{msk} = \text{1-CPFE.msk}$.
2. $r^{(q)} \leftarrow \{0,1\}^\lambda$.
3. For $i \in [\hat{K}]$, compute $\mathsf{SKE.ct}_i^{(q)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_i, y_i^{(q)})$.
4. For $i \in [\hat{K}+1, K]$, choose $\mathsf{SKE.ct}_i^{(q)} \leftarrow \mathcal{C}$.
5. Run $\text{1-CPFE.sk}^{(q)} \leftarrow \text{1-CPFE.KeyGen}(\text{1-CPFE.msk}, (x^{(q)}, r^{(q)}, \{\mathsf{SKE.ct}_i^{(q)}\}_{i\in[K]}))$.
6. Output $\mathsf{sk}^{(q)} := (r^{(q)}, \{\mathsf{SKE.ct}_i\}_{i\in[K]}, \text{1-CPFE.sk}^{(q)})$.

We consider the following sequence of games. Let $\mathcal{E}_{\text{xx}}$ be the event that $\mathsf{Game}_{\text{xx}}$ outputs 1.

$\mathbf{Game}_0$: This is the experiment $\mathsf{Exp}^{\mathsf{ideal}}_{K\text{-CPFE},\mathsf{A},\mathsf{Sim}}(1^\lambda)$ defined in Definition 4.2. Specifically, the game works as follows:

1. Run $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{out}}, 1^{\mathsf{size}}) \leftarrow \mathsf{A}(1^\lambda)$.
2. Run $(\text{1-CPFE.mpk}, \text{1-CPFE.msk}) \leftarrow \text{1-CPFE.Setup}(1^\lambda, \mathsf{prm}')$.
3. Run $(\{y_i\}_{i\in[K'-1]}, C, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\text{1-CPFE.mpk})$ where $1 \le K' \le K$.
   - Let $(x^{(1)}, \ldots, x^{(Q_1)})$ be A's oracle queries.
   - Let $\mathsf{sk}^{(q)} = (r^{(q)}, \{\mathsf{SKE.ct}_i^{(q)}\}_{i\in[K]}, \text{1-CPFE.sk}^{(q)})$ be the oracle reply to $x^{(q)}$.
   - Let $y_{K'}^{(q)} := C(x^{(q)})$.
   - Let $\mathcal{V}' := \left\{\left(\{y_i^{(q)}\}_{i\in[K']}, x^{(q)}, \mathsf{sk}^{(q)}\right)\right\}_{q\in[Q_1]}$.
4. Run $(\{\mathsf{ct}_i\}_{i\in[K']}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}', 1^{\mathsf{size}}, 1^Q)$. Specifically, do the following:
   (a) For $i \in [K']$, run $\mathsf{K}_i \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (b) For $i \in [K']$, run
   $$\text{1-CPFE.ct}_i \leftarrow \text{1-CPFE.Enc}(\text{1-CPFE.mpk}, \mathsf{Circuit}[\perp, i, \mathsf{K}_i, \{r^{(q)}, y_i^{(q)}\}_{q\in[Q]}, 1], 1^Q)$$
   where $r^{(q)}$ and $y_i^{(q)}$ are set to be $\perp$ for $Q_1 + 1 \le q \le Q$ and $\mathsf{Circuit}[\perp, i, \mathsf{K}_i, \{r^{(q)}, y_i^{(q)}\}_{q\in[Q]}, 1]$ is the circuit defined in Figure 13.
   (c) Set $\{\mathsf{ct}_i\}_{i\in[K']} := \{\text{1-CPFE.ct}_i\}_{i\in[K']}$ and $\mathsf{st} := \{\mathsf{K}_i\}_{i\in[K']}$.

5. Run $b \leftarrow \mathsf{A}^{\mathcal{O}'(\mathsf{st},\mathsf{msk},\cdot,\cdot)}(\text{1-CPFE.mpk}, \{\mathsf{ct}_i\}_{i\in[K']})$ where $\mathcal{O}'(\mathsf{st},\mathsf{msk},\cdot,\cdot)$ is an oracle that takes as input $\{y_i^{(q)}\}_{i\in[K']}$ and $x^{(q)}$ for $q \in [Q_1+1, Q_1+Q_2]$ and works as follows:
   (a) Choose $r^{(q)} \leftarrow \{0,1\}^\lambda$.
   (b) For $i \in [K']$, compute $\mathsf{SKE.ct}_i^{(q)} \leftarrow \mathsf{SKE.Enc}(\mathsf{K}_i, y_i^{(q)})$.
   (c) For $i \in [K'+1, K]$, choose $\mathsf{SKE.ct}_i^{(q)} \leftarrow \mathcal{C}$.
   (d) Run $\text{1-CPFE.sk}^{(q)} \leftarrow \text{1-CPFE.KeyGen}(\text{1-CPFE.msk}, (x^{(q)}, r^{(q)}, \{\mathsf{SKE.ct}_i\}_{i\in[K]}))$.
   (e) Output $\mathsf{sk}^{(q)} := (r^{(q)}, \{\mathsf{SKE.ct}_i\}_{i\in[K]}, \text{1-CPFE.sk}^{(q)})$.
6. Output b.

$\mathbf{Game}_1$: This game works similarly to $\mathbf{Game}_0$ except that if there are $q \neq q'$ such that $r^{(q)} = r^{(q')}$, it outputs a uniformly random bit. Since $r^{(q)}$ is uniformly chosen from $\{0,1\}^\lambda$ for each $q \in [Q_1+Q_2]$ and $Q_1+Q_2 \leq Q$, the probability that this happens is at most $Q^2 \cdot 2^{-\lambda}$. Therefore, we have

$$|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq Q^2 \cdot 2^{-\lambda} = \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$: This game is identical to $\mathbf{Game}_0$ except that $\text{1-CPFE.ct}_{K'}$ is generated as follows:

$$\text{1-CPFE.ct}_{K'} \leftarrow \text{1-CPFE.Enc}(\text{1-CPFE.mpk}, \mathsf{Circuit}[C, \bot, \bot, \{\bot,\bot\}_{q\in[Q]}, 0], 1^Q).$$

It is easy to see that

$$\begin{aligned}&\mathsf{Circuit}[\bot, i, \mathsf{K}_i, \{r^{(q)}, y_i^{(q)}\}_{q\in[Q]}, 1](x^{(q)}, r^{(q)}, \{\mathsf{SKE.ct}_i^{(q)}\}_{i\in[K]})\\ =\ &\mathsf{Circuit}[C, \bot, \bot, \{\bot,\bot\}_{q\in[Q]}, 0](x^{(q)}, r^{(q)}, \{\mathsf{SKE.ct}_i^{(q)}\}_{i\in[K]}) = C(x^{(q)})\end{aligned}$$

for all $q \in [Q_1+Q_2]$ unless there are $q \neq q'$ such that $r^{(q)} = r^{(q')}$, in which case, the game just outputs a uniformly random bit. Then, by AD-IND security of 1-CPFE, which immediately follows from AD-SIM-security, we have

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| = \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$: This game is identical to $\mathbf{Game}_2$ except that $\mathsf{SKE.ct}_{K'}^{(q)}$ for $q \in [Q_1+1, Q_2]$ is independently and randomly chosen from $\mathcal{C}$.
Noting that $\mathsf{K}_{K'}$ is no longer used for generating $\text{1-CPFE.ct}_{K'}$, by a straightforward reduction to the pseudorandom ciphertext property of SKE with a standard hybrid argument, we have

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| = \mathrm{negl}(\lambda).$$

We can see that $\mathbf{Game}_3$ is identical to $\mathsf{Exp}^{\mathsf{real}}_{K\text{-CPFE},\mathsf{A},\mathsf{Sim}}(1^\lambda)$. This completes the proof of Lemma 4.1.

C PIR with Shallow Answer Circuits

In this section, we show that there are PIR schemes that satisfy the efficiency properties required in Section 5.1 from various assumption. In particular, we observe that we have a PIR scheme with answer function in NC from LWE and a scheme with answer function in NC^1 from DDH and QR, where all of them achieve the polylogarithmic computational cost for the client. The LWE based construction is immediately obtained from FHE [BV11], whereas the DDH and QR based constructions are based on [DGI+19]. Since the DDH and QR based constructions are too complicated to directly describe, we follow the presentation by [DGI+19] in which a number of conversions are applied to the basic construction to obtain the final scheme. Since we are interested in the depth of the circuit implementing the answer function, we keep track of the depths of the circuits implementing the relevant functionalities for the intermediate primitives.

At first we see some of the definitions and previous results. We then provide the constructions of trapdoor hash for index predicates from the DDH assumption and the QR assumption by [DGI+19] in Appendices C.6 and C.9, respectively. From Appendices C.10 to C.12, we apply a number of conversions to the trapdoor hash above to obtain a string OT as is done in [DGI+19]. The construction of the PIR scheme by [DGI+19] is provided in Appendix C.13. Finally, we observe that the PIR scheme obtained from LWE has NC implementation of the answer function in Appendix C.14.

C.1 Definitions and Lemmas

Here, we summarize the necessary definitions and some useful facts.

C.2 Arithmetic Computations in NC^1.

Throughout this section, we will heavily rely on the following lemma, which says that (iterated) addition, (iterated) multiplication, and division can be computed in NC^1.

Lemma C.1 ([BCH84]). *For each of the following computations, there is corresponding circuit whose depth is $O(\log n)$. Here, we assume that integers are represented in binary forms and we identify $\{0,1\}^n$ and $\{0,1,\dots,2^n-1\}$.*

Addition: *Given $x, y \in \{0,1\}^n$, output $x+y$.*
Iterated Addition: *Given $x_1,\dots,x_n \in \{0,1\}^n$, output $\sum_{i\in[n]} x_i$.*
Multiplication: *Given $x, y \in \{0,1\}^n$, output xy.*
Iterated Multiplication: *Given $x_1,\dots,x_n \in \{0,1\}^n$, output $\prod_{i\in[n]} x_i$.*
Division: *Given $x, y \in \{0,1\}^n$, output $(x \mod y)$ and $\lceil x/y \rceil$.*

C.3 Error Correcting Codes.

We then introduce error correcting codes.

Definition C.1 (Error Correcting Codes). *A (binary) (N,n) code consists of a pair of efficiently computable functions $(\mathsf{Encode},\mathsf{Decode})$, where $\mathsf{Encode} : \{0,1\}^n \to \{0,1\}^N$ and $\mathsf{Decode} : \{0,1,\bot\}^N \to \{0,1\}^n$. The rate r of such a code is defined as $r := n/N$. We say that a code $(\mathsf{Encode},\mathsf{Decode})$ efficiently corrects from t errors if the following holds. Let $x \in \{0,1\}^n$ and $y' \in \{0,1\}^N$ be such that y' differs from $y = \mathsf{Encode}(x)$ in at most t positions . Then it holds that $\mathsf{Decode}(y') = x$.*

For our purpose, we need the following lemma.

Lemma C.2. *There is a family of binary code $\{\mathsf{Encode}_n, \mathsf{Decode}_n\}_n$ that can correct up to dN error for a constant $0 < d < 1/2$, that has a rate $0 < c < 1$, and there is a circuit with depth $O(\log n)$ that implements Encode_n.*

Proof. We claim that the concatenated code obtained by combining the random linear code and the Reed-Solomon code that is described in [Gol08, Appendix E.1.1.5] satisfies the required properties. Since the requirements regarding the error correction and the rate are already proven in [Gol08], we focus on the requirement on the depth fo the circuit. Recall that the encoding procedure of the above code sequentially applies the encoding algorithm of the random linear code and Reed-Solomon code. Therefore, it suffices to show that the encoding algorithms of both codes are in NC^1. The random linear codes simply compute the matrix multiplication over $\mathbb{Z}_2$ when it encodes a message. This can be straightforwardly implemented in NC^1. In the Reed-Solomon code, the encoding algorithm consists of the computation of polynomials over a finite filed on fixed inputs. By using a prime field, this computation can be implemented in NC^1 by Lemma C.1, since the computation of a polynomial over a prime field $\mathbb{F}_p$ can be implemented by the applications of iterated multiplications, iterated additions, and a division by p.

C.4 Trapdoor Hash Scheme.

Here, we introduce trapdoor hash scheme defined by [DGI+19]. We specialize the original definition to the setting of index predicate instead of more general functionality they consider, since the former is sufficient for our purpose.

Definition C.2 (Trapdoor Hash Scheme (TDH)). *A TDH scheme for index predicate is a tuple of five PPT algorithms $\mathcal{H} = (\mathsf{S},\mathsf{G},\mathsf{H},\mathsf{E},\mathsf{D})$ with the following properties.*

$\mathsf{S}(1^\lambda, 1^n) \to \mathsf{hk}$**:** *The sampling algorithm takes as input a security parameter λ and an input length n, and outputs a hash key hk.*

$\mathsf{G}(\mathsf{hk}, i) \to (\mathsf{ek}, \mathsf{td})$**:** *The generating algorithm takes as input a hash key* hk *and an index* $i \in [n]$*, and outputs a pair of an encoding key* ek *and a trapdoor* td*.*

$\mathsf{H}(\mathsf{hk}, x) \to \mathsf{h}$**:** *The hashing algorithm takes as input a hash key* hk *and a string* $x \in \{0,1\}^n$ *and deterministically outputs a hash value* $\mathsf{h} \in \{0,1\}^\eta$*.*

$\mathsf{E}(\mathsf{ek}, x) \to \mathsf{e}$**:** *The encoding algorithm takes as input an encoding key* ek *and string* $x \in \{0,1\}^n$ *and deterministically outputs an encoding* e*.*

$\mathsf{D}(\mathsf{td}, \mathsf{h}) \to (\mathsf{e}_0, \mathsf{e}_1)$**:** *The decoding algorithm takes as input a trapdoor* td*, a hash value* $h \in \{0,1\}^\eta$*, and outputs a pair of a 0-encoding and a 1-encoding* $(\mathsf{e}_0, \mathsf{e}_1)$*.*

Here, size of e *is referred to as the rate of the scheme.*

Definition C.3 (Weakly Correct TDH**).** *A* TDH *scheme* $\mathcal{H} = (\mathsf{S}, \mathsf{G}, \mathsf{H}, \mathsf{E}, \mathsf{D})$ *for the class of index predicates is* $(1-\epsilon)$*-weakly correct, for* $\epsilon := \epsilon(\lambda) < 1$*, if the following holds for any* $\lambda, n \in \mathbb{N}$*, any* $x \in \{0,1\}^n$*, and any predicate* $i \in [n]$*.*

$$\Pr[\mathsf{e} = \mathsf{e}_{x[i]}] \geq 1 - \epsilon - \mathrm{negl}(\lambda) \qquad \Pr[\mathsf{e} \neq \mathsf{e}_{1-x[i]}] \geq 1 - \epsilon - \mathrm{negl}(\lambda)$$

where $\mathsf{hk} \leftarrow \mathsf{S}(1^\lambda, 1^n)$*,* $(\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{G}(\mathsf{hk}, f)$*,* $\mathsf{h} := \mathsf{H}(\mathsf{hk}, x)$*,* $\mathsf{e} := \mathsf{E}(\mathsf{ek}, x)$*, and* $(\mathsf{e}_0, \mathsf{e}_1) \leftarrow \mathsf{D}(\mathsf{td}, \mathsf{h})$*.*

Security Notions For TDH, [DGI+19] defined two security notions. Input privacy stipulates that given the hash key hk and hash value h as input, the adversary learns no information about the underlying input x. Index privacy[23] means that given the hash key hk and encoding key ek as input, the adversary learns nothing about the index i. We omit the formal definitions here and refer to [DGI+19] for the details because they are not strictly necessary for our purpose.

Remark C.1. We note that the above definition of Trapdoor Hash Scheme is different from that given in [DGI+19] in that E and H are deterministic. The reason why they are randomized in [DGI+19] is that they need the input privacy for the scheme, which is necessary for their applications. However, for our purpose of constructing PIR, input privacy is not necessary and index privacy suffices, where the latter can be achieved even in the deterministic setting. Since we can slightly simplify their construction and presentation of our scheme in the deterministic setting, we modify the definition as above.

C.5 General Two-Message Protocols.

We then define general two-message protocol for sender-receiver functionality here. The intermediate primitives that appear in this section such as batch oblivious transfer can be captured as special cases of the protocol.

Definition C.4 (Two-message protocol for sender-receiver functionality). *A two-message protocol (a protocol for short) is a triple of* PPT *algorithms* $\Pi = (\Pi_1, \Pi_2, \Pi_3)$ *with the following syntax:*

$\Pi_1(1^\lambda, x) \to (\mathsf{st}, \mathsf{msg}_1)$**:** Π_1 *takes as input a security parameter* λ *and a receiver input* $x \in \{0,1\}^*$*, and outputs a receiver message* msg_1 *and a receiver state* st*.*

$\Pi_2(\mathsf{msg}_1, y) \to \mathsf{msg}_2$**:** Π_2 *takes as input a receiver message* msg_1 *and a sender input* $y \in \{0,1\}^*$*, and outputs a sender message* msg_2*.*

$\Pi_3(\mathsf{st}, \mathsf{msg}_2) \to z$**:** Π_3 *takes as input a receiver state* st *and a sender message* msg_2*, and outputs a receiver output* $z \in \{0,1\}^*$*.*

Correctness: Π is correct if there exists a negligible function $\mathrm{negl}(\lambda)$ such that following holds for all $\lambda \in \mathbb{N}$ and all $(x, y) \in \mathcal{D}_f$.

$$\Pr\left[z = f(x,y) \;\middle|\; \begin{array}{r} \Pi_1(1^\lambda, x) \to (\mathsf{st}, \mathsf{msg}_1) \\ \Pi_2(\mathsf{msg}_1, y) \to \mathsf{msg}_2 \\ \Pi_3(\mathsf{st}, \mathsf{msg}_2) \to z \end{array} \right] \geq 1 - \epsilon(\lambda)$$

[23] In the original definition, the notion is called function privacy instead of index privacy. This is because they consider general function instead of index predicate.

We say that Π is weakly correct if the above holds for $\epsilon(\lambda)$ that is vanishing (but possibly non-negligible).

Sender and receiver privacy: We can consider two security notions for the protocol. Reciever privacy requires that msg_1 does not reveal the information of x. Sender privacy requires that msg_2 does not reveal any information more than that is obtained from x and z. Since formal definitions are not necessary for our purpose, we omit the details and refer to [DGI+19].

Definition C.5 (Download Rate). *Let $0 \leq \omega \leq 1$. We say that a protocol $\Pi_f = (\Pi_1, \Pi_2, \Pi_3)$ for a sender-receiver functionality f has download rate ω if there exists a polynomial $B(\lambda)$ such for all polynomial-length input sequences $\{(x_\lambda, y_\lambda)\}_{\lambda\in\mathbb{N}}$ such that $(x_\lambda, y_\lambda) \in \mathcal{D}_f$ and $|f(x_\lambda, y_\lambda)| \geq B(\lambda)$ for all λ, we have*

$$\lim_{\lambda\to\infty} \inf \frac{|f(x_\lambda, y_\lambda)|}{m_\lambda} = \omega$$

where m_λ is the maximal length of sender's message when Π_f runs on inputs (x_λ, y_λ) and security parameter λ.

Definition C.6 (Batch Oblivious Transfer). *In batch (single-bit) OT, we consider a setting where a batch of independent single-bit OT instances are carried out in parallel. The sender's input consists of n k-tuples of single-bit secrets $\langle(s_{j,1}, \ldots, s_{j,k})\rangle_{j\in[n]}$, each corresponding to a single-bit OT instance, and the receiver's input is a tuple of n indices $\langle i_j\rangle_{j\in[n]}$, where $i_j \in [k]$ for every j. We require that the receiver's output contains the i_j^{th} secret of every OT instance j.*

$$\mathcal{D}_{\mathsf{bOT}} = \left\{(\langle i_j\rangle_{j\in[n]}, \langle(s_{j,1}, \ldots, s_{j,k})\rangle_{j\in[n]}) | n, k \in \mathbb{N} \text{ and } \forall j, l, i_j \in [k], s_{j,l} \in \{0,1\}\right\}$$

$$\mathsf{bOT}(\langle i_j\rangle_{j\in[n]}, \langle(s_{j,1}, \ldots, s_{j,k})\rangle_{j\in[n]}) = \langle s_{j,i_j}\rangle_{j\in[n]}$$

Definition C.7 (String Oblivious Transfer). *String OT (or just OT) is a generalization of the standard setting where the sender has k secret n-bit strings (rather than single bits). In fact, string OT can be seen as a special case of batch OT, where all OT instances share a common receiver's input.*

$$\mathcal{D}_{\mathsf{OT}} = \left\{((1^k, i), (s_1, \ldots, s_k)) \mid n, k \in \mathbb{N} \text{ and } i \in [k], \forall j, s_j \in \{0,1\}^n\right\}$$

$$\mathsf{OT}((1^k, i), (s_1, \ldots, s_k)) = s_i$$

C.6 Construction of Trapdoor Hash for Index Predicates from DDH

In this section, we present slightly modified version of [DGI+19] construction of trapdoor hash function (TDH) from the DDH assumption. We first show a scheme with rate $\frac{1}{\lambda}$ and then show how to enhance its rate at the cost of having non-negligible error probability.

For the purpose of having a shallow circuit for the answer function in the eventual PIR scheme constructed from the TDH, we instantiate the TDH scheme with a special group, while the original construction works for any prime-order groups. In particular, we use a group $\mathbb{G}$ of prime order p that is a subgroup of the multiplicative group $\mathbb{Z}_q^*$ for prime q. We denote a group generator that takes a security parameter 1^λ and then outputs the group description $\mathbb{G}$, generator g, and the group order $p > 2^{2\lambda}$ by $\mathcal{G}$. We assume that the group description $\mathcal{G}$ contains q and the DDH assumption holds with respect to $\mathcal{G}$.

C.7 Rate-$\frac{1}{\lambda}$TDH Construction.

Here, we present the construction of TDH scheme from DDH with rate-$\frac{1}{\lambda}$TDH given in [DGI+19].

Construction. The basic DDH based TDH scheme consists of 5 algorithms $\mathsf{TDH} = (\mathsf{S}, \mathsf{G}, \mathsf{H}, \mathsf{E}, \mathsf{D})$ as follows:

$\mathsf{S}(1^\lambda, 1^n)$: Proceed as follows:

1. Sample $(\mathbb{G}, p, g) \leftarrow \mathcal{G}$.

2. Sample a matrix $\mathbf{A} := \begin{pmatrix} g_{1,0}, g_{2,0}, \ldots, g_{n,0} \\ g_{1,1}, g_{2,1}, \ldots, g_{n,1} \end{pmatrix} \leftarrow \mathbb{G}^{2\times n}$.
3. Output $\mathsf{hk} := ((\mathbb{G}, p, g), \mathbf{A})$.

$\mathsf{G}(\mathsf{hk}, i)$: Parse hk as $\mathsf{hk} \to ((\mathbb{G}, p, g), \mathbf{A})$ and proceed as follow:
1. Sample $s, t \leftarrow \mathbb{Z}_p$.
2. Set $u := g^s$ and

$$\mathbf{B} := \begin{pmatrix} u_{1,0}, u_{2,0}, \ldots, u_{n,0} \\ u_{1,1}, u_{2,1}, \ldots, u_{n,1} \end{pmatrix} \qquad \text{where} \qquad u_{j,b} := \begin{cases} g_{j,b}^s g^t & \text{if } (j,b) = (i,1) \\ g_{j,b}^s & \text{otherwise} \end{cases}$$

3. Output $\mathsf{ek} := (u, \mathbf{B})$ and $\mathsf{td} := (s, t)$.

$\mathsf{H}(\mathsf{hk}, x)$: Parse hk as $\mathsf{hk} \to ((\mathbb{G}, p, g), \mathbf{A})$, $\mathbf{A} = (g_{j,b})_{j\in[n],b\in\{0,1\}}$ and output

$$\mathsf{h} := \prod_{j=1}^{n} g_{j,x[j]} \tag{C.1}$$

$\mathsf{E}(\mathsf{ek}, x)$: Parse ek as in $(u, \mathbf{B})$, $\mathbf{B} = (u_{j,b})_{j\in[n],b\in\{0,1\}}$ and output

$$\mathsf{e} := \prod_{j=1}^{n} u_{j,x[j]} \tag{C.2}$$

$\mathsf{D}(\mathsf{td}, \mathsf{h})$: Parse $\mathsf{h} \in \mathbb{G}$ and td as $\mathsf{td} \to (s, t)$ and output $\mathsf{e}_0 := h^s$ and $\mathsf{e}_1 := h^s g^t$.

As shown in [DGI+19], the above scheme satisfies perfect correctness and index privacy assuming the DDH assumption with respect to $\mathcal{G}$.

Remark C.2. We note that in the above scheme, E and H are deterministic, while they are randomized in [DGI+19]. This change makes the description of the scheme slightly simpler. The scheme is still index private even with this change (because the index privacy only refers to the distribution of hk and ek), though it looses input privacy. See also Remark C.1.

Depth of the Circuits. Here, we discuss the depth of the circuits implementing the computation of E and H, which will be relevant for our application. We observe that both algorithms consist of the computation of an iterated product followed by the computation of the residue modulo q. These steps are implemented by the circuits given by Lemma C.1, where the computation of iterated product is performed over the integers (without taking the modulus). Since each group element can be represented by a string of length $O(\log q) = O(\log \lambda)$ and $2n$ group elements are input to the circuit, the depth of the circuit is $O(\log \lambda n)$.

C.8 Rate-1 Construction.

In this section, we present the construction of rate-1 TDH scheme. Before describing the scheme, we introduce the following function.

Definition C.8 (The Distance Function [BGI16, DGI+19]). *Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p, and let $g \in \mathbb{G}$ be a generator. Let $\mathsf{Dist}_{\mathbb{G},g}$ be an algorithm that takes as input an element $h \in \mathbb{G}$, bounds on the failure probability $\delta > 0$ and on the input range $M \in \mathbb{N}$, and a key K of a pseudo-random function $\mathsf{PRF}_K : \mathbb{G} \to \{0,1\}^{\lceil \log(2M/\delta)\rceil}$ and outputs a bit.*

$\mathsf{Dist}_{\mathbb{G},g}(h, \delta, M, K)$**:** *It proceeds as follows.*
1. *Define $T := [2M \log_e(2/\delta)]/\delta$, and set $i := 0$.*
2. *While $i \le T$:*
   (a) If $\mathsf{PRF}_K(hg^i) = 0^{\lceil \log(2M/\delta)\rceil}$ then output $\mathsf{LSB}(i)$, else set $i := i + 1$.

Circuit implementing $\mathsf{Dist}_{\mathbb{G},g}(\cdot, 1/\lambda, 1, K)$**.**

Hardwired constants: Group element $g'_i := g^i$ and a bit $b_i := \mathsf{LSB}(i)$ for $i \in \{0, 1, \ldots, T+1\}$, PRF key K, a value $T = \lceil 2 \log_e 2\lambda \rceil \lambda$.
Input: A group element $h \in \mathbb{G}$.

1. For $i \in \{0, 1, \ldots, T+1\}$, do the following *in parallel*.
   (a) Compute $h'_i := h \cdot g'_i$.
   (b) Compute $y_i := \mathsf{PRF}_K(h'_i)$.
   (c) If $y_i = 0^{\lceil \log 2\lambda \rceil}$, set $\mathsf{pre}_i := 1$. Otherwise, $\mathsf{pre}_i = 0$.

Consider a binary tree whose leaves correspond to nodes $i = 0, 1, \ldots, T+1$, where the leftmost node corresponds to 1 and the index of the node increases as we move from a left leaf to a right leaf. The tree needs not be a complete tree, but is with depth $\lceil \log T \rceil$. A leaf node i is associated with (b_i, pre_i), where b_i is the hardwired value and pre_i is computed as above.

2. Compute the value (b_g, pre_g) associated with a node g for all nodes in the tree as follows. We go up the tree from the bottom to top.
   (a) Let L_g and R_g be the values associated with the left child and the right child of g, respectively.
   (b) Set $\mathsf{pre}_g := \mathsf{pre}_{L_g} \vee \mathsf{pre}_{R_g}$ and $b_g := \begin{cases} b_{L_g} & \text{If } \mathsf{pre}_{L_g} = 1 \\ b_{R_g} & \text{Otherwise.} \end{cases}$
3. Let (b_r, pre_r) be the value associated with the root node. Output b_r.

Fig. 14 Circuit implementing $\mathsf{Dist}_{\mathbb{G},g}(\cdot, 1/\lambda, 1, K)$.

3. Output $\mathsf{LSB}(i)$.
where LSB *return the least significant bit of the integer.*

For our purpose, we use a circuit that implements the same functionality as $\mathsf{Dist}_{\mathbb{G},g}(h, \lambda, 1, K)$ and has shallow depth. Specifically, we have the following lemma.

Lemma C.3. *Assuming a PRF function computable in* NC^1*, there exists a circuit that takes as input* $h \in \mathbb{G}$ *and outputs* $\mathsf{Dist}_{\mathbb{G},g}(h, 1/\lambda, 1, K)$ *whose depth is bounded by* $O(\log \lambda)$.

Proof. We claim that the circuit constructed as Figure 14 satisfies the requirements. First, it is easy to see that the functionality of the circuit is equivalent to $\mathsf{Dist}_{\mathbb{G},g}(\cdot, 1/\lambda, 1, K)$, since both circuits output $\mathsf{LSB}(i)$ for the smallest $i \in [0, T+1]$ such that $\mathsf{PRF}(h \cdot g^i) = 0^{\lceil \log 2\lambda \rceil}$ and $\mathsf{LSB}(T+1)$ otherwise. We then argue that the circuit can be implemented with depth $O(\log \lambda)$. By our assumption that PRF_K is in NC^1 and the fact that the multiplication in $\mathbb{G}$ can be implemented in NC^1, Step 1 of the circuit can be implemented with depth $O(\log \lambda)$. Step 2 can be implemented by a circuit of depth $O(\log T) = O(\log \lambda)$, where each node of the binary tree is replaced by a circuit of size $O(1)$. Therefore, the depth of the overall circuit is $O(\log \lambda)$ as desired.

We note that a PRF in NC^1 can be implemented by the DDH assumption on the group we use. We then show the construction of the TDH.

Construction. Let $\mathcal{H}' = (\mathsf{S}', \mathsf{G}', \mathsf{H}', \mathsf{E}', \mathsf{D}')$ be the basic DDH based TDH from the construction in Appendix C.6. Rate-1 TDH scheme for index predicates consists of the following algorithms.

$\mathsf{S}(1^\lambda, 1^n)$: Output $\mathsf{hk} \leftarrow \mathsf{S}'(1^\lambda, 1^n)$.
$\mathsf{G}(\mathsf{hk}, i)$: Sample $((u, \mathbf{B}), (s, t)) \leftarrow \mathsf{G}'(\mathsf{hk}, i)$ and a random PRF key K and output

$$\mathsf{ek} := (u, \mathbf{B}, t, K) \qquad \mathsf{td} := (s, t, K). \tag{C.3}$$

$\mathsf{H}(\mathsf{hk}, x)$: Output $\mathsf{h} := \mathsf{H}'(\mathsf{hk}, x)$

$\mathsf{E}(\mathsf{ek}, x)$: Parse ek as in equation (C.3), set $e := \mathsf{E}'(\mathsf{ek}, x)$, and output

$$\mathsf{e} := \mathsf{Dist}_{g^t}(e, 1/\lambda, 1, K). \tag{C.4}$$

$\mathsf{D}(\mathsf{td}, \mathsf{h})$: Parse $\mathsf{h} \in \mathbb{G}$ and td as in equation (C.3), set $(\mathsf{e}_0, \mathsf{e}_1) := \mathsf{D}'(s, \mathsf{h})$ and output

$$\mathsf{e}_0 := \mathsf{Dist}_{g^t}(e_0, 1/\lambda, 1, K) \qquad \mathsf{e}_1 := \mathsf{Dist}_{g^t}(e_1, 1/\lambda, 1, K). \tag{C.5}$$

As shown in [DGI+19], the above scheme satisfies correctness with probability $1 - 1/\lambda$, where the probability is taken over the choice of K. It inherits the index privacy of the rate-$\frac{1}{\lambda}$ construction.

Depth of the Circuits. Here, we discuss the depth of the circuits implementing the computation of E and H, which will be relevant for our application. The depth of circuit implementing H' is the same as that for H', which is $O(\log \lambda n)$. We implement the computation of E' by a circuit that sequentially computes e and e. Then the depth of the circuit is $O(\log \lambda n) + O(\log \lambda) = O(\log \lambda n)$ by Lemma C.3.

C.9 Construction of Trapdoor Hash for Index Predicates from QR

In this section we present [DGI+19] construction of rate-1 TDH scheme for index predicates from the QR assumption.

Comparison Operator. Define the function $\mathsf{LEq} : \mathbb{J}_N \times \mathbb{J}_N \to \{0, 1\}$ to return 1 if the bit representation of the first input is smaller or equal than the bit representation of the second input according to some order (e.g., lexicographical).

Construction. The basic QR based TDH scheme consists of 5 algorithms $\mathsf{TDH} = (\mathsf{S}, \mathsf{G}, \mathsf{H}, \mathsf{E}, \mathsf{D})$ as follows:

$\mathsf{S}(1^\lambda, 1^n)$: Proceed as follows:

1. Sample a blum integer $N = pq$, where $p = q = 3 (\mathrm{mod}\ 4)$.
2. Sample $g \leftarrow \mathbb{QR}_N$.
3. Sample a matrix $\mathbf{A} := \begin{pmatrix} g_{1,0}, g_{2,0}, \ldots, g_{n,0} \\ g_{1,1}, g_{2,1}, \ldots, g_{n,1} \end{pmatrix} \leftarrow \mathbb{QR}^{2\times n}$
4. Output $\mathsf{hk} := ((N, g), \mathbf{A})$

$\mathsf{G}(\mathsf{hk}, i)$: Parse hk as $\mathsf{hk} \to ((N, g), \mathbf{A})$ and proceed as follows:

1. Sample $s \leftarrow \left[\frac{N-1}{2}\right]$.
2. Set $u := g^s$ and

$$\mathbf{B} := \begin{pmatrix} u_{1,0}, u_{2,0}, \ldots, u_{n,0} \\ u_{1,1}, u_{2,1}, \ldots, u_{n,1} \end{pmatrix} \qquad \text{where} \qquad u_{j,b} := \begin{cases} g_{j,b}^s (-1)^{\alpha_j} & \text{if } b = 1 \\ g_{j,b}^s & \text{otherwise} \end{cases}$$

3. Output $\mathsf{ek} := (u, \mathbf{B}) \qquad \mathsf{td} := s$

$\mathsf{H}(\mathsf{hk}, x)$: Parse hk as $\mathsf{hk} \to ((N, g), \mathbf{A})$, $\mathbf{A} = (g_{j,b})_{j\in[n], b\in\{0,1\}}$ and output

$$\mathsf{h} := \prod_{j=1}^{n} g_{j,x[j]}. \tag{C.6}$$

$\mathsf{E}(\mathsf{ek}, x)$: Parse ek as in $(u, \mathbf{B})$, $\mathbf{B} = (u_{j,b})_{j\in[n], b\in\{0,1\}}$, set

$$\mathsf{e} := \prod_{j=1}^{n} u_{j,x[j]} \tag{C.7}$$

and output $\mathsf{e} := \mathsf{LEq}(\mathsf{e}, (-1)\mathsf{e})$.

$\mathsf{D}(\mathsf{td}, \mathsf{h})$: Parse $\mathsf{h} \in \mathbb{QR}$ and td as $\mathsf{td} \to s$ and output $\mathsf{e}_0 := \mathsf{LEq}(\mathsf{h}^s, (-1)\mathsf{h}^s)$ and $\mathsf{e}_1 := \mathsf{LEq}((-1)\mathsf{h}^s, \mathsf{h}^s)$.

As shown in [DGI+19], the above scheme satisfies perfect correctness and index privacy assuming the QR assumption with respect to $\mathbb{QR}_N$.

Remark C.3. Similarly to our DDH based construction, E and H are deterministic, while they are randomized in the original construction in [DGI+19]. The scheme is still index private even with this change, though it looses input privacy. See also Remark C.1.

Depth of the Circuits. Here, we discuss the depth of the circuits implementing the computation of E and H, which will be relevant for our application. We observe that the algorithm H consists of the computation of an iterated product followed by the computation of the residue modulo q. These steps are implemented by the circuits given by Lemma C.1, where the computation of iterated product is performed over the integers (without taking the modulus). Since each group element can be represented by a string of length $O(\log q) = O(\log \lambda)$ and $2n$ group elements are input to the circuit, the depth of the circuit is $O(\log \lambda n)$.

The algorithm E also consists of the computation of an iterated product followed by the computation of the residue modulo q. It then uses comparison operator. The comparison between integers x and y can be done in NC^1 by computing $\lceil x/y \rceil$ and see if it is 0 or not. Hence, the depth of the circuit is $O(\log \lambda n)$.

C.10 Construction of Rate-1 Batch Oblivious Transfer from Trapdoor Hash

In this section, we present the construction of rate-1 batch OT from the TDH in [DGI+19].[24] The underlying TDH can be either the one from DDH or QR.

Construction. Let $\mathcal{H} = (\mathsf{S}, \mathsf{G}, \mathsf{H}, \mathsf{E}, \mathsf{D})$ be the rate-1 (weakly) correct TDH for index predicates. A batch OT scheme $\mathsf{bOT} = (\mathsf{bOT}_1, \mathsf{bOT}_2, \mathsf{bOT}_3)$ proceeds as follows.

$\mathsf{bOT}_1(1^\lambda, 1^k, \langle i_j \rangle_{j\in[n]})$: proceed as follows:

1. Sample $\mathsf{hk} \leftarrow \mathsf{S}(1^\lambda, 1^{nk})$.
2. For every $j = 1, \ldots, n$, sample $(\mathsf{ek}_j, \mathsf{td}_j) \leftarrow \mathsf{G}(\mathsf{hk}, f_{[(j-1)k+i_j]})$.
3. Output

$$\mathsf{st} := \left(\langle i_j \rangle_{j\in[n]}, \mathsf{hk}, \mathsf{td}_1, \ldots, \mathsf{td}_n\right), \quad \mathsf{msg}_1 := (\mathsf{hk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_n) \tag{C.8}$$

$\mathsf{bOT}_2(\mathsf{msg}_1, \langle s_j = (s_{j,1}, \ldots, s_{j,k}) \rangle_{j\in[n]})$: Parse msg_1 as in equation (C.8) and proceed as follows:

1. Compute $\mathsf{h} \leftarrow \mathsf{H}(\mathsf{hk}, \langle (s_{1,1}, \ldots, s_{1,k}), \ldots (s_{n,1}, \ldots, s_{n,k}) \rangle)$.
2. For every $j \in [n]$, compute $\mathsf{e}_j \leftarrow \mathsf{E}(\mathsf{ek}_j, \langle (s_{1,1}, \ldots, s_{1,k}), \ldots (s_{n,1}, \ldots, s_{n,k}) \rangle)$.
3. Output

$$\mathsf{msg}_2 := (\mathsf{h}, \mathsf{e}_1, \ldots, \mathsf{e}_n). \tag{C.9}$$

$\mathsf{bOT}_3(\mathsf{st}, \mathsf{msg}_2)$: Parse st and msg_2 as in equation (C.8) and (C.9) and proceed as follows:

1. For every $j = 1, \ldots, n$,
   (a) Compute $(\mathsf{e}_{j,0}, \mathsf{e}_{j,1}) \leftarrow \mathsf{D}(\mathsf{td}_j, \mathsf{h})$.
   (b) If $\mathsf{e}_{j,0} = \mathsf{e}_{j,1}$, set $\tilde{s}_j := \bot$.
   (c) Otherwise, if $\mathsf{e}_j = \mathsf{e}_{j,0}$, set $\tilde{s}_j := 0$.
   (d) Otherwise, if $\mathsf{e}_j = \mathsf{e}_{j,1}$, set $\tilde{s}_j := 1$.
2. Output $\langle \tilde{s}_1, \ldots, \tilde{s}_n \rangle$.

As shown in [DGI+19], when the underlying TDH is $(1-\epsilon)$-correct for a non-negligible ϵ, we obtain a weakly correct batch OT protocol, where a failure in each of the instances in the batch occurs independently and with probability ϵ. If the underlying TDH has perfect correctness, so does the above construction. It is also shown that the above construction has receiver privacy and download rate 1 if the underlying TDH is index private and has rate 1.

Depth of the Circuits. Here, we discuss the depth of the circuit implementing bOT_2, which is relevant for our purpose. We can implement the computation of bOT_2 by the sequential application of E followed by *parallel* computation of H. Therefore, regardless of whether we instantiate TDH from DDH or QR, we can implement the function bOT_2 by a circuit of depth $O(\log \lambda nk)$.

[24] Since we discard sender privacy, the construction here might have to be called PIR rather than OT. However, we stick to the term "OT" in order to be consistent with the description in [DGI+19].

C.11 Construction of String OT from (Weakly Correct) Batch OT

Here, we convert the batch OT scheme constructed in Appendix C.10 to be a string OT scheme. Furthermore, we reduce the correctness error probability of the scheme to be negligible even in the case where the underlying batch OT has non-negligible error probability using the error correcting codes.

Construction. Let $\mathsf{bOT} = (\mathsf{bOT}_1, \mathsf{bOT}_2, \mathsf{bOT}_3)$ be a batch OT protocol constructed in Appendix C.10 and $\{(\mathsf{Encode}_n, \mathsf{Decode}_n)\}_{n\in\mathbb{N}}$ be a family of error-correcting codes that is described in Lemma C.2, which can correct up to dN errors and has rate c for constants $0 < c < 1$ and $0 < d < 1/2$. Here, we assume that $dN = \Omega(\lambda)$. The string OT protocol $\mathsf{sOT} = (\mathsf{sOT}_1, \mathsf{sOT}_2, \mathsf{sOT}_3)$ consists of the following algorithms.

$\mathsf{sOT}_1(1^\lambda, 1^n, 1^k, i)$: Output $(\mathsf{st}, \mathsf{msg}_1) \leftarrow \mathsf{bOT}_1(1^\lambda, 1^k, \langle i^{N_n}\rangle)$.
$\mathsf{sOT}_2(\mathsf{msg}_1, (s_1, \ldots, s_k))$: Proceed as follows:

1. For every $j = 1, \ldots, k$, compute $(z_{1,j}, \ldots, z_{N,j}) \leftarrow \mathsf{Encode}_n(s_j)$.
2. Output $\mathsf{msg}_2 \leftarrow \mathsf{bOT}_2(\mathsf{msg}_1, \langle (z_{1,1}, \ldots, z_{1,k}), \ldots, (z_{N,1}, \ldots, z_{N,k})\rangle)$.

$\mathsf{sOT}_3(\mathsf{st}, \mathsf{msg}_2)$: Compute $z \leftarrow \mathsf{bOT}_3(\mathsf{st}, \mathsf{msg}_2)$ and output $\tilde{s} \leftarrow \mathsf{Decode}_{\lambda,n}(z)$.

Remark C.4. The syntax of the construction above is slightly different from that of string OT defined in Definition C.7 in that sOT_1 takes 1^n as input. This is fixed in Appendix C.12 similarly to [DGI+19].

Remark C.5. In [DGI+19], the authors use error correcting codes with extremely high rate for the purpose of having rate 1 in the resulting string OT construction. In our case, it suffices to have a constant rate and thus we use simpler error correcting code with lower rate, for which it is easy to show that the encoding algorithm is in NC^1 (See Lemma C.2).

Correctness. If the correctness error of bOT is negligible, it is clear that sOT has negligible error as well. Even in the case where bOT has non-negligible error of $1/\lambda$, which is the case for the construction from DDH, sOT has negligible error as is shown in [DGI+19].

Download Rate. Since the asymptotic download rate of bOT is 1 and the rate of the error correcting code we use is a constant $1/c$ for $c > 1$, the asymptotic download rate of sOT is $1/c$. The rate of sOT is strictly smaller than 1 due to our choice of error correcting code. However, we still have constant rate and this is sufficient for our purpose of constructing PIR.

Depth of the Circuits. Here, we discuss the depth of the circuit implementing sOT_2, which is relevant for our purpose. We compute sOT_2 by the sequential application of Encode_n in *parallel* followed by the computation of bOT_2. The former is implemented by a circuit of depth $O(\log n)$ by Lemma C.2. Furthermore, as we have observed, the latter can be implemented by a circuit of depth $O(\log \lambda nk)$. Therefore, the total depth of the circuit is $O(\log \lambda nk)$.

C.12 Making the First Message Fixed Polynomial Size

The string OT scheme sOT we saw in Appendix C.11 has constant download rate, but the length of the first message is polynomially dependent on n. We remove this restriction and make the length of the first message a fixed polynomial in the security parameter. The construction is essentially the same as [DGI+19], but we use slightly different parameter. The description of the construction follows.

Construction. Let $\mathsf{sOT} = (\mathsf{sOT}_1, \mathsf{sOT}_2, \mathsf{sOT}_3)$ be a string OT scheme constructed in Appendix C.10 and $B(\lambda) = \mathrm{poly}(\lambda)$ be a polynomial chosen later. A string OT protocol OT protocol $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ consists of the following algorithms.

$\mathsf{OT}_1(1^\lambda, 1^k, i)$: Output $(\mathsf{st}, \mathsf{msg}_1) \leftarrow \mathsf{sOT}_1(1^\lambda, 1^k, 1^B, i)$.
$\mathsf{OT}_2(\mathsf{msg}_1, (s_1, \ldots, s_k))$: Proceed as follows:

1. For every $j = 1, \ldots, k$, divide the string $s_j \in \{0,1\}^n$ as $s_j \to (s_{j,1}, \ldots, s_{j,n'})$, where $n' = \lceil n/B \rceil$ and $s_{j,t} \in \{0,1\}^B$ for all t.
2. For every $t = 1, \ldots, n'$, compute $\mathsf{msg}_{2,t} \leftarrow \mathsf{sOT}_2(\mathsf{msg}_1, (s_{1,t}, \ldots, s_{k,t}))$.
3. Output $\mathsf{msg}_2 := \{\mathsf{msg}_{2,t}\}_{t\in[n']}$.

$\mathsf{OT}_3(\mathsf{st}, \mathsf{msg}_2)$: For every $t = 1, \ldots, n'$, compute $z_t \leftarrow \mathsf{sOT}_3(\mathsf{st}, \mathsf{msg}_{2,t})$ and output $z := (z_1, \ldots, z_{n'})$.

Correctness and Security. It is obvious that the correctness and the receiver privacy are preserved by the above conversion.

Communication Complexity. Since download rate of sOT is $1/c$, the above construction has a rate $1/c_B$ for a constant $c_B > c$ by making $B(\lambda)$ sufficiently large polynomial. Once we fix $B(\lambda)$, it is clear that the size of the first message is a fixed polynomial in λ.

Depth of the Circuits. Here, we discuss the depth of the circuit implementing OT_2, which is relevant for our purpose. Since the computation can be performed in parallel, the depth of the circuit computing OT_2 is the same as that of sOT_2. In particular, the depth of the circuit can be bounded by $O(\log \lambda B k) = O(\log \lambda k)$.

C.13 PIR Construction

The following is the PIR constrution of [DGI+19].

Construction. Let $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ be the string OT protocol constructed in Appendix C.12 and $B(\lambda)$ to be the polynomial chosen there. In the construction, we assume that our database DB consists of N entries, where $N = \lambda^d$ holds exactly for some $d \in \mathbb{N}$. This is without loss of generality because if there is no such $d \in \mathbb{N}$, we can pad the database. The PIR protocol $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ consists of the following algorithms.

$\mathsf{Query}(1^\lambda, i, N)$: Parse $i \in [N]$ as $(i_1, \dots, i_d)$, where $i_j \in [\lambda]$ for all j, and proceed as follows:

1. For all $\ell \in [d]$ compute $\left(\mathsf{st}^{(\ell)}, \mathsf{msg}_1^{(\ell)}\right) \leftarrow \mathsf{OT}_1(1^\lambda, 1^\lambda, i_\ell)$.
2. Output a query query and an internal state st defined as follows:

$$\mathsf{query} = \left(\mathsf{msg}_1^{(1)}, \dots, \mathsf{msg}_1^{(d)}\right) \qquad \mathsf{st} = \left(\mathsf{st}_1^{(1)}, \dots, \mathsf{st}_1^{(d)}\right) \tag{C.10}$$

$\mathsf{Answer}(\mathsf{query}, \mathsf{DB}, 1^N)$: Parse query as in Eq. (C.10) and proceed as follows:

1. Set $Y = (\mathsf{DB}[1]^B, \mathsf{DB}[2]^B, \dots, \mathsf{DB}[N]^B)$, where $\mathsf{DB}[i]^B$ means B times repetition of the bit $\mathsf{DB}[i]$.[25]
2. Set $B^{(1)} = B$.
3. For every $\ell = 1, \dots, d$,
   (a) Parse Y as the concatenation of N/λ^ℓ tuples, where the j-th tuple consists of λ entries $(y_{j,1}, \dots, y_{j,\lambda})$, each of length $B^{(\ell)}$ bits.
   (b) For all $j \in [N/\lambda^\ell]$, set $\mathsf{msg}_{2,j}^{(\ell)} \leftarrow \mathsf{OT}_2\left(\mathsf{query}^{(\ell)}, (y_{j,1}, \dots, y_{j,\lambda})\right)$.
   (c) Update $Y \leftarrow \left(\mathsf{msg}_{2,1}^{(\ell)}, \dots, \mathsf{msg}_{2,N/\lambda^\ell}^{(\ell)}\right)$ and $B^{(\ell+1)} = \left|\mathsf{msg}_{2,\cdot}^{(\ell)}\right|$.
4. Output $\mathsf{answer} = \mathsf{msg}_{2,1}^{(d)}$.

$\mathsf{Reconstruct}(\mathsf{st}, \mathsf{answer}, N)$: parse st as in Eq. (C.10) and proceed as follows:

1. Set $\mathsf{msg}_2^{(d)} = \mathsf{answer}$.
2. For every $\ell = 1, \dots, d$, set $\mathsf{msg}_2^{(\ell-1)} \leftarrow \mathsf{OT}_3\left(\mathsf{msg}_2^{(\ell)}, \mathsf{st}^{(\ell)}\right)$.
3. Output the first bit $\mathsf{msg}_2^{(0)}[1]$ of $\mathsf{msg}_2^{(0)}$.

Correctness and Security. As is shown in [DGI+19], PIR satisfies correctness if so does OT. Furthermore, assuming OT is receiver private, PIR is private.

Complexity of Query and Reconstruct. Here, we discuss that PIR satisfies the efficiency properties as per Definition 2.9. Namely, we show that Query and Reconstruct run in fixed polynomial time for any N with size $N \leq 2^{\log^2 \lambda}$. We first observe that Query runs in a fixed polynomial time that is independent of N, since each execution of $\mathsf{OT}_1(1^\lambda, 1^\lambda, i_\ell)$ runs in fixed polynomial time for each ℓ and there are d executions, where $d = \log_\lambda N \leq \log \lambda$. We then observe that Reconstruct runs in a fixed polynomial that is independent of N. To see this, it suffices to bound the size of the input to OT_3 by a fixed polynomial. It follows that the size of $\mathsf{st}^{(\ell)}$ is fixed polynomial because so is the

[25] In [DGI+19], it is assumed that the database consists of N entries, each of size B. We encode DB into Y to meet this requirement.

running time of Query. Furthermore, we have $B^{(\ell+1)}/B^{(\ell)} \to 1/c_B$ with $\lambda \to \infty$ and this means $|\mathsf{msg}_2^\ell|/B \to 1/c_B^\ell$ with $\lambda \to \infty$ for all $\ell \in [d]$ by induction. Since we have

$$(1/c_B)^d = (1/c_B)^{\log_\lambda N} \leq (1/c_B)^{\log \lambda} = \lambda^{\log(1/c_B)},$$

the input to OT_3 in the reconstruction algorithm is bounded by a fixed polynomial in λ as desired.

Depth of the Circuits. Here, we discuss the depth of the circuit implementing $\mathsf{Answer}(\cdot, \mathsf{DB}, 1^N)$. We first observe that each invocation of OT_2 can be implemented by a circuit of depth $O(\log \lambda)$ by the analysis in Appendix C.12, since we have $k = \lambda$ here. Since we need $d = \log_\lambda |\mathsf{DB}|$ times of sequential computation in the function, the depth of the overall computation is bounded by $O(\log \lambda) \cdot \log_\lambda |\mathsf{DB}| = O(\log |\mathsf{DB}|)$.

C.14 PIR from LWE with Shallow Answer Function

Here, we sketch a PIR scheme from LWE with NC circuit implementation of the answer function and with polylogarithmic computational cost for the client. To obtain such a scheme, we follow the well-known construction of PIR from leveled FHE [BV11]. The idea is to let the client who wants to know the i-th entry of the database generate FHE public and secret key pair and encrypt i using an FHE, where i is represented in a binary form. Then, the server with the database homomorphically computes the encryption of

$$\sum_{j \in [N]} (i \stackrel{?}{=} j) \cdot \mathsf{DB}[j]$$

from the ciphertext and the database, where $\mathsf{DB}[j]$ and N denote the j-th bit and the size of the database DB, respectively. Note that the above computation can be implemented with depth $O(\log N)$ circuit if we consider the computation on plaintext. Since (leveled) FHE scheme such as [GSW13] supports homomorphic multiplication in NC^1, the homomorphic evaluation of the above can be implemented in $\mathrm{poly}(\log N, \log \lambda) = \mathrm{poly}(\log \lambda)$ as required.