Compact FE for Unbounded Attribute-Weighted Sums for Logspace from SXDH

Pratish Datta[0000−0002−3938−7594]1, Tapas Pal[0000−0001−6278−0418]2 and Katsuyuki Takashima[0000−0001−5216−2229]3

[1] NTT Research, Inc., Sunnyvale, CA 94085, USA
pratish.datta@ntt-research.com,
[2] NTT Social Informatics Laboratories, Musashino-shi, Tokyo, Japan 180-8585
tapas.pal.wh@hco.ntt.co.jp,
[3] Waseda University, Shinjuku-ku, Tokyo, Japan 169-8050
ktakashima@waseda.jp

Abstract. This paper presents the *first* functional encryption (FE) scheme for the attribute-weighted sum (AWS) functionality that supports the *uniform* model of computation. In such an FE scheme, encryption takes as input a pair of attributes (x, z) where the attribute x is public while the attribute z is private. A secret key corresponds to some weight function f, and decryption recovers the weighted sum $f(x)z$. This is an important functionality with a wide range of potential real life applications, many of which require the attribute lengths to be flexible rather than being fixed at system setup. In the proposed scheme, the public attributes are considered as binary strings while the private attributes are considered as vectors over some finite field, both having arbitrary polynomial lengths that are not fixed at system setup. The weight functions are modelled as Logspace Turing machines.

Prior schemes [Abdalla, Gong, and Wee, CRYPTO 2020 and Datta and Pal, ASIACRYPT 2021] could only support non-uniform Logspace. The proposed scheme is built in asymmetric prime-order bilinear groups and is proven *adaptively simulation* secure under the well-studied symmetric external Diffie-Hellman (SXDH) assumption against an arbitrary polynomial number of secret key queries both before and after the challenge ciphertext. This is the best possible level of security for FE as noted in the literature. As a special case of the proposed FE scheme, we also obtain the first adaptively simulation secure inner-product FE (IPFE) for vectors of arbitrary length that is not fixed at system setup.

On the technical side, our contributions lie in extending the techniques of Lin and Luo [EUROCRYPT 2020] devised for payload hiding attribute-based encryption (ABE) for uniform Logspace access policies avoiding the so-called "one-use" restriction in the indistinguishability-based security model as well as the "three-slot reduction" technique for simulation-secure attribute-hiding FE for non-uniform Logspace devised by Datta and Pal [ASIACRYPT 2021] to the context of simulation-secure attribute-hiding FE for uniform Logspace.

Keywords: functional encryption, attribute-weighted sums, Logspace, Turing machines

1 Introduction

Functional Encryption: *Functional encryption* (FE), formally introduced by Boneh et al. [8] and O'Neill [22], redefines the classical encryption procedure with the motivation to overcome the limitation of the "all-or-nothing" paradigm of decryption. In a traditional encryption system, there is a single secret key such that a user given a ciphertext can either recover the whole message or learns nothing about it, depending on the availability of the secret key. FE in contrast provides fine grained access control over encrypted data by generating artistic secret keys according to the desired functions of the encrypted data to be disclosed. More specifically, in a public-key FE scheme for a function class $\mathcal{F}$, there is a setup authority which produces a master secret key and publishes a master public key. Using the master secret key, the setup authority can derive secret keys or functional decryption keys SK_f associated with functions $f \in \mathcal{F}$. Anyone can encrypt messages msg belonging to a specified message space $\mathsf{msg} \in \mathbb{M}$ using the master public key to produce a ciphertext CT. The ciphertext CT along with a secret key SK_f recovers the function of the message $f(\mathsf{msg})$ at the time of decryption, while unable to extract any other information about msg. More specifically, the security of FE requires *collusion resistance* meaning that any polynomial number of secret keys together cannot gather more information about an encrypted message except the union of what each of the secret keys can learn individually.

FE for Attribute-Weighted Sum: Recently, Abdalla, Gong and Wee [2] and Datta and Pal [13] studied FE schemes for a new class of functionalities termed as "attribute-weighted sums" (AWS). This is a generalization of the inner product functional encryption (IPFE) [1,4]. In such a scheme, an attribute pair (x, z) is encrypted using the master public key of the scheme, where x is a public attribute (e.g., demographic data) and z is a private attribute containing sensitive information (e.g., salary, medical condition, loans, college admission outcomes). A recipient having a secret key corresponding to a weight function f can learn the attribute-weighted sum $f(x)z$. The attribute-weighted sum functionality appears naturally in several real life applications. For instance, as discussed by Abdalla et al. [2] if we consider the weight function f as a boolean predicate, then the attribute-weighted sum functionality $f(x)$ would correspond to the average z over all users whose attribute x satisfies the predicate f. Important practical scenarios include average salaries of minority groups holding a particular job (z = salary) and approval ratings of an election candidate amongst specific demographic groups in a particular state (z = rating).

The works of [2, 13] considered a more general case of the notion where the domain and range of the weight functions are vectors, in particular, the attribute pair of public/private attribute vectors $(\boldsymbol{x}, \boldsymbol{z})$, which is encrypted to a ciphertext CT. A secret key SK_f generated for a weight function f allows a recipient to learn $f(\boldsymbol{x})^\top \boldsymbol{z}$ from CT without leaking any information about the private attribute $\boldsymbol{z}$.

The FE schemes of [2, 13] support an expressive function class of *arithmetic branching programs* (ABPs) which captures non-uniform Logspace computations. Both schemes were built in asymmetric bilinear groups of prime order and are

proven secure in the simulation-based security model, which is known to be the desirable security model for FE [22, 8], under the (bilateral) k-Linear (k-Lin)/ (bilateral) *Matrix Diffie-Hellman* (MDDH) assumption. The FE scheme of [2] achieves semi-adaptive security, where the adversary is restricted to making secret key queries only after making the ciphertext queries, whereas the FE scheme of [13] achieves adaptive security, where the adversary is allowed to make secret key queries both before and after the ciphertext queries.

However, as mentioned above, ABP is a non-uniform computational model. As such, in both the FE schemes [2, 13], the length of the public and private attribute vectors must be fixed at system setup. This is clearly a bottleneck in several applications of this primitive especially when the computation is done over attributes whose lengths vary widely among ciphertexts and are not fixed at system setup. For instance, suppose a government hires an external audit service to perform a survey on average salary of employees working under different job categories in various companies to resolve salary discrepancy.The companies create salary databases (X, Z) where $X = (x_i)_i$ contains public attributes $x_i = (\text{job title}, \text{department}, \text{company name})$ and $Z = (z_i)_i$ includes private attribute $z_i = \text{salary}$. To facilitate this auditing process without revealing individual salaries (private attribute) to the auditor, the companies encrypt their own database (X, Z) using an FE scheme for AWS. The government provides the auditor a functional secret key SK_f for a function f that takes input a public attribute X and outputs 1 for x_i's for which the "job title" matches with a particular job, say *manager*. The auditor decrypts ciphertexts of the various companies using SK_f and calculates the average salaries of employees working under that job category in those companies. Now, if the existing FE schemes for AWS [2, 13] supporting non-uniform computations are employed then to make the system *sustainable* the government would have to fix a probable size (an upper bound) of the number of employees in all the companies. Also, the size of all ciphertexts ever generated would scale with that upper bound even if the number of employees in some companies, at the time of encryption, are much smaller than that upper bound. This motivates us to consider the following problem.

Open Problem *Can we construct an* FE *scheme for* AWS *in some uniform computational model capable of handling public/private attributes of arbitrary length?*

Our Results. This work resolves the above open problem. For the *first* time in the literature, we formally define and construct a FE scheme for *unbounded* AWS (UAWS) functionality where the setup only depends on the security parameter of the system and the weight functions are modeled as Turing machines. The proposed FE scheme supports both public and private attributes of *arbitrary* lengths. In particular, the public parameters of the system are completely independent of the lengths of attribute pairs. Moreover, the ciphertext size is *compact* meaning that it does not grow with the number of occurrences of a specific attribute in the weight functions which are represented as Logspace Turing machines. The scheme is *adaptively simulation* secure against the release of an unbounded (polynomial) number of secret keys both before and after the challenge ciphertext. As noted in [8, 22], simulation security is the best possible and the

most desirable model for FE. Moreover, simulation-based security also captures indistinguishability-based security but the converse does not hold in general.

Our FE for UAWS is proven secure in the standard model based on the symmetric external Diffie-Hellman (SXDH) assumption in the asymmetric prime-order pairing groups. Our main result in the paper is summarized as follows.

Theorem 1.1 (Informal) *Assuming the* SXDH *assumption holds in asymmetric pairing groups of prime-order, there exists an adaptively simulation secure* FE *scheme for the attribute weighted sum functionality with the weight functions modeled as Logspace Turing machines such that the lengths of public and private attributes are unbounded and can be chosen at the time of encryption, the ciphertexts are compact with respect to the multiple occurrences of attributes in the weight functions.*

Viewing IPFE as a special case of FE for AWS, we also obtain the *first* adaptively *simulation* secure IPFE scheme for *unbounded* length vectors (UIPFE), i.e., the length of the vectors is not fixed in setup. Observe that all prior simulation secure IPFE [26, 2, 3, 13] could only support *bounded* length vectors, i.e., the lengths must be fixed in the setup. On the other hand, the only known construction of UIPFE [23] is proven secure in the *indistinguishability-based* model.

The proposed FE construction is semi-generic and extends the frameworks of the works of Lin and Luo [18] and Datta and Pal [13]. Lin and Luo [18] develop an adaptively secure attribute-based encryption (ABE) scheme for Logspace Turing machines proven secure in the indistinguishability-based model. Although the input length of their ABE is unbounded, but an ABE is an "*all-or-nothing*" type primitive which fully discloses the message to a secret key generated for accepting policies. Further, the ABE of [18] is only payload hiding secure meaning that the ciphertexts themselves can leak sensitive information about the associated attributes. In contrast, our FE for UAWS provides more fine grained encryption methodologies where the ciphertexts reveal nothing about the private part of associated attributes but their weighted sums. Our FE construction depends on two building blocks, an *arithmetic key garbling scheme* (AKGS) for Logspace Turing machines which is an information-theoretic tool and a function hiding (bounded) slotted IPFE scheme which is a computational primitive. An important motivation of [18] is to achieve compact ciphertexts for ABEs. In other words, they get rid of the so-called *one-use restriction* from prior adaptively secure ABEs [16,15,20,17,21,25,9,6,10] by replacing the core information-theoretic step with the computational primitive of function hiding slotted IPFE. The FE of [13] is able to accomplish this property for non-uniform computations by developing a three-slot encryption technique. Specifically, three slots are utilized to simulate the label functions obtained from the underlying AKGS garbling for pre-ciphertext secret keys. Note that, the three-slot encryption technique is an extension of dual system encryption methodologies [24, 16, 15]. In this work, we extend their frameworks [18, 13] to avoid the one-use restriction in the case of FE for UAWS that computes weights via Logspace Turing machines. It is non-trivial to implement such three-slot techniques in the uniform model. The main reason behind this fact is that in case of ABPs [13] the garbling randomness

can be sampled knowing the size of ABPs, and hence the garbling algorithm is possible to run while generating secret keys. However, in the case of AKGS for Logspace Turing machines, the garbling randomness depends on the size of the Turing machine as well as its input lengths. Consequently, it is not possible to execute the garbling in the key generation or encryption algorithms as the information about the garbling randomness is distributed between these two algorithms. We tackle this by developing a more advanced three-slot encryption technique with *distributed randomness* which enables us to carry out such a sophisticated procedure for Logspace Turing machines.

Our FE for UAWS is a one-slot scheme. This means one pair of public-private attribute can be processed in a single encryption. An unbounded-slot FE for UAWS [2] enables us to encrypt unbounded many such pairs in a single encryption. Abdalla et al. [2] devise a generic transformation for bootstrapping from one-slot to unbounded-slot scheme. However, this transformation only works if the underlying one-slot scheme is semi-adaptively secure [13]. Thus, if we restrict our scheme to semi-adaptive security then using such transformations [2,13] our one-slot FE scheme can be bootstrapped to support unbounded slots.

Organization. We discuss a detailed technical overview of our results in Section 2. We provide useful notations, related definitions, and complexity assumptions in Section 3. Our construction of a single key and single ciphertext secure FE scheme for UAWS is described in Section 4. The simulator and security analysis of the scheme can be found in the full version. Next, we build our full fledge 1-slot FE scheme for UAWS in Section 5. The correctness and security analysis of the scheme is available in the full version. For completeness, we present the definition of function-hiding slotted IPFE and the construction of AKGS for Turing machine computations [18] in the full version.

2 Technical Overview

We now present an overview of our techniques for achieving a FE scheme for AWS functionality which supports the uniform model of computations. We consider prime-order bilinear pairing groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, g_1, g_2, e)$ with a generator $g_\mathrm{T} = e(g_1, g_2)$ of $\mathbb{G}_\mathrm{T}$ and denote $[\![a]\!]_i$ by an element $g_i^a \in \mathbb{G}_i$ for $i \in \{1, 2, \mathrm{T}\}$. For any vector $\boldsymbol{z}$, the k-th entry is denoted by $\boldsymbol{z}[k]$ and $[n]$ denotes the set $\{1, \ldots, n\}$.

The unbounded AWS Functionality. In this work, we consider an unbounded FE scheme for the AWS functionality for Logspace Turing machines (or the class of L), in shorthand it is written as $\mathsf{UAWS}^{\mathsf{L}}$. More specifically, the setup only takes input the security parameter of the system and is independent of any other parameter, e.g., the lengths of the public and private attributes. $\mathsf{UAWS}^{\mathsf{L}}$ generates secret keys $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$ for a tuple of Turing machines denoted by $\boldsymbol{M} = \{M_k\}_{k\in\mathcal{I}_{\boldsymbol{M}}}$ such that the index set $\mathcal{I}_{\boldsymbol{M}}$ contains any *arbitrary* number of Turing machines $M_k \in \mathsf{L}$. The ciphertexts are computed for a pair of public-private attributes $(\boldsymbol{x}, \boldsymbol{z})$ whose lengths are *arbitrary* and are decided at the time of encryption. Precisely, the public attribute $\boldsymbol{x}$ of length N comes with a polynomial time bound $T = \mathsf{poly}(N)$ and a logarithmic space bound S, and the private attribute $\boldsymbol{z}$ is an integer vector of length n. At the time of decryption,

if $\mathcal{I}_{\boldsymbol{M}} \subseteq [n]$ then it reveals an integer value $\sum_{k \in \mathcal{I}_{\boldsymbol{M}}} M_k(\boldsymbol{x})\boldsymbol{z}[k]$. Since $M_k(\boldsymbol{x})$ is binary, we observe that the summation selects and adds the entries of $\boldsymbol{z}$ for which the corresponding Turing machine accepts the public attribute $\boldsymbol{x}$. An appealing feature of the functionality is that the secret key $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$ can decrypt ciphertexts of unbounded length attributes in unbounded time/ (logarithmic) space bounds. In contrast, existing FE for AWSs [2, 13] are designed to handle non-uniform computations that can only handle attributes of bounded lengths and the public parameters grows linearly with the lengths. Next, we describe the formulation of Turing machines in L considered in $\mathsf{UAWS}^{\mathsf{L}}$.

Turing machines Formulation. We introduce the notations for Logspace Turning machines (TM) over binary alphabets. A Turing machine $M = (Q, \boldsymbol{y}_{\mathsf{acc}}, \delta)$ consists of Q states with the initial state being 1 and a characteristic vector $\boldsymbol{y}_{\mathsf{acc}} \in \{0,1\}^Q$ of accepting states and a transition function δ. When an input $(\boldsymbol{x}, N, T, S)$ with length N and time, space bounds T, S is provided, the computation of $M|_{N,T,S}(\boldsymbol{x})$ is performed in T steps passing through *configurations* $(\boldsymbol{x}, (i, j, \boldsymbol{W}, q))$ where $i \in [N]$ is the input tape pointer, $j \in [S]$ is the work tape pointer, $\boldsymbol{W} \in \{0,1\}^S$ the content of work tape, and $q \in [Q]$ the state under consideration. The initial *internal* configuration is $(1, 1, \mathbf{0}_S, 1)$ and the transition function δ determines whether, on input $\boldsymbol{x}$, it is possible to move from one internal configuration $(i, j, \boldsymbol{W}, q)$ to the next $((i', j', \boldsymbol{W}', q'))$, namely if $\delta(q, \boldsymbol{x}[i], \boldsymbol{W}[j]) = (q', w', \Delta i, \Delta j)$. In other words, the transition function δ on input state q, an input bit $\boldsymbol{x}[i]$ and an work tape bit $\boldsymbol{W}[j]$, outputs the next state q', the new bit w' overwriting $w = \boldsymbol{W}[j]$ by $w' = \boldsymbol{W}'[j]$ (keeping $\boldsymbol{W}[j''] = \boldsymbol{W}'[j'']$ for all $j \neq j''$), and the directions $\Delta i, \Delta j \in \{0, \pm 1\}$ to move the input and work tape pointers.

Our construction of adaptively simulation secure $\mathsf{UAWS}^{\mathsf{L}}$ depends on two building blocks: AKGS for Logspace Turing machines, an information-theoretic tool and slotted IPFE, a computation tool. We only need a *bounded* slotted IPFE, meaning that the length of vectors of the slotted IPFE is fixed in the setup, and we only require the primitive to satisfy adaptive indistinguishability based security. Hence, our work shows how to (semi-)generically bootstrap a bounded IPFE to an unbounded FE scheme beyond the inner product functionality. Before going to describe the $\mathsf{UAWS}^{\mathsf{L}}$, we briefly discuss about these two building blocks.

AKGS for Logspace Turing machines. In [18], the authors present an ABE scheme for Logspace Turing machines by constructing an efficient AKGS for sequence of matrix multiplications over $\mathbb{Z}_p$. Thus, their core idea was to represent a Turing machine computation through a sequence of matrix multiplications. An internal configuration $(i, j, \boldsymbol{W}, q)$ is represented as a basis vector $\boldsymbol{e}_{(i,j,\boldsymbol{W},q)}$ of dimension $NS2^S Q$ with a single 1 at the position $(i, j, \boldsymbol{W}, q)$. We define a *transition matrix* given by

$$\mathbf{M}(\boldsymbol{x})[(i, j, \boldsymbol{W}, q), (i', j', \boldsymbol{W}', q')] = \begin{cases} 1, & \text{if } \delta(q, \boldsymbol{x}[i], \boldsymbol{W}[j]) = (q', \boldsymbol{W}'[j], i' - i, j' - j) \\ & \quad \text{and } \boldsymbol{W}'[j''] = \boldsymbol{W}[j''] \text{ for all } j'' \neq j; \\ 0, & \text{otherwise;} \end{cases}$$

such that $\boldsymbol{e}^{\top}_{(i,j,\boldsymbol{W},q)}\mathbf{M}(\boldsymbol{x}) = \boldsymbol{e}^{\top}_{(i',j',\boldsymbol{W}',q')}$. This holds because the $((i, j, \boldsymbol{W}, q), (i', j', \boldsymbol{W}', q'))$-th entry of $\mathbf{M}(\boldsymbol{x})$ is 1 if and only if there is a valid transition

from $(q, \boldsymbol{x}[i], \boldsymbol{W}[j])$ to $(q', \boldsymbol{W}'[j], i' - i, j' - j)$. Therefore, one can write the Turing machine computation by right multiplying the matrix $\mathbf{M}(\boldsymbol{x})$ for T times with the initial configuration $\boldsymbol{e}^{\top}_{(1,1,\mathbf{0}_S,1)}$ to reach of one of the final configurations $\mathbf{1}_{[N]\times[S]\times\{0,1\}^S} \otimes \boldsymbol{y}_{\mathsf{acc}}$. In other words, the function $M|_{N,T,S}(\boldsymbol{x})$ is written as

$$M|_{N,T,S}(\boldsymbol{x}) = \boldsymbol{e}^{\top}_{(1,1,\mathbf{0}_S,1)}(\mathbf{M}_{N,S}(\boldsymbol{x}))^T(\mathbf{1}_{[N]\times[S]\times\{0,1\}^S} \otimes \boldsymbol{y}_{\mathsf{acc}}) \tag{2.1}$$

Thus, [18] constructs an AKGS for the the sequence of matrix multiplications as in Equation (2.1). Their AKGS is inspired from the randomized encoding scheme of [5] and homomorphic evaluation procedure of [7]. Given the function $M|_{N,T,S}$ over $\mathbb{Z}_p$ and two secrets z, β, the garbling procedure computes the label functions

$$\begin{aligned} L_{\mathsf{init}}(\boldsymbol{x}) &= \beta + \boldsymbol{e}^{\top}_{(1,1,\mathbf{0}_S,1)}\boldsymbol{r}_0, \\ \text{for } t \in [T]: \quad (L_{t,\theta})_\theta &= -\boldsymbol{r}_{t-1} + \mathbf{M}_{N,S}(\boldsymbol{x})\boldsymbol{r}_t, \\ (L_{T+1,\theta})_\theta &= -\boldsymbol{r}_T + z\mathbf{1}_{[N]\times[S]\times\{0,1\}^S} \otimes \boldsymbol{y}_{\mathsf{acc}}. \end{aligned}$$

and outputs the coefficients of these label functions $\ell_{\mathsf{init}}, \boldsymbol{\ell}_t = (\boldsymbol{\ell}_{t,\theta})_\theta$ where $\theta = (i, j, \boldsymbol{W}, q)$ and $\boldsymbol{r}_t \leftarrow \mathbb{Z}_p^{[N]\times[S]\times\{0,1\}^S\times[Q]}$. To compute the functional value for an input $\boldsymbol{x}$, the evaluation procedure add ℓ_{init} with a telescoping sum $\boldsymbol{e}^{\top}_{(1,1,\mathbf{0}_S,1)} \cdot \sum_{t=1}^{T}(\mathbf{M}_{N,S}(\boldsymbol{x}))^{t-1}\boldsymbol{\ell}_t$ and outputs $zM|_{N,T,S}(\boldsymbol{x}) + \beta$. More precisely, it uses the fact that

$$\boldsymbol{e}^{\top}_{i_{t+1},j_{t+1},\boldsymbol{W}_{t+1},q_{t+1}}\boldsymbol{r}_{t+1} = \boldsymbol{e}^{\top}_{i_t,j_t,\boldsymbol{W}_t,q_t}\boldsymbol{r}_t + \boldsymbol{e}^{\top}_{i_t,j_t,\boldsymbol{W}_t,q_t}(\underbrace{-\boldsymbol{r}_t + \mathbf{M}(\boldsymbol{x})\boldsymbol{r}_{t+1}}_{\boldsymbol{\ell}_{t+1}})$$

A crucial and essential property that the AKGS have is the *linearity* of evaluation procedure, meaning that the procedure is linear in the label function values ℓs and, hence can be performed even if ℓs are available in the exponent of a group. Lin and Luo identify two important security notions of AKGS, jointly called *piecewise security*. Firstly, ℓ_{init} can be reversely sampled given a functional value and all other label values, which is known as the *reverse sampleability*. Secondly, ℓ_t is random with respect to the subsequent label functions $L_{t',\theta}$ for all $t' > t$ and z, which is called the *marginal randomness*.

Function Hiding Slotted IPFE. A normal IPFE computes inner product between two vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ using a secret key $\mathsf{IPFE.SK}_{\boldsymbol{v}}$ and a ciphertext $\mathsf{IPFE.CT}_{\boldsymbol{u}}$. The IPFE is said to satisfy indistinguishability-based security if an adversary having received many functional secret keys $\{\mathsf{IPFE.SK}_{\boldsymbol{v}}\}$ remains incapable to extract any information about the message vector $\boldsymbol{u}$ except the inner products $\{\boldsymbol{v} \cdot \boldsymbol{u}\}$. It is easy to observe that if encryption is done publicly then no security can be ensured about $\boldsymbol{v}$ from the secret key $\mathsf{IPFE.SK}_{\boldsymbol{v}}$ [11] due to the linear functionality. However, if the encryption algorithm is private then $\mathsf{IPFE.SK}_{\boldsymbol{v}}$ can be produced in a fashion to hide sensitive information about $\boldsymbol{v}$. This is termed as *function hiding* security notion for private key IPFE. Slotted IPFE [19] is a hybrid of public and private IPFE where vectors are divided into public and private slots, and function hiding is only guaranteed for the entries in the private slots. Further, Slotted IPFEs of [19, 18] generate secret keys and ciphertexts even when the vectors are given in the exponent of source groups whereas decryption recovers the inner product in the target group.

2.1 From *All-or-Nothing* to *Functional* Encryption

We are all set to describe our approach to extend the framework of [18] from *all-or-nothing* to *functional* encryption for the uniform model of computations. In a previous work of Datta and Pal [13], an adaptively secure FE for AWS functionality was built for the non-uniform model of computations, ABPs to be precise. Their idea was to garble a function $f_k(\boldsymbol{x})\boldsymbol{z}[k]+\beta_k$ during key generation (keeping $\boldsymbol{z}[k]$ and $\boldsymbol{x}$ as variables) and compute IPFE secret keys to encode the m labels, and a ciphertext associated to a tuple $(\boldsymbol{x}, \boldsymbol{z})$ consists of a collection of IPFE ciphertexts which encode the attributes:

$$\mathsf{SK}_{\boldsymbol{f}} = \{\mathsf{IPFE.SK}_{\boldsymbol{v}_{k,t<m}}, \widetilde{\mathsf{IPFE.SK}}_{\widetilde{\boldsymbol{v}}_{k,m}}\}_{k,m} : \quad \begin{array}{c}\boldsymbol{v}_{k,t<m} = \boldsymbol{\ell}_{k,t}, \widetilde{\boldsymbol{v}}_{k,m} = \boldsymbol{\ell}_{k,m} \text{ where} \\ (\boldsymbol{\ell}_{k,t})_t \leftarrow \mathsf{Garble}(f_k(\boldsymbol{x})\boldsymbol{z}[k]+\beta_k) \text{ s.t. } \sum_k \beta_k = 0\end{array}$$

$$\mathsf{CT}_{\boldsymbol{x}} = (\mathsf{IPFE.CT}_{\boldsymbol{u}}, \{\widetilde{\mathsf{IPFE.CT}}_{\widetilde{\boldsymbol{u}}_k}\}_k) : \quad \boldsymbol{u} = (1, \boldsymbol{x}),\ \widetilde{\boldsymbol{u}}_k = (1, \boldsymbol{z}[k])$$

Therefore, using the inner product functionality, decryption computes the actual label values with $\boldsymbol{x}, \boldsymbol{z}[k]$ as inputs and recovers $f_k(\boldsymbol{x})\boldsymbol{z}[k] + \beta_k$ for each k, and hence finally $\sum_k f_k(\boldsymbol{x})\boldsymbol{z}[k]$. However, this approach fails to build $\mathsf{UAWS}^{\mathsf{L}}$ because we can not execute the AKGS garbling for the function $M_k|_{N,T,S}(\boldsymbol{x})\boldsymbol{z}[k] + \beta_k$ at the time of generating keys. More specifically, the garbling randomness depends on parameters N, T, S, n that are unknown to the key generator. Note that, in contrast to the ABE of [18] where $\boldsymbol{z}$ can be viewed as a payload (hence $n = 1$), the UAWS functionality has an additional parameter n (length of $\boldsymbol{z}$) the value of which is chosen at the time of encryption. Moreover, the compactness of $\mathsf{UAWS}^{\mathsf{L}}$ necessitates the secret key size $|\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}| = O(|\mathcal{I}_{\boldsymbol{M}}|Q)$ to be linear in the number of states Q and the ciphertext size $|\mathsf{CT}_{(\boldsymbol{x},T,S)}| = O(nTNS2^S)$ be linear in $TNS2^S$.

The obstacle is circumvented by the randomness distribution technique used in [18]. Instead of computing the AKGS garblings in key generation or encryption phase, the label values are produced by a joint effort of both the secret key and ciphertext. To do so, the garbling is executed under the hood of IPFE using pseudorandomness, instead of true randomness. That is, some part of the garbling randomness is sampled in key generation whereas the rest is sampled in encryption. More specifically, every true random value $\boldsymbol{r}_t[(i, j, \boldsymbol{W}, q)]$ is written as a product $\boldsymbol{r}_{\boldsymbol{x}}[(t, i, j, \boldsymbol{W})]\boldsymbol{r}_{k,f}[q]$ where $\boldsymbol{r}_{\boldsymbol{x}}[(t, i, j, \boldsymbol{W})]$ is used in the ciphertext and $\boldsymbol{r}_{k,f}[q]$ is utilized to encode the transition blocks of M_k in the secret key. To enable this, the transition matrix associated to M_k is represented as follows:

$$\mathbf{M}(\boldsymbol{x})[(i, j, \boldsymbol{W}, q), (i', j', \boldsymbol{W}', q')] = \delta^{(?)}((i, j, \boldsymbol{W}, q), (i', j', \boldsymbol{W}', q')) \times \mathbf{M}_{\boldsymbol{x}[i],\boldsymbol{W}[j],\boldsymbol{W}'[j],i'-i,j'-j}[q, q']$$

where $\delta^{(?)}((i, j, \boldsymbol{W}, q), (i', j', \boldsymbol{W}', q'))$ is 1 if there is a valid transition from the configuration $(i, j, \boldsymbol{W}, q)$ to $(i', j', \boldsymbol{W}', q')$, otherwise 0. Therefore, every block of $\mathbf{M}(\boldsymbol{x})[(i, j, \boldsymbol{W}, q), (i', j', \boldsymbol{W}', q')]$ is either a $Q \times Q$ zero matrix or a *transition block* that belongs to a small set

$$\mathcal{T} = \{\mathbf{M}_\tau |\ \ \tau = (x, w, w', \Delta i, \Delta j) \in \{0, 1\}^3 \times \{0, \pm 1\}^2\}$$

The $(i, j, \boldsymbol{W}, q)$-th *block row* $\mathbf{M}_\tau = \mathbf{M}_{x,w,w',\Delta i,\Delta j}$ appears at $\mathbf{M}_{N,S}(\boldsymbol{x})[(i, j, \boldsymbol{W}, \llcorner), (i', j', \boldsymbol{W}', \llcorner)]$ if $x = \boldsymbol{x}[i], w = \boldsymbol{W}[j], \Delta i = i' - i, \Delta j = j' - j$, and $\boldsymbol{W}'$ is $\boldsymbol{W}$ with j-th entry changed to w'. Thus, every label $\boldsymbol{\ell}_{k,t}[\mathrm{i}, q]$ with $\mathrm{i} = (i, j, \boldsymbol{W})$ can be *decomposed* as inner product $\boldsymbol{v}_{k,q} \cdot \boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}$. More precisely,

$$\begin{aligned}
\boldsymbol{\ell}_{k,t}[\mathrm{i}, q] &= -\boldsymbol{r}_{t-1}[\mathrm{i}, q] + \mathbf{M}_{k,N,S}(\boldsymbol{x})[(\mathrm{i}, q), (\llcorner, \llcorner, \llcorner, \llcorner)]\boldsymbol{r}_t \\
&= -\boldsymbol{r}_{t-1}[\mathrm{i}, q] + \sum_{w',\Delta i,\Delta j} (\mathbf{M}_{k,\boldsymbol{x}[i],\boldsymbol{W}[j],w',\Delta i,\Delta j}\boldsymbol{r}_t[\mathrm{i}', \llcorner])[q] \quad (\mathrm{i}' = (i + \Delta i, j + \Delta j, \boldsymbol{W}')) \\
&= \boldsymbol{r}_{\boldsymbol{x}}[t - 1, \mathrm{i}]\boldsymbol{r}_{k,f}[q] + \sum_{w',\Delta i,\Delta j} \boldsymbol{r}_{\boldsymbol{x}}[t, \mathrm{i}'](\mathbf{M}_{k,\boldsymbol{x}[i],\boldsymbol{W}[j],w',\Delta i,\Delta j}\boldsymbol{r}_{k,f})[q] \\
&= \boldsymbol{r}_{\boldsymbol{x}}[t - 1, \mathrm{i}]\boldsymbol{r}_{k,f}[q] + \sum_{w',\Delta i,\Delta j} \boldsymbol{r}_{\boldsymbol{x}}[t, \mathrm{i}'](\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] = \boldsymbol{v}_{k,q} \cdot \boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}
\end{aligned}$$

so that one can set the vectors

$$\begin{aligned}
\boldsymbol{v}_{k,q} &= (\ \ -\boldsymbol{r}_{k,f}[q],\ \ 0,\ (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] \ \ \|\ \ \mathbf{0}\ ), \\
\boldsymbol{u}_{t,\mathrm{i}} &= (\ \boldsymbol{r}_{\boldsymbol{x}}[t - 1, \mathrm{i}],\ 0,\ \ \ c_\tau(\boldsymbol{x}; \boldsymbol{r}_{\boldsymbol{x}}) \ \ \ \|\ \ \mathbf{0}\ )
\end{aligned}$$

where $c_\tau(\boldsymbol{x}; \boldsymbol{r}_{\boldsymbol{x}})$ (a shorthand of the notation $c_\tau(\boldsymbol{x}, t, i, j, \boldsymbol{W}; \boldsymbol{r}_{\boldsymbol{x}})$ [18]) is given by

$$c_\tau(\boldsymbol{x}; \boldsymbol{r}_{\boldsymbol{x}}) = \begin{cases} \boldsymbol{r}_{\boldsymbol{x}}[t, \mathrm{i}'], & \text{if } x = \boldsymbol{x}[i], w = \boldsymbol{W}[j]; \\ 0, & \text{otherwise.} \end{cases}$$

Similarly, the other labels can be decomposed: $\ell_{k,\mathsf{init}} = (\boldsymbol{r}_{k,f}[1], \beta_k, 0) \cdot (\boldsymbol{r}_{\boldsymbol{x}}[(0, 1, 1, \mathbf{0}_S)], 1, 0) = \beta_k + \boldsymbol{e}^\top_{(1,1,\mathbf{0}_S,1)}\boldsymbol{r}_0$ and $\boldsymbol{\ell}_{k,T+1}[(\mathrm{i}, q)] = \widetilde{\boldsymbol{v}}_{k,q} \cdot \widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}} = -\boldsymbol{r}_T[(\mathrm{i}, q)] + \boldsymbol{z}[k]\boldsymbol{y}_{k,\mathsf{acc}}[q]$ where

$$\begin{aligned}
\widetilde{\boldsymbol{v}}_{k,q} &= (\ -\boldsymbol{r}_{k,f}[q],\ \boldsymbol{y}_{k,\mathsf{acc}}[q] \ \ \|\ \ \mathbf{0}\ ), \\
\widetilde{\boldsymbol{u}}_{T+1,\mathrm{i}} &= (\ \ \boldsymbol{r}_{\boldsymbol{x}}[T, \mathrm{i}],\ \ \ \boldsymbol{z}[k] \ \ \ \|\ \ \mathbf{0}\ )
\end{aligned}$$

A First Attempt. Armed with this, we now present the first candidate $\mathsf{UAWS}^{\mathsf{L}}$ construction in the secret key setting and it supports a single key. We consider two independent master keys imsk and $\widetilde{\mathsf{imsk}}$ of IPFE. For simplicity, we assume the length of private attribute $\boldsymbol{z}$ is the same as the number of Turing machines present in $\boldsymbol{M} = (M_k)_{k \in \mathcal{I}_{\boldsymbol{M}}}$, i.e., $n = |\mathcal{I}_{\boldsymbol{M}}|$. We also assume that each Turing machine in the secret key share the same set of states.

$$\begin{aligned}
\mathsf{SK}_{\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}}} = \{&\mathsf{IPFE.SK}_{\boldsymbol{v}_{k,\mathsf{init}}}, \mathsf{IPFE.SK}_{\boldsymbol{v}_{k,q}}, \widetilde{\mathsf{IPFE.SK}}_{\widetilde{\boldsymbol{v}}_{k,q}}\}_{k \in \mathcal{I}_{\boldsymbol{M}}} : \\
[\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2 &= [\![(\ -\boldsymbol{r}_{k,f}[1],\ \ \beta_k,\ \ \ 0,\ \ \ \|\ \ \mathbf{0}\ )]\!]_2, \\
[\![\boldsymbol{v}_{k,q}]\!]_2 &= [\![(\ -\boldsymbol{r}_{k,f}[q],\ \ 0,\ \ (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] \ \ \|\ \ \mathbf{0}\ )]\!]_2, \\
[\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2 &= [\![(\ -\boldsymbol{r}_{k,f}[q],\ \boldsymbol{y}_{k,\mathsf{acc}}[q] \ \ \|\ \ \mathbf{0}\ )]\!]_2
\end{aligned}$$

$$\begin{aligned}
\mathsf{CT}_{\boldsymbol{x}} = (&\mathsf{IPFE.CT}_{\boldsymbol{u}_{\mathsf{init}}}, \mathsf{IPFE.CT}_{\boldsymbol{u}}, \{\widetilde{\mathsf{IPFE.CT}}_{\widetilde{\boldsymbol{u}}_k}\}_k) : \\
[\![\boldsymbol{u}_{\mathsf{init}}]\!]_1 &= [\![(\ \boldsymbol{r}_{\boldsymbol{x}}[(0, 1, 1, \mathbf{0}_S)],\ \ 1,\ \ \ 0,\ \ \ \|\ \ \mathbf{0}\ )]\!]_1, \\
[\![\boldsymbol{u}_{t<T,\mathrm{i}}]\!]_1 &= [\![(\ \ \ \boldsymbol{r}_{\boldsymbol{x}}[t - 1, \mathrm{i}],\ \ \ 0,\ \ c_\tau(\boldsymbol{x}; \boldsymbol{r}_{\boldsymbol{x}}) \ \ \|\ \ \mathbf{0}\ )]\!]_1, \\
[\![\widetilde{\boldsymbol{u}}_{k,T+1,\mathrm{i}}]\!]_1 &= [\![(\ \ \ \ \boldsymbol{r}_{\boldsymbol{x}}[T, \mathrm{i}],\ \ \ \ \boldsymbol{z}[k] \ \ \ \ \ \|\ \ \mathbf{0}\ )]\!]_1
\end{aligned}$$

Observe that the inner products between the ciphertext and secret key vectors yield the label values $[\![\ell_{k,\mathsf{init}}]\!]_{\mathrm{T}}, [\![\boldsymbol{\ell}_{k,t}]\!]_{\mathrm{T}} = [\![(\boldsymbol{\ell}_{k,t,\theta})_\theta]\!]_{\mathrm{T}}$ for $\theta = (i, j, \boldsymbol{W}, q)$. Now, the evaluation procedure of AKGS is applied to obtain the partial values

$[\![\boldsymbol{z}[k]M_k|_{N,T,S}(\boldsymbol{x}) + \beta_k]\!]_{\mathrm{T}}$. Combining all this values gives the required attribute weighted sum $\sum_k M_k|_{N,T,S}(\boldsymbol{x})\boldsymbol{z}[k]$ Since $\sum_k \beta_k = 0$.

However, this scheme is not *fully* unbounded, in particular, the setup needs to know the length of the private attribute. To realise this, let us try to prove the security of the scheme. The main idea of the proof would be to make all the label values $(\boldsymbol{\ell}_{k,t,\theta})_\theta$ truly random and simulated except the initial labels $\ell_{k,\mathsf{init}}$ so that one can reversely sample $\ell_{k,\mathsf{init}}$ hardcoded with a desired functional value. Suppose, for instance, the single secret key is queried before the challenge ciphertext. In this case, the honest label values are first hardwired in the ciphertext vectors and then the labels are transformed into their simulated version. This is because the ciphertext vectors are computed after the secret key. So, the first step is to hardwire the initial label values $\ell_{k,\mathsf{init}}$ into the ciphertext vector $\boldsymbol{u}_{\mathsf{init}}$ and hence it indicates that the length of $\boldsymbol{u}_{\mathsf{init}}$ must grow with respect to the number of $\ell_{k,\mathsf{init}}$'s. The same situation arises while simulating the other label values through $\boldsymbol{u}_{t,\mathsf{i}}$. In other word, we need to know the size of $\mathcal{I}_{\boldsymbol{M}}$ or the length of $\boldsymbol{z}$ in setup, which is against our desired functionality.

To tackle this, we increase the number of $\boldsymbol{u}_{\mathsf{init}}$ and $\boldsymbol{u}_{t<T,\mathsf{i}}$ in the above system. More specifically, each of these vectors are now computed for all $k \in [n]$, just like $\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}$. Although this fix the requirement of unboundedness of the system, there is another issue related to the security that must be solved. Note that, in the current structure, there is a possibility of *mix-and-match* attack since, for example, $\widetilde{\boldsymbol{u}}_{k_1,T+1,\mathsf{i}}$ can be paired with $\widetilde{\boldsymbol{v}}_{k_2,q}$ and this results in some *unwanted* attribute weighted sum of the form $\sum_{k\neq k_1,k_2} M_k(\boldsymbol{x})\boldsymbol{z}[k] + M_{k_1}(\boldsymbol{x})\boldsymbol{z}[k_2] + M_{k_2}(\boldsymbol{x})\boldsymbol{z}[k_1]$. We employ the *index encoding* technique used in previous works achieving unbounded ABE or IPFE [21,23] to overcome the attack. In particular, we add two extra dimension $\rho_k(-k,1)$ in the ciphertext and $\pi_k(1,k)$ in the secret key for encoding the index k in each of the vectors of the system. Observe that for each Turing machine M_k an independent randomness π_k is sampled. It ensures that an adversary can only recover the desired attribute weighted sum and whenever vectors from different indices are paired only a garbage value is obtained.

Combining the Ideas. After combining the above ideas, we describe our $\mathsf{UAWS}^{\mathsf{L}}$ supporting a single key as follows.

$$\mathsf{SK}_{\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}}} = \{\mathsf{IPFE.SK}_{\boldsymbol{v}_{k,\mathsf{init}}}, \mathsf{IPFE.SK}_{\boldsymbol{v}_{k,q}}, \widetilde{\mathsf{IPFE.SK}}_{\widetilde{\boldsymbol{v}}_{k,q}}\}_{k\in\mathcal{I}_{\boldsymbol{M}}} :$$

$$\begin{aligned}
[\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2 &= [\![(\ \pi_k(1,k),\ -\boldsymbol{r}_{k,f}[1],\quad \beta_k,\quad 0,\quad\ \|\ \mathbf{0}\ )]\!]_2,\\
[\![\boldsymbol{v}_{k,q}]\!]_2 &= [\![(\ \pi_k(1,k),\ -\boldsymbol{r}_{k,f}[q],\quad 0,\quad (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q]\ \|\ \mathbf{0}\ )]\!]_2,\\
[\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2 &= [\![(\ \pi_k(1,k),\ -\boldsymbol{r}_{k,f}[q],\ \boldsymbol{y}_{k,\mathsf{acc}}[q]\qquad\qquad\ \|\ \mathbf{0}\ )]\!]_2
\end{aligned}$$

$$\mathsf{CT}_{\boldsymbol{x}} = \{\mathsf{IPFE.CT}_{\boldsymbol{u}_{k,\mathsf{init}}}, \mathsf{IPFE.CT}_{\boldsymbol{u}_{k,t<T,\mathsf{i}}}, \widetilde{\mathsf{IPFE.CT}}_{\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}}\}_k :$$

$$\begin{aligned}
[\![\boldsymbol{u}_{k,\mathsf{init}}]\!]_1 &= [\![(\ \rho_k(-k,1),\ \boldsymbol{r}_{\boldsymbol{x}}[(0,1,1,\mathbf{0}_S)],\ \ 1,\quad 0,\quad\ \|\ \mathbf{0}\ )]\!]_1,\\
[\![\boldsymbol{u}_{k,t<T,\mathsf{i}}]\!]_1 &= [\![(\ \rho_k(-k,1),\quad \boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}],\quad 0,\ c_\tau(\boldsymbol{x};\boldsymbol{r}_{\boldsymbol{x}})\ \|\ \mathbf{0}\ )]\!]_1,\\
[\![\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}]\!]_1 &= [\![(\ \rho_k(-k,1),\quad\ \boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}],\quad \boldsymbol{z}[k]\qquad\ \|\ \mathbf{0}\ )]\!]_1
\end{aligned}$$

Although the above construction satisfies our desired functionality, preserves the compactness of ciphertexts and resists the aforementioned attack, we face multiple challenges in adapting the proof ideas of previous works [23, 18, 13].

Security Challenges and Solutions. Next, we discuss the challenges in proving the adaptive simulation security of the scheme. Firstly, the unbounded IPFE scheme of Tomida and Takashima [23] is proved in the *indistinguishability-based* model whereas we aim to prove simulation security that is much more challenging. The work closer to ours is the FE for AWS of Datta and Pal [13], but it only supports a *non-uniform* model of computation and the inner product functionality is *bounded*. Moreover, since the garbling randomness is distributed in the secret key and ciphertext vectors, we can not adapt their proof techniques [23, 13] in a straightforward manner. Although the ABE scheme of Lin and Luo [18] handles a uniform model of computation, they only consider *all-or-nothing* type encryptions and hence the adversary is allowed to query secret keys which always fail to decrypt the challenge ciphertext. In contrast, we construct a more advanced encryption mechanism which overcomes all the above constraints of prior works, i.e., our $\mathsf{UAWS}^{\mathsf{L}}$ is an adaptively *simulation* secure *functional encryption scheme* that supports *unbounded* inner product functionality with a *uniform* model of computations over the public attributes.

Our proof technique is inspired by that of [18, 13]. One of the core technical challenges is involved in the case where the secret key is queried before the challenge ciphertext. Thus, we focus more on "sk queried before ct" in this overview. As noted above, in the security analysis of [18] the adversary $\mathcal{A}$ is not allowed to decrypt the challenge ciphertext and hence they completely randomize the ciphertext in the final game. However, since we are building a FE scheme any secret key queried by $\mathcal{A}$ should be able to decrypt the challenge ciphertext. For this, we use the pre-image sampleability technique from prior works [12, 13]. In particular, the reduction samples a dummy vector $\boldsymbol{d} \in \mathbb{Z}_p^n$ satisfying $\sum_k M_k|_{N,T,S}(\boldsymbol{x})\boldsymbol{z}[k] = \sum_k M_k|_{N,T,S}(\boldsymbol{x})\boldsymbol{d}[k]$ where $\boldsymbol{M} = (M_k)_k$ is a pre-challenge secret key. To plant the dummy vector into the ciphertext, we first need to make all label values $\{\ell_{k,t,\mathsf{i},q}\}$ truly random depending on the terms $\boldsymbol{r}_{k,f}[q]\boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}]$'s and then turn them into their simulated forms, and finally traverse in the reverse path to get back the original form of the ciphertext with $\boldsymbol{d}$ taking place of the private attribute $\boldsymbol{z}$. In order to make all these labels truly random, the honest label values are needed to be hardwired into the ciphertext vectors (since these are computed later) so that we can apply the DDH assumption in $\mathbb{G}_1$ to randomize the term $\boldsymbol{r}_{k,f}[q]\boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}]$ (hence the label values). However, this step is much more complicated than [18] since there are two independent IPFE systems in our construction and $\boldsymbol{r}_{k,f}[q]$ appears in both $\boldsymbol{v}_{k,q}$ and $\widetilde{\boldsymbol{v}}_{k,q}$ (i.e., in both the IPFE systems). We design a two-level nested loop running over q and t for relocating $\boldsymbol{r}_{k,f}[q]$ from $\boldsymbol{v}$'s and $\widetilde{\boldsymbol{v}}_{k,q}$ to $\boldsymbol{u}$'s and $\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}$. To this end, we note that the case of "sk queried after ct" is simpler where we embed the reversely sampled initial label values into the secret key. Before going to discuss the hybrids, we first present the *simulator* of the ideal world.

$$
\begin{aligned}
&\mathsf{SK}_{\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}}} = \{\mathsf{IPFE.SK}_{\boldsymbol{v}_{k,\mathsf{init}}}, \mathsf{IPFE.SK}_{\boldsymbol{v}_{k,q}}, \widetilde{\mathsf{IPFE.SK}}_{\widetilde{\boldsymbol{v}}_{k,q}}\}_{k\in\mathcal{I}_{\boldsymbol{M}}} : (\mathsf{sk} \text{ queried before } \mathsf{ct})\\
&\quad [\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2 = [\![(\ \pi_k(1,k),\ -\boldsymbol{r}_{k,f}[1],\quad \beta_k,\quad 0 \quad\|\ \mathbf{0}\ )]\!]_2,\\
&\quad [\![\boldsymbol{v}_{k,q}]\!]_2 = [\![(\ \pi_k(1,k),\ -\boldsymbol{r}_{k,f}[q],\quad 0,\quad (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] \ \|\ \mathbf{0}\ )]\!]_2,\\
&\quad [\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2 = [\![(\ \pi_k(1,k),\ -\boldsymbol{r}_{k,f}[q],\ \boldsymbol{y}_{k,\mathsf{acc}}[q] \quad\|\ \mathbf{0}\ )]\!]_2
\end{aligned}
$$

$$\begin{aligned}
&\mathsf{CT}_{\boldsymbol{x}} = \{\mathsf{IPFE.CT}_{\boldsymbol{u}_{k,\mathsf{init}}}, \mathsf{IPFE.CT}_{\boldsymbol{u}_{k,t<T,\mathsf{i}}}, \widetilde{\mathsf{IPFE.CT}}_{\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}}\}_k :\\
&\quad [\![\boldsymbol{u}_{k,\mathsf{init}}]\!]_1 = [\![(\ \rho_k(-k,1),\ \boldsymbol{r}_{\boldsymbol{x}}[(0,1,1,\mathbf{0}_S)],\ 1,\ 0,\ \|\ 1,\ \mathbf{0}\ )]\!]_1,\\
&\quad [\![\boldsymbol{u}_{k,t<T,\mathsf{i}}]\!]_1 = [\![(\ \rho_k(-k,1),\ \boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}],\ 0,\ c_\tau(\boldsymbol{x};\boldsymbol{r}_{\boldsymbol{x}})\ \|\ \boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}],\ \mathbf{0}\ )]\!]_1,\\
&\quad [\![\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}]\!]_1 = [\![(\ \rho_k(-k,1),\ \boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}],\ \boldsymbol{d}[k]\ \|\ \boldsymbol{s}_{\boldsymbol{x}}[T+1,\mathsf{i}],\mathbf{0}\ )]\!]_1
\end{aligned}$$

$$\begin{aligned}
&\mathsf{SK}_{\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}}} = \{\mathsf{IPFE.SK}_{\boldsymbol{v}_{k,\mathsf{init}}}, \mathsf{IPFE.SK}_{\boldsymbol{v}_{k,q}}, \widetilde{\mathsf{IPFE.SK}}_{\widetilde{\boldsymbol{v}}_{k,q}}\}_{k\in\mathcal{I}_{\boldsymbol{M}}} :\ (\mathsf{sk}\text{ queried after }\mathsf{ct})\\
&\quad [\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2 = [\![(\ \pi_k(1,k),\ 0,\ 0,\ 0\ \|\ \ell_{k,\mathsf{init}},\ \mathbf{0}\ )]\!]_2,\\
&\quad [\![\boldsymbol{v}_{k,q}]\!]_2 = [\![(\ \pi_k(1,k),\ 0,\ 0,\ 0\ \|\ \boldsymbol{s}_{k,f}[q],\mathbf{0}\ )]\!]_2,\\
&\quad [\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2 = [\![(\ \pi_k(1,k),\ 0,\ 0\ \|\ \boldsymbol{s}_{k,f}[q],\mathbf{0}\ )]\!]_2
\end{aligned}$$

where $\ell_{k,\mathsf{init}} \leftarrow \mathsf{RevSamp}((M_k, \boldsymbol{x}, M_k[\boldsymbol{x}]\boldsymbol{z}[k] + \beta_k, \{\ell_{k,t,\mathsf{i},q}\})$ s.t. $\sum_{k\in\mathcal{I}_{\boldsymbol{M}}} \beta_k = 0$ if $\mathcal{I}_{\boldsymbol{M}} \subseteq [n]$; otherwise $\beta_k \leftarrow \mathbb{Z}_p$.

Security Analysis. We use a three-step approach and each step consists of a group of hybrid sequence. At a very high level, we discuss the case of "sk queried before ct". In this overview, for simplicity, we assume that the challenger knows the length of $\boldsymbol{z}$ while it generates the secret key.

First group of Hybrids: The reduction starts with the real scheme. In the first step, the label function $\ell_{k,\mathsf{init}}$ is reversely sampled with the value $M_k[\boldsymbol{x}]\boldsymbol{z}[k] + \beta_k$ which is hardwired in $\boldsymbol{u}_{k,\mathsf{init}}$.

$$\begin{aligned}
\boldsymbol{v}_{k,\mathsf{init}} &= (\ \cdots,\ \boxed{1},\ \boxed{0},\ 0\ \|\ 0,\ \mathbf{0}\ ),\\
\boldsymbol{v}_{k,q} &= (\ \cdots,\ -\boldsymbol{r}_{k,f}[q],\ 0,\ (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q]\ \|\ \boxed{\boldsymbol{s}_{k,f}[q]},\mathbf{0}\ ),\\
\widetilde{\boldsymbol{v}}_{k,q} &= (\ \cdots,\ -\boldsymbol{r}_{k,f}[q],\ \boldsymbol{y}_{k,\mathsf{acc}}[q]\ \|\ 0,\ \mathbf{0}\ )
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{u}_{k,\mathsf{init}} &= (\ \cdots,\ \boxed{\ell_{k,\mathsf{init}}},\ \boxed{0},\ 0,\ \|\ 0,\ \mathbf{0}\ ),\\
\boldsymbol{u}_{k,t<T,\mathsf{i}} &= (\ \cdots,\ \boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}],\ 0,\ c_\tau(\boldsymbol{x};\boldsymbol{r}_{\boldsymbol{x}})\ \|\ 0,\ \mathbf{0}\ ),\\
\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}} &= (\ \cdots,\ \boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}],\ \boldsymbol{z}[k]\ \|\ \boxed{\boldsymbol{s}_{\boldsymbol{x}}[T+1,\mathsf{i}]},\mathbf{0}\ )
\end{aligned}$$

where $\ell_{k,\mathsf{init}} \leftarrow \mathsf{RevSamp}((M_k, \boldsymbol{x}, M_k[\boldsymbol{x}]\boldsymbol{z}[k] + \beta_k, \{\ell_{k,t,\mathsf{i},q}\})$ and $\ell_{k,t,\mathsf{i},q}$'s are computed honestly. Note that, the secret values $\{\beta_k\}$ are sampled depending on whether the queried key is eligible for decryption. More specifically, if $\mathcal{I}_{\boldsymbol{M}} \subseteq [n]$, then β_k's are sampled as in the original key generation algorithm, i.e., $\sum_k \beta_k = 0$. On the other hand, if $\max\mathcal{I}_{\boldsymbol{M}} > n$ then β_k's are sampled uniformly at random, i.e., they do not necessarily be secret shares of zero. This can be done by the function hiding property of IPFE which ensures that the distributions $\{\{\mathsf{IPFE.SK}_{\boldsymbol{v}_k^{(\mathfrak{b})}}\}_{k\in[n+1,|\mathcal{I}_{\boldsymbol{M}}|]}, \{\mathsf{IPFE.CT}_{\boldsymbol{u}_{k'}}\}_{k'\in[n]}\}$ for $\mathfrak{b} \in \{0,1\}$ are indistinguishable where

$$\begin{aligned}
\boldsymbol{v}_k^{(\mathfrak{b})} &= (\ \ \pi_k,\ \ k\cdot\pi_k,\ \mathbf{0},\ \beta_k + \mathfrak{b}\cdot r_k,\ \mathbf{0}\ )\quad \text{for } k \in [n+1, |\mathcal{I}_{\boldsymbol{M}}|], r_k \leftarrow \mathbb{Z}_p\\
\boldsymbol{u}_{k'} &= (\ -k'\cdot\rho_{k'},\ \ \rho_{k'},\ \ \mathbf{0},\ \ 1,\ \ \mathbf{0}\ )\quad \text{for } k' \in [n]
\end{aligned}$$

Thus, the indistinguishability between the group of hybrids can be guaranteed by the piecewise security of AKGS and the function hiding security of IPFE.

Second group of Hybrids: The second step is a loop. The purpose of the loop is to change all the honest label values $\ell_{k,t,\mathsf{i},q}$ to simulated ones that take the

form $\ell_{k,t,\mathsf{i},q} = \boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}]\boldsymbol{s}_{k,f}[q]$ where $\boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}]$ is hardwired in $\boldsymbol{u}_{k,t,\mathsf{i}}$ or $\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}$ and $\boldsymbol{s}_{k,f}[q]$ is hardwired in $\boldsymbol{v}_{k,q}$ or $\widetilde{v}_{k,q}$.

The whole procedure is executed in via a two-level loop with outer loop running over t and inner loop running over q (both in increasing order). In each iteration of the loop, we move all occurrences of $\boldsymbol{r}_{k,f}[q]$ into the $\boldsymbol{u}$'s in one shot and hardwire the honest labels $\ell_{k,t,\mathsf{i},q}$ into $\boldsymbol{u}_{k,t,\mathsf{i}}$ for all i. Below we present two crucial intermediate hybrids of the loop when $t \le T$.

$$\boldsymbol{v}_{k,q} = (\ \cdots\ , -\boxed{\boldsymbol{\times r}_{k,f}[q]}-\ \|\ \boxed{0}, \boxed{1}, \mathbf{0}\ ),$$
$$\widetilde{\boldsymbol{v}}_{k,q} = (\ \cdots\ , \quad -\boxed{0}-\quad \|\ \ 0,\ \boxed{1}, \mathbf{0}\ ),$$

$$\boldsymbol{u}_{k,t<T,\mathsf{i}} = (\ \cdots\ , -\boxed{\boldsymbol{\checkmark r}_{k,f}[q]}-\ \|\ \quad \boxed{\boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}]},\ \boxed{\begin{array}{l}\text{honest } \ell_{k,t,\mathsf{i},q}\\ = -\boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}]\boldsymbol{r}_{k,f}[q] + \cdots\end{array}}, \mathbf{0}\ ),$$

$$\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}} = (\ \cdots\ , \boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}], \boldsymbol{z}[k] \quad \|\ \boldsymbol{s}_{\boldsymbol{x}}[T+1,\mathsf{i}],\ \boxed{\begin{array}{l}\text{honest } \ell_{k,T+1,\mathsf{i},q}\\ = -\boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}]\boldsymbol{r}_{k,f}[q] + \cdots\end{array}},\quad \mathbf{0}\ )$$

where $\boldsymbol{\times r}_{k,f}[q]$ and $\boldsymbol{\checkmark r}_{k,f}[q]$ indicate the presence of $\boldsymbol{r}_{k,f}[q]$ in their respective positions. The indistinguishability can be argued using the function hiding security of IPFE. Next, by invoking DDH in $\mathbb{G}_1$, we first make $\boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}]\boldsymbol{r}_{k,f}[q]$ truly random for all i and then transform the label values into their simulated form $\ell_{k,\mathsf{i},q} = \boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}]\boldsymbol{s}_{k,f}[q]$ again by using DDH in $\mathbb{G}_1$ for all i. We *emphasize* that the labels $\ell_{k,T+1,\mathsf{i},q}$ are kept as honest and hardwired when the loop runs for $t \le T$. Finally, the terms $\boldsymbol{s}_{k,f}[q]$ are shifted back to $\boldsymbol{v}_{k,q}$ or $\widetilde{\boldsymbol{v}}_{k,q}$.

$$\boldsymbol{v}_{k,q} = (\ \cdots\ , \boxed{-\boldsymbol{r}_{k,f}[q]},\quad 0,\quad \boxed{(\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q]}\ \|\ \boxed{\boldsymbol{s}_{k,f}[q]}, \boxed{0}, \mathbf{0}\ ),$$
$$\widetilde{\boldsymbol{v}}_{k,q} = (\ \cdots\ , \boxed{-\boldsymbol{r}_{k,f}[q]}, \boxed{\boldsymbol{y}_{k,\mathsf{acc}}[q]}\ \|\ \quad 0,\quad \boxed{0}, \mathbf{0}\ ),$$

$$\boldsymbol{u}_{k,t<T,\mathsf{i}} = (\ \cdots\ , \quad -\boxed{0}-\quad \|\ \quad \boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}],\quad \boxed{0}, \mathbf{0}\ ),$$
$$\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}} = (\ \cdots\ , \boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}], \boldsymbol{z}[k] \quad \|\ \boldsymbol{s}_{\boldsymbol{x}}[T+1,\mathsf{i}], \boxed{0}, \mathbf{0}\ )$$

After the two-label loop finishes, the reduction run an additional loop over q with t fixed at $T+1$ to make the last few label values $\ell_{k,T+1,\mathsf{i},q}$ simulated. The indistinguishability between the hybrids follows from a similar argument as in the two-level loop.

$$\boldsymbol{v}_{k,q} = (\ \cdots\ , -\boldsymbol{r}_{k,f}[q],\quad 0,\quad (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q]\ \|\ \boldsymbol{s}_{k,f}[q],\ 0, \mathbf{0}\ ),$$
$$\widetilde{\boldsymbol{v}}_{k,q} = (\ \cdots\ , -\boldsymbol{r}_{k,f}[q], \boldsymbol{y}_{k,\mathsf{acc}}[q]\ \|\ \boxed{\boldsymbol{s}_{k,f}[q]}, 0, \mathbf{0}\ ),$$

$$\boldsymbol{u}_{k,t<T,\mathsf{i}} = (\ \cdots\ , \ -0-\ \|\ \quad \boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}],\quad 0, \mathbf{0}\ ),$$
$$\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}} = (\ \cdots\ , -\boxed{0}-\ \|\ \boldsymbol{s}_{\boldsymbol{x}}[T+1,\mathsf{i}], 0, \mathbf{0}\ )$$

Third group of Hybrids: After all the label values $\ell_{k,t,\mathsf{i},q}$ are simulated, the third step uses a few more hybrids to reversely sample $\ell_{1,\mathsf{init}}$ and $\ell_{k,\mathsf{init}}|_{k>1}$ with the hardcoded values $\boldsymbol{M}(\boldsymbol{x})^\top\boldsymbol{z}+\beta_1$ and $\beta_k|_{k>1}$ respectively. This can be achieved through a statistical transformation on $\{\beta_k|\ \sum_k \beta_k = 0\}$. Finally, we are all set

to insert the dummy vector $\boldsymbol{d}$ in place of $\boldsymbol{z}$ keeping $\mathcal{A}$'s view identical.

$$\begin{aligned}
\boldsymbol{v}_{k,\mathsf{init}} &= (\ \cdots, 1, 0, 0 \ \|\ \ 0, \ \ 0, \mathbf{0}\ ),\\
\boldsymbol{v}_{k,q} &= (\ \cdots, -\boxed{0}- \ \|\ \boldsymbol{s}_{k,f}[q], 0, \mathbf{0}\ ),\\
\widetilde{\boldsymbol{v}}_{k,q} &= (\ \cdots, -\boxed{0}- \ \|\ \boldsymbol{s}_{k,f}[q], 0, \mathbf{0}\ ),
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{u}_{k,\mathsf{init}} &= (\ \cdots, \boxed{\ell_{k,\mathsf{init}}}, 0, 0, \ \|\ \ 0, \ \ 0, \mathbf{0}\ ),\\
\boldsymbol{u}_{k,t<T,\mathsf{i}} &= (\ \cdots, \ \ -0- \ \ \|\ \boldsymbol{s}_{\boldsymbol{x}}[t,\mathsf{i}], \ \ 0, \mathbf{0}\ ),\\
\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}} &= (\ \cdots, \ \ -0- \ \ \|\ \boldsymbol{s}_{\boldsymbol{x}}[T+1,\mathsf{i}], 0, \mathbf{0}\ )
\end{aligned}$$

where all the label values $\{\ell_{k,t,\mathsf{i},q}\}$ are simulated and the initial label values are computed as follows

$$\begin{aligned}
\ell_{1,\mathsf{init}} &\leftarrow \mathsf{RevSamp}(M_1, \boldsymbol{x}, \boldsymbol{M}(\boldsymbol{x})^\top \boldsymbol{d} + \beta_1, \{\ell_{k,t,\mathsf{i},q}\}),\\
\ell_{k,\mathsf{init}} &\leftarrow \mathsf{RevSamp}(M_k, \boldsymbol{x}, \beta_k, \{\ell_{k,t,\mathsf{i},q}\}), \quad \text{for all } k > 1
\end{aligned}$$

From this hybrid we can traverse in the reverse direction all the way to the very first hybrid while keeping the private attribute as $\boldsymbol{d}$. We also rearrange the elements using the security of IPFE so that the distribution of the ciphertext does not change with the occurrence of the secret key whether it comes before or after the ciphertext. This is important for the public key $\mathsf{UAWS}^{\mathsf{L}}$. The formal security is discussed in Theorem 4.1.

From Single Key to Full-Fledge $\mathsf{UAWS}^{\mathsf{L}}$. The next and final goal is to bootstrap the single key, single ciphertext secure $\mathsf{UAWS}^{\mathsf{L}}$ to a public key $\mathsf{UAWS}^{\mathsf{L}}$ scheme that supports releasing many secret keys and ciphertexts. Observe that our secret key $\mathsf{UAWS}^{\mathsf{L}}$ already supports multiple keys and single ciphertext. However, it fails to remain secure if two ciphertexts are published. This is because the piecewise security of AKGS can not be guaranteed if the label functions are reused. Our bootstrapping procedure takes inspiration from prior works [18,13], that is to sample a random multiplier $s \leftarrow \mathbb{Z}_p$ at the time of encryption, which will randomize the label values in the exponent of $\mathbb{G}_2$. In particular, using IPFE security the random multiplier s is moved to the secret key vectors where the DDH assumption ensures that $s\boldsymbol{\ell}_{k,t,\mathsf{i},q}$'s are pseudorandom in the exponent of $\mathbb{G}_2$. To upgrade the scheme into public key setting, we employ the Slotted IPFE that enables encrypting into the public slots using the public key whereas the function hiding security still holds in the private slots. We describe below our public key $\mathsf{UAWS}^{\mathsf{L}}$ scheme.

$$\mathsf{SK}_{\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}}} = \{\mathsf{IPFE.SK}_{\boldsymbol{v}_{\mathsf{pad}}}, \mathsf{IPFE.SK}_{\boldsymbol{v}_{k,\mathsf{init}}}, \mathsf{IPFE.SK}_{\boldsymbol{v}_{k,q}}, \widetilde{\mathsf{IPFE.SK}}_{\widetilde{\boldsymbol{v}}_{k,q}}\}_{k\in\mathcal{I}_{\boldsymbol{M}}} : \alpha \leftarrow \mathbb{Z}_p$$

$$\begin{aligned}
[\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2 &= [\![(\ \ 0, \ \ \alpha, \ \ 0, \ \ 0, \ \ 0, \ \ \|\ \mathbf{0}\ )]\!]_2,\\
[\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2 &= [\![(\ \pi_k(1,k),\ 0,\ -\boldsymbol{r}_{k,f}[1],\ \ \beta_k,\ \ 0, \ \ \|\ \mathbf{0}\ )]\!]_2,\\
[\![\boldsymbol{v}_{k,q}]\!]_2 &= [\![(\ \pi_k(1,k),\ 0,\ -\boldsymbol{r}_{k,f}[q],\ \ 0, \ \ (\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] \ \|\ \mathbf{0}\ )]\!]_2,\\
[\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2 &= [\![(\ \pi_k(1,k),\ 0,\ -\boldsymbol{r}_{k,f}[q],\ \alpha\boldsymbol{y}_{k,\mathsf{acc}}[q] \ \ \|\ \mathbf{0}\ )]\!]_2
\end{aligned}$$

$$\mathsf{CT}_{\boldsymbol{x}} = \{\mathsf{IPFE.CT}_{\boldsymbol{u}_{k,\mathsf{init}}}, \mathsf{IPFE.CT}_{\boldsymbol{u}_{k,t<T,\mathsf{i}}}, \widetilde{\mathsf{IPFE.CT}}_{\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}}\}_k : s \leftarrow \mathbb{Z}_p$$

$$\begin{aligned}
[\![\boldsymbol{u}_{\mathsf{pad}}]\!]_1 &= [\![(\ \ 0, \ \ s, \ \ 0, \ \ 0, \ \ 0, \ \ \|\ \perp\ )]\!]_1,\\
[\![\boldsymbol{u}_{k,\mathsf{init}}]\!]_1 &= [\![(\ \rho_k(-k,1),\ 0,\ s\cdot\boldsymbol{r}_{\boldsymbol{x}}[(0,1,1,\mathbf{0}_S)],\ \ s, \ \ 0, \ \ \|\ \perp\ )]\!]_1,\\
[\![\boldsymbol{u}_{k,t<T,\mathsf{i}}]\!]_1 &= [\![(\ \rho_k(-k,1),\ 0,\ \ s\cdot\boldsymbol{r}_{\boldsymbol{x}}[t-1,\mathsf{i}],\ \ 0, \ \ s\cdot c_\tau(\boldsymbol{x};\boldsymbol{r}_{\boldsymbol{x}}) \ \|\ \perp\ )]\!]_1,\\
[\![\widetilde{\boldsymbol{u}}_{k,T+1,\mathsf{i}}]\!]_1 &= [\![(\ \rho_k(-k,1),\ 0,\ \ s\cdot\boldsymbol{r}_{\boldsymbol{x}}[T,\mathsf{i}],\ \ s\cdot\boldsymbol{z}[k] \ \ \|\ \perp\ )]\!]_1
\end{aligned}$$

The slots at the left/right of " ‖ " are public/private. The ciphertexts are computed using only the public slots and the private slots are utilized only in the security analysis. At a very high level, we utilize the triple-slot encryption technique devised in [13] to simulate the pre-challenge secret keys with a dummy vector encoded into the ciphertext and hardwire the functional value into the post-challenge secret keys. As mentioned earlier that the triple-slot encryption technique [13] was devised for non-uniform model which crucially uses the fact that the garbling randomness can be (fully) sampled in the key generation process. It does not hold in our setting. Thus, we design a more advanced three-slot encryption technique that is compatible with *distributed randomness* of AKGS garbling procedure. More specifically, we add one additional hidden subspace in order to realize such sophisticated mechanism for Logspace Turing machines. This additional subspace enables us to simulate the post-ciphertext secret keys with distributed randomness. However, shuttle technical challenges still remain to be overcome due to the structure of AKGS for Logspace Turing machines. We prove the security of the scheme in Theorem 5.1 and provide detailed security analysis in the full version.

3 Preliminaries

In this section, we provide the necessary definitions and backgrounds that will be used in the sequence.

Notations. We denote by λ the security parameter that belongs to the set of natural number $\mathbb{N}$ and 1^λ denotes its unary representation. We use the notation $s \leftarrow S$ to indicate the fact that s is sampled uniformly at random from the finite set S. For a distribution $\mathcal{X}$, we write $x \leftarrow \mathcal{X}$ to denote that x is sampled at random according to distribution $\mathcal{X}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is said to be a negligible function of λ, if for every $c \in \mathbb{N}$ there exists a $\lambda_c \in \mathbb{N}$ such that for all $\lambda > \lambda_c$, $|\mathsf{negl}(\lambda)| < \lambda^{-c}$.

Let Expt be an interactive security experiment played between a challenger and an adversary, which always outputs a single bit. We assume that $\mathsf{Expt}^{\mathsf{C}}_{\mathcal{A}}$ is a function of λ and it is parametrized by an adversary $\mathcal{A}$ and a cryptographic protocol C. Let $\mathsf{Expt}^{\mathsf{C},0}_{\mathcal{A}}$ and $\mathsf{Expt}^{\mathsf{C},1}_{\mathcal{A}}$ be two such experiment. The experiments are computationally/statistically indistinguishable if for any PPT/computationally unbounded adversary $\mathcal{A}$ there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\mathsf{Adv}^{\mathsf{C}}_{\mathcal{A}}(\lambda) = |\Pr[1 \leftarrow \mathsf{Expt}^{\mathsf{C},0}_{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathsf{Expt}^{\mathsf{C},1}_{\mathcal{A}}(1^\lambda)]| < \mathsf{negl}(\lambda)$$

We write $\mathsf{Expt}^{\mathsf{C},0}_{\mathcal{A}} \overset{c}{\approx} \mathsf{Expt}^{\mathsf{C},1}_{\mathcal{A}}$ if they are *computationally indistinguishable* (or simply *indistinguishable*). Similarly, $\mathsf{Expt}^{\mathsf{C},0}_{\mathcal{A}} \overset{s}{\approx} \mathsf{Expt}^{\mathsf{C},1}_{\mathcal{A}}$ means *statistically indistinguishable* and $\mathsf{Expt}^{\mathsf{C},0}_{\mathcal{A}} \equiv \mathsf{Expt}^{\mathsf{C},1}_{\mathcal{A}}$ means they are *identically* distributed.

Sets and Indexing. For $n \in \mathbb{N}$, we denote $[n]$ the set $\{1, 2, \ldots, n\}$ and for $n, m \in \mathbb{N}$ with $n < m$, we denote $[n, m]$ be the set $\{n, n+1, \ldots, m\}$. We use

lowercase boldface, e.g., $\boldsymbol{v}$, to denote column vectors in $\mathbb{Z}_p^n$ and uppercase boldface, e.g., $\mathbf{M}$, to denote matrices in $\mathbb{Z}_p^{n\times m}$ for $p, n, m \in \mathbb{N}$. The i-th component of a vector $\boldsymbol{v} \in \mathbb{Z}_p^n$ is written as $\boldsymbol{v}[i]$ and the (i,j)-th element of a matrix $\mathbf{M} \in \mathbb{Z}_p^{n\times m}$ is denoted by $\mathbf{M}[i,j]$. The transpose of a matrix $\mathbf{M}$ is denoted by $\mathbf{M}^\top$ such that $\mathbf{M}^\top[i,j] = \mathbf{M}[j,i]$. To write a vector of length n with all zero elements, we write $\mathbf{0}_n$ or simply $\mathbf{0}$ when the length is clear from the context. Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_p^n$, then the inner product between the vectors is denoted as $\boldsymbol{u}\cdot\boldsymbol{v} = \boldsymbol{u}^\top\boldsymbol{v} = \sum_{i\in[n]} \boldsymbol{u}[i]\boldsymbol{v}[i] \in \mathbb{Z}_p$. We define *generalized* inner product between two vectors $\boldsymbol{u} \in \mathbb{Z}_p^{\mathcal{I}_1}, \boldsymbol{v} \in \mathbb{Z}_p^{\mathcal{I}_2}$ by $\boldsymbol{u}\cdot\boldsymbol{v} = \sum_{i\in\mathcal{I}_1\cap\mathcal{I}_2} \boldsymbol{u}[i]\boldsymbol{v}[i]$.

Tensor Products. Let $\boldsymbol{u} \in \mathbb{Z}_p^{\mathcal{I}_1}$ and $\boldsymbol{v} \in \mathbb{Z}_p^{\mathcal{I}_2}$ be two vectors, their tensor product $\boldsymbol{w} = \boldsymbol{u}\otimes\boldsymbol{v}$ is a vector in $\mathbb{Z}_p^{\mathcal{I}_1\times\mathcal{I}_2}$ with entries defined by $\boldsymbol{w}[(i,j)] = \boldsymbol{u}[i]\boldsymbol{v}[j]$. For two matrices $\mathbf{M}_1 \in \mathbb{Z}_p^{\mathcal{I}_1\times\mathcal{I}_2}$ and $\mathbf{M}_1 \in \mathbb{Z}_p^{\mathcal{I}_1'\times\mathcal{I}_2'}$,their tensor product $\mathbf{M} = \mathbf{M} = \mathbf{M}_1\otimes\mathbf{M}_2$ is a matrix in $\mathbb{Z}_p^{(\mathcal{I}_1\times\mathcal{I}_1')\times\mathcal{I}_2\times\mathcal{I}_2'}$ with entries defined by $\mathbf{M}[(i_1,i_1'),(i_2,i_2')] = \mathbf{M}_1[i_1,i_2]\mathbf{M}_2[i_1',i_2']$.

Currying. Currying is the product of partially applying a function or specifying part of the indices of a vector/matrices, which yields another function with fewer arguments or another vector/matrix with fewer indices. We use the usual syntax for evaluating a function or indexing into a vector/matrix, except that unspecified variables are represented by "$\_$". For example, let $\mathbf{M} \in \mathbb{Z}_p^{([\mathcal{I}_1]\times[\mathcal{I}_2])\times([\mathcal{J}_1]\times[\mathcal{J}_2])}$ and $i_1 \in \mathcal{I}_1, j_2 \in \mathcal{J}_2$, then $\mathbf{M}[(i_1,\_),(\_,j_2)]$ is a matrix $\mathbf{N} \in \mathbb{Z}_p^{[\mathcal{I}_2]\times[\mathcal{J}_2]}$ such that $\mathbf{N}[i_2,j_1] = \mathbf{M}[(i_1,i_2),(j_1,j_2)]$ for all $i_2 \in \mathcal{I}_2, j_1 \in \mathcal{J}_1$.

Coefficient Vector: Let $f : \mathbb{Z}_p^{\mathcal{I}} \to \mathbb{Z}_p$ be an affine function with coefficient vector $\mathbf{f} \in \mathbb{Z}_p^{\mathcal{S}}$ for $\mathcal{S} = \{\mathsf{const}\} \cup \{\mathsf{coef}_i |\ i \in \mathcal{I}\}$. Then for any $\boldsymbol{x} \in \mathbb{Z}_p^{\mathcal{I}}$, we have $f(\boldsymbol{x}) = \mathbf{f}[\mathsf{const}] + \sum_{i\in\mathcal{I}} \mathbf{f}[\mathsf{coef}_i]\boldsymbol{x}[i]$.

3.1 Bilinear Groups and Hardness Assumptions

We use a pairing group generator $\mathcal{G}$ that takes as input 1^λ and outputs a tuple $\mathsf{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, g_1, g_2, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}$ are groups of prime order $p = p(\lambda)$ and g_i is a generator of the group $\mathbb{G}_i$ for $i \in \{1,2\}$. The map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ satisfies the following properties:

- *bilinear*: $e(g_1^a, g_2^b) = e(g_1,g_2)^{ab}$ for all $a, b \in \mathbb{Z}_p$.
- *non-degenerate*: $e(g_1,g_2)$ generates $\mathbb{G}_\mathrm{T}$.

The group operations in $\mathbb{G}_i$ for $i \in \{1,2,\mathrm{T}\}$ and the map e are efficiently computable in deterministic polynomial time in the security parameter λ. For a matrix $\mathbf{A}$ and each $i \in \{1,2,\mathrm{T}\}$, we use the notation $[\![\mathbf{A}]\!]_i$ to denote $g_i^{\mathbf{A}}$ where the exponentiation is element-wise. The group operation is written additively while using the bracket notation, i.e. $[\![\mathbf{A}+\mathbf{B}]\!]_i = [\![\mathbf{A}]\!]_i + [\![\mathbf{B}]\!]_i$ for matrices $\mathbf{A}$ and $\mathbf{B}$. Observe that, given $\mathbf{A}$ and $[\![\mathbf{B}]\!]_i$, we can efficiently compute $[\![\mathbf{AB}]\!]_i = \mathbf{A}\cdot[\![\mathbf{B}]\!]_i$. We write the pairing operation multiplicatively, i.e. $e([\![\mathbf{A}]\!]_1, [\![\mathbf{B}]\!]_2) = [\![\mathbf{A}]\!]_1[\![\mathbf{B}]\!]_2 = [\![\mathbf{AB}]\!]_\mathrm{T}$.

Assumption 3.1 (Symmetric External Diffie-Hellman Assumption) *We say that the* SXDH *assumption holds in a pairing group* $\mathsf{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, g_1, g_2, e)$ *of order* p*, if the* DDH *assumption holds in* $\mathbb{G}_i$*, i.e.,* $\{[\![a]\!]_i, [\![b]\!]_i, [\![ab]\!]_i\} \approx \{[\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i\}$ *for* $i \in \{1, 2, \mathrm{T}\}$ *and* $a, b, c \leftarrow \mathbb{Z}_p$.

3.2 Turing Machine Formulation

In this subsection, we describe the main computational model of this work, which is Turing machines with a read-only input and a read-write work tape. This type of Turing machines are used to handle decision problems belonging to space-bounded complexity classes such as Logspace predicates. We define below Turing machines with time complexity T and space complexity S. The Turing machine can either accept or reject an input string within this time/space bound. We also stick to the binary alphabet for the shake of simplicity.

Definition 3.1 (Turing machine with time/space bound computation)

[18] A (deterministic) Turing machine over $\{0,1\}$ is a tuple $M = (Q, \boldsymbol{y}_{\mathsf{acc}}, \delta)$, where $Q \geq 1$ is the number of states (we use $[Q]$ as the set of states and 1 as the initial state), $\boldsymbol{y}_{\mathsf{acc}} \in \{0,1\}^Q$ indicates whether each state is accepting, and

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

is the state transition function, which, given the current state q, the symbol x on the input tape under scan, and the symbol w on the work tape under scan, specifies the new state q', the symbol w' overwriting w, the direction Δi to which the input tape pointer moves, and the direction Δj to which the work tape pointer moves. The machine is required to hang (instead of halting) once it reaches on the accepting state, i.e., for all $q \in [Q]$ such that $\boldsymbol{y}_{\mathsf{acc}}[q] = 1$ and all $x, w \in \{0,1\}$, it holds that $\delta(q, x, w) = (q, w, 0, 0)$.

For input length $N \geq 1$ and space complexity bound $S \geq 1$, the set of *internal configurations* of M is

$$\mathcal{C}_{M,N,S} = [N] \times [S] \times \{0,1\}^S \times [Q],$$

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

For any bit-string $\boldsymbol{x} \in \{0,1\}^N$ for $N \geq 1$ and time/space complexity bounds $T, S \geq 1$, the machine M accepts $\boldsymbol{x}$ within time T and space S if there exists a sequence of internal configurations (*computation* path of T steps) $c_0, \ldots, c_T \in \mathcal{C}_{M,N,S}$ with $c_t = (i_t, j_t, \boldsymbol{W}_t, q_t)$ such that

$$\begin{aligned} & i_0 = 1, j_0 = 1, \boldsymbol{W}_0 = \boldsymbol{0}_S, q_0 = 1 (\text{initial configuration}), \\ \text{for } 0 \leq t < T \Bigg\{ & \delta(q_t, \boldsymbol{x}[i_t], \boldsymbol{W}_t[j_t]) = (q_{t+1}, \boldsymbol{W}_{t+1}[j_t], i_{t+1} - i_t, j_{t+1} - j_t), \\ & \boldsymbol{W}_{t+1}[j] = \boldsymbol{W}_t[j] \text{ for all } j \neq j_t \quad (\text{valid transitions}); \\ & \boldsymbol{y}_{\mathsf{acc}}[q_T] = 1 \quad (\text{accepting}). \end{aligned}$$

Denote by $M|_{N,T,S}$ the function $\{0,1\}^N \to \{0,1\}$ mapping $\boldsymbol{x}$ to whether M accepts $\boldsymbol{x}$ in time T and space S. Define $\mathsf{TM} = \{M|\ M \text{ is a Turing machine}\}$ to be the set of all Turing machines.

Note that,the above definition does not allow the Turing machines moving off the input/work tape. For instance, if δ specifies moving the input pointer to the left/right when it is already at the leftmost/rightmost position, there is no valid next internal configuration. This type of situation can be handled by encoding the input string described in [18]. The problem of moving off the work tape to the left can be managed similarly, however, moving off the work tape to the right is undetectable by the machine, and this is intended due to the space bound. That is, when the space bound is violated, the input is *silently* rejected.

3.3 Functional Encryption for Unbounded Attribute-Weighted Sum for Turing machines

We formally present the syntax of FE for unbounded attribute-weighted sum (AWS) and define adaptive simulation security of the primitive. We consider the set of all Turing machines $\mathsf{TM} = \{M|\ M \text{ is a Turing machine}\}$ with time bound T and space bound S.

Definition 3.2 (The AWS Functionality for Turing machines) For any n, $N \in \mathbb{N}$, the class of attribute-weighted sum functionalities is defined as

$$\left\{ ((\boldsymbol{x} \in \{0,1\}^N, 1^T, 1^{2^S}), \boldsymbol{z} \in \mathbb{Z}_p^n) \mapsto \boldsymbol{M}(\boldsymbol{x})^\top \boldsymbol{z} = \sum_{k \in \mathcal{I}_{\boldsymbol{M}}} \boldsymbol{z}[k] \cdot M_k(\boldsymbol{x}) \;\middle|\; \begin{array}{c} N, T, S \geq 1, \\ M_k \in \mathsf{TM}\ \forall k \in [n], \\ \mathcal{I}_{\boldsymbol{M}} \subseteq [n] \text{ with } |\mathcal{I}_{\boldsymbol{M}}| \geq 1 \end{array} \right\}$$

Definition 3.3 (Functional Encryption for Attribute-Weighted Sum) An unbounded-slot FE for unbounded attribute-weighted sum associated to the set of Turing machines TM and the message space $\mathbb{M}$ consists of four PPT algorithms defined as follows:

$\mathbf{Setup}(\mathbf{1}^{\boldsymbol{\lambda}})$: The setup algorithm takes as input a security parameter and outputs the master secret-key MSK and the master public-key MPK.

$\mathbf{KeyGen}(\mathbf{MSK}, (\boldsymbol{M}, \boldsymbol{\mathcal{I}_M}))$: The key generation algorithm takes as input MSK and a tuple of Turing machines $\boldsymbol{M} = (M_k)_{k \in \mathcal{I}_{\boldsymbol{M}}}$. It outputs a secret-key $\mathsf{SK}_{(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})}$ and make $(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})$ available publicly.

$\mathbf{Enc}(\mathbf{MPK}, ((\boldsymbol{x_i}, \mathbf{1}^{\boldsymbol{T_i}}, \mathbf{1}^{\boldsymbol{S_i}}), \boldsymbol{z_i})_{\boldsymbol{i} \in [\mathcal{N}]})$: The encryption algorithm takes as input MPK and a message consisting of $\mathcal{N}$ number of public-private pair of attributes $(\boldsymbol{x}_i, \boldsymbol{z}_i) \in \mathbb{M}$ such that the public attribute $\boldsymbol{x}_i \in \{0,1\}^{N_i}$ for some $N_i \geq 1$ with time and space bounds given by $T_i, S_i \geq 1$, and the private attribute $\boldsymbol{z}_i \in \mathbb{Z}_p^{n_i}$. It outputs a ciphertext $\mathsf{CT}_{(\boldsymbol{x}_i, T_i, S_i)}$ and make $(\boldsymbol{x}_i, T_i, S_i)_{i \in [\mathcal{N}]}$ available publicly.

$\mathbf{Dec}((\mathbf{SK}_{(\boldsymbol{M}, \boldsymbol{\mathcal{I}_M})}, (\boldsymbol{M}, \boldsymbol{\mathcal{I}_M})), (\mathbf{CT}_{(\boldsymbol{x_i}, \boldsymbol{T_i}, \boldsymbol{S_i})}, (\boldsymbol{x_i}, \boldsymbol{T_i}, \boldsymbol{S_i})_{\boldsymbol{i} \in [\boldsymbol{\mathcal{N}}]}))$: The decryption algorithm takes as input $\mathsf{SK}_{(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})}$ along with the tuple of Turing machines and index sets $(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})$, and a ciphertext $\mathsf{CT}_{(\boldsymbol{x}_i, T_i, S_i)}$ along with a collection of associated public attributes $(\boldsymbol{x}_i, T_i, S_i)_{i \in [\mathcal{N}]}$. It outputs a value in $\mathbb{Z}_p$ or $\perp$.

Correctness: The unbounded-slot FE for unbounded attribute-weighted sum is said to be correct if for all $((\boldsymbol{x}_i \in \{0,1\}^{N_i}, 1^{T_i}, 1^{S_i}), \boldsymbol{z}_i \in \mathbb{Z}_p^{n_i})_{i\in[\mathcal{N}]}$ and for all $(\boldsymbol{M} = (M_k)_{k\in\mathcal{I}_{\boldsymbol{M}}}, \mathcal{I}_{\boldsymbol{M}})$, we get

$$\Pr\left[\begin{array}{c}\mathsf{Dec}((\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}, (\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})), (\mathsf{CT}_{(\boldsymbol{x}_i,T_i,S_i)}, (\boldsymbol{x}_i,T_i,S_i)_{i\in[\mathcal{N}]}) = \sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{I}_{\boldsymbol{M}}} M_k(\boldsymbol{x}_i)\boldsymbol{z}_i[k] : \\ (\mathsf{MSK},\mathsf{MPK}) \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})} \leftarrow \mathsf{KeyGen}(\mathsf{MSK},(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})), \\ \mathsf{CT}_{(\boldsymbol{x}_i,T_i,S_i)} \leftarrow \mathsf{Enc}(\mathsf{MPK},((\boldsymbol{x}_i,1^{T_i},1^{S_i}),\boldsymbol{z}_i)_{i\in[\mathcal{N}]}), \mathcal{I}_{\boldsymbol{M}} \subseteq [n_i]\ \forall i \in \mathcal{N}\end{array}\right] = 1$$

We now define the adaptively simulation-based security of FE for unbounded attribute-weighted sum for Turing machines.

Definition 3.4 (Adaptive Simulation Security) Let $(\mathsf{Setup},\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec})$ be an unbounded-slot FE for unbounded attribute-weighted sum for TM and message space $\mathbb{M}$. The scheme is said to be $(\Phi_{\mathsf{pre}},\Phi_{\mathsf{CT}},\Phi_{\mathsf{post}})$-adaptively simulation secure if for any PPT adversary $\mathcal{A}$ making at most Φ_{CT} ciphertext queries and $\Phi_{\mathsf{pre}},\Phi_{\mathsf{post}}$ secret key queries before and after the ciphertext queries respectively, we have $\mathsf{Expt}^{\mathsf{UAWS}}_{\mathcal{A},\mathsf{real}}(1^\lambda) \overset{c}{\approx} \mathsf{Expt}^{\mathsf{UAWS}}_{\mathcal{A},\mathsf{ideal}}(1^\lambda)$, where the experiments are defined as follows. Also, an unbounded-slot FE for attribute-weighted sums is said to be $(\mathsf{poly},\Phi_{\mathsf{CT}},\mathsf{poly})$-adaptively simulation secure if it is $(\Phi_{\mathsf{pre}},\Phi_{\mathsf{CT}},\Phi_{\mathsf{post}})$-adaptively simulation secure as well as Φ_{pre} and Φ_{post} are unbounded polynomials in the security parameter λ.

$\underline{\mathsf{Expt}^{\mathsf{UAWS}}_{\mathcal{A},\mathsf{real}}(1^\lambda)}$
1. $1^{\mathcal{N}} \leftarrow \mathcal{A}(1^\lambda)$;
2. $(\mathsf{MSK},\mathsf{MPK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
3. $(((\boldsymbol{x}_i,1^{T_i},1^{S_i}),\boldsymbol{z}_i \in \mathbb{Z}_p^{n_i})_{i\in[\mathcal{N}]}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}(\mathsf{MSK},\cdot)}}(\mathsf{MPK})$;
4. $\mathsf{CT}_{(\boldsymbol{x}_i,T_i,S_i)} \leftarrow \mathsf{Enc}(\mathsf{MPK},((\boldsymbol{x}_i,1^{T_i},1^{S_i}),\boldsymbol{z}_i)_{i\in[\mathcal{N}]})$;
5. return $\mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}(\mathsf{MSK},\cdot)}}(\mathsf{MPK},\mathsf{CT})$

$\underline{\mathsf{Expt}^{\mathsf{UAWS}}_{\mathcal{A},\mathsf{ideal}}(1^\lambda)}$
1. $1^N \leftarrow \mathcal{A}(1^\lambda)$;
2. $(\mathsf{MSK}^*,\mathsf{MPK}) \leftarrow \mathsf{Setup}^*(1^\lambda,1^N)$;
3. $(((\boldsymbol{x}_i,1^{T_i},1^{S_i}),\boldsymbol{z}_i \in \mathbb{Z}_p^{n_i})_{i\in[\mathcal{N}]}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}_0^*(\mathsf{MSK}^*,\cdot)}}(\mathsf{MPK})$
4. $\mathsf{CT}_{(\boldsymbol{x}_i,T_i,S_i)} \leftarrow \mathsf{Enc}^*(\mathsf{MPK},\mathsf{MSK}^*,(\boldsymbol{x}_i,1^{T_i},1^{S_i},n_i)_{i\in[\mathcal{N}]},\mathcal{V})$;
5. return $\mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}_1^*(\mathsf{MSK}^*,(\boldsymbol{x}_i,1^{T_i},1^{S_i})_{i\in[\mathcal{N}]},\cdot,\cdot)}}(\mathsf{MPK},\mathsf{CT}_{(\boldsymbol{x}_i,T_i,S_i)})$

$\underline{\mathcal{O}_{\mathsf{KeyGen}(\mathsf{MSK},\cdot)}}$
1. input: $(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})$
2. output: $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$

$\underline{\mathcal{O}_{\mathsf{KeyGen}_0^*(\mathsf{MSK}^*,\cdot)}}$
1. input: $(\boldsymbol{M}_\phi,\mathcal{I}_{\boldsymbol{M}_\phi})$ for $\phi \in [\Phi_{\mathsf{pre}}]$
2. output: $\mathsf{SK}_{(\boldsymbol{M}_\phi,\mathcal{I}_{\boldsymbol{M}_\phi})}$

$\underline{\mathsf{Enc}^*(\mathsf{MPK},\mathsf{MSK}^*,(\boldsymbol{x}_i,1^{T_i},1^{S_i},n_i)_{i\in[\mathcal{N}]},\cdot)}$
1. input: $\mathcal{V} = \{(\boldsymbol{M}_\phi,\mathcal{I}_{\boldsymbol{M}_\phi}),\sum_{i\in[\mathcal{N}]}\boldsymbol{M}_\phi(\boldsymbol{x}_i)^\top\boldsymbol{z}_i : \phi\in[\Phi_{\mathsf{pre}}]\}$
2. output: $\mathsf{CT}_{(\boldsymbol{x}_i,T_i,S_i)}$

$\underline{\mathcal{O}_{\mathsf{KeyGen}_1^*(\mathsf{MSK}^*,(\boldsymbol{x}_i^*)_{i\in[N]},\cdot,\cdot)}}$
1. input: $(\boldsymbol{M}_\phi,\mathcal{I}_{\boldsymbol{M}_\phi}),\sum_{i\in\mathcal{N}}\boldsymbol{M}_\phi(\boldsymbol{x}_i)^\top\boldsymbol{z}_i$ for $\phi\in[\Phi_{\mathsf{post}}]$
2. output: $\mathsf{SK}_{(\boldsymbol{M}_\phi,\mathcal{I}_{\boldsymbol{M}_\phi})}$

3.4 Arithmetic Key Garbling Scheme for Turing machines

Lin and Luo [18] introduced arithmetic key garbling scheme (AKGS). The notion of AKGS is an information theoretic primitive, inspired by randomized encodings [5] and partial garbling schemes [14]. It garbles a function $f : \mathbb{Z}_p^n \to \mathbb{Z}_p$ (possibly

of size $(m+1)$) along with two secrets $z, \beta \in \mathbb{Z}_p$ and produces affine label functions $L_1, \ldots, L_{m+1} : \mathbb{Z}_p^n \to \mathbb{Z}_p$. Given f, an input $\boldsymbol{x} \in \mathbb{Z}_p^n$ and the values $L_1(\boldsymbol{x}), \ldots, L_{m+1}(\boldsymbol{x})$, there is an efficient algorithm which computes $zf(\boldsymbol{x}) + \beta$ without revealing any information about z and β. Lin and Luo [18] additionally design AKGS for Turing machines with time/space bounds. Many parts of this section is verbatim to the sections 5 and 7.1 of [18]. Thus, the reader familiar with the notion of AKGS for Turing machines can skip this section. We define AKGS for the function class

$$\mathcal{F} = \{M|_{N,T,S} : \mathbb{Z}_p^N \to \mathbb{Z}_p, N, T, S \geq 1, p \text{ prime}\}$$

for the set of all time/space bounded Turing machine computations. We refer to [18] for a detailed discussion on the computation of Turing machines as a sequence of matrix multiplications, and the construction of AKGS for matrix multiplication.

Definition 3.5 (Arithmetic Key Garbling Scheme (AKGS), [18]) An arithmetic garbling scheme (AKGS) for the function class $\mathcal{F}$, consists of two efficient algorithms:

$\mathsf{Garble}(\boldsymbol{(M, 1^N, 1^T, 1^S, p), z, \beta})$: The garbling is a randomized algorithm that takes as input a tuple of a function $M|_{N,T,S}$ over $\mathbb{Z}_p$ from $\mathcal{F}$, an input length N, a time bound T, a space bound S with $N, T, S \geq 1$, a prime p, and two secret integers $z, \beta \in \mathbb{Z}_p$. It outputs a set of affine functions $L_{\mathsf{init}}, (L_{t,\theta})_{t\in[T+1],\theta\in\mathcal{C}_{M,N,S}} : \mathbb{Z}_p^N \to \mathbb{Z}_p$ which are called label functions that specifies how an input of length N is encoded as labels. Pragmatically, it outputs the coefficient vectors $\boldsymbol{\ell}_{\mathsf{init}}$, $(\boldsymbol{\ell}_{t,\theta})_{t\in[T+1],\theta\in\mathcal{C}_{M,N,S}}$.

$\mathsf{Eval}(\boldsymbol{(M, 1^N, 1^T, 1^S, p), x, \ell_{\mathsf{init}}, (\ell_{t,\theta})_{t\in[T+1],\theta\in\mathcal{C}_{M,N,S}}})$: The evaluation is a deterministic algorithm that takes as input a function $M|_{N,T,S}$ over $\mathbb{Z}_p$ from $\mathcal{F}$, an input vector $\boldsymbol{x} \in \mathbb{Z}_p^N$ and the integers $\ell_{\mathsf{init}}, (\ell_{t,\theta})_{t\in[T+1],\theta\in\mathcal{C}_{M,N,S}} \in \mathbb{Z}_p$ which are supposed to be the values of the label functions at $\boldsymbol{x} \in \mathbb{Z}_p^N$. It outputs a value in $\mathbb{Z}_p$.

Correctness: The AKGS is said to be correct if for all tuple $(M, 1^N, 1^T, 1^S, p)$, integers $z, \beta \in \mathbb{Z}_p$ and $\boldsymbol{x} \in \mathbb{Z}_p^N$, we have

$$\Pr\left[\begin{array}{c}\mathsf{Eval}((M,1^N,1^T,1^S,p),\boldsymbol{x},\ell_{\mathsf{init}},(\ell_{t,\theta})_{t\in[T+1],\theta\in\mathcal{C}_{M,N,S}}) = zM|_{N,T,S}(\boldsymbol{x}) + \beta : \\ (\boldsymbol{\ell}_{\mathsf{init}},(\boldsymbol{\ell}_{t,\theta})_{t\in[T+1],\theta\in\mathcal{C}_{M,N,S}}) \leftarrow \mathsf{Garble}((M,1^N,1^T,1^S,p),z,\beta), \text{where } \ell \leftarrow L(\boldsymbol{x})\end{array}\right] = 1$$

The scheme have *deterministic shape*, meaning that the number of label functions, $m = 1+(T+1)NS2^SQ$, is determined solely by the tuple $(M, 1^N, 1^T, 1^S, p)$, independent of z, β and the randomness in Garble. The number of label functions m is called the *garbling size* of $M|_{N,T,S}$ under this scheme. For the shake of simpler representation, let us number the label values (or functions) as $1, \ldots, m$ in the lexicographical order where the first two label values are $\ell_{\mathsf{init}}, \ell_{(1,1,1,\mathbf{0}_S,1)}$ and the last label value is $\ell_{(T+1,N,S,\mathbf{1}^S,Q)}$.

Linearity: The AKGS is said to be *linear* if the following conditions hold:

- $\mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta)$ uses a uniformly random vector $\boldsymbol{r} \leftarrow \mathbb{Z}_p^m$ as its randomness, where m is determined solely by $(M, 1^N, 1^T, 1^S, p)$, independent of z, β.
- The coefficient vectors $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m$ produced by $\mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta)$ are linear in $(z, \beta, \boldsymbol{r})$.
- $\mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, \ell_1, \ldots, \ell_m)$ is linear in $\ell_1, \ldots, \ell_m$.

For our UAWS, we consider the piecewise security notion of AKGS defined by Lin and Luo [18][4].

Definition 3.6 (Piecewise Security of AKGS, [18]) An AKGS = (Garble, Eval) for the function class $\mathcal{F}$ is *piecewise* secure if the following conditions hold:

- The first label value is *reversely sampleable* from the other labels together with $(M, 1^N, 1^T, 1^S, p)$ and $\boldsymbol{x}$. This reconstruction is perfect even given all the other label functions. Formally, there exists an efficient algorithm RevSamp such that for all $M|_{N,T,S} \in \mathcal{F}, z, \beta \in \mathbb{Z}_p$ and $\boldsymbol{x} \in \mathbb{Z}_p^N$, the following distributions are identical:

$$\left\{ (\ell_1, \boldsymbol{\ell}_2, \ldots, \boldsymbol{\ell}_m) : \begin{array}{l} (\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m) \leftarrow \mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta), \\ \ell_1 \leftarrow L_1(\boldsymbol{x}) \end{array} \right\},$$

$$\left\{ (\ell_1, \boldsymbol{\ell}_2, \ldots, \boldsymbol{\ell}_m) : \begin{array}{l} (\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m) \leftarrow \mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta), \\ \ell_j \leftarrow L_j(\boldsymbol{x}) \text{ for } j \in [2, m], \\ \ell_1 \leftarrow \mathsf{RevSamp}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, zM|_{N,T,S}(\boldsymbol{x}) + \beta, \ell_2, \ldots, \ell_m) \end{array} \right\}$$

- For the other labels, each is *marginally random* even given all the label functions after it. Formally, this means for all $M|_{N,T,S} \in \mathcal{F}, z, \beta \in \mathbb{Z}_p, \boldsymbol{x} \in \mathbb{Z}_p^n$ and all $j \in [2, m]$, the following distributions are identical:

$$\left\{ (\ell_j, \boldsymbol{\ell}_{j+1}, \ldots, \boldsymbol{\ell}_m) : \begin{array}{l} (\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m) \leftarrow \mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta), \\ \ell_j \leftarrow L_j(\boldsymbol{x}) \end{array} \right\},$$

$$\left\{ (\ell_j, \boldsymbol{\ell}_{j+1}, \ldots, \boldsymbol{\ell}_m) : \begin{array}{l} (\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m) \leftarrow \mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta), \\ \ell_j \leftarrow \mathbb{Z}_p \end{array} \right\}$$

We now define special structural properties of AKGS as given in [18], related to the piecewise security of it.

Definition 3.7 (Special Piecewise Security of AKGS, [18]) An AKGS = (Garble, Eval) for a function class $\mathcal{F}$ is *special* piecewise secure if for any $(M, 1^N, 1^T, 1^S, p) \in \mathcal{F}, z, \beta \in \mathbb{Z}_p$ and $\boldsymbol{x} \in \mathbb{Z}_p^N$, it has the following special form:

- The first label value ℓ_1 is always non-zero, i.e., $\mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, 1, 0, \ldots, 0) \neq 0$ where we take $\ell_1 = 1$ and $\ell_j = 0$ for $1 < j \leq m$.

[4] The usual simulation-based security considered in previous works [14, 13] follows from the piecewise security of AKGS.

– Let $\boldsymbol{r} \leftarrow \mathbb{Z}_p^m$ be the randomness used in $\mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta)$. For all $j \in [2, m]$. the label function L_j produced by $\mathsf{Garble}((M, 1^N, 1^T, 1^S, p), z, \beta; \boldsymbol{r})$ can be written as

$$L_j(\boldsymbol{x}) = k_j \boldsymbol{r}[j-1] + L'_j(\boldsymbol{x}; z, \beta, \boldsymbol{r}[j], \boldsymbol{r}[j+1], \ldots, \boldsymbol{r}[m])$$

where $k_j \in \mathbb{Z}_p$ is a non-zero constant (not depending on $\boldsymbol{x}, z, \beta, \boldsymbol{r}$) and L'_j is an affine function of $\boldsymbol{x}$ whose coefficient vector is linear in $(z, \beta, \boldsymbol{r}[j], \boldsymbol{r}[j+1], \ldots, \boldsymbol{r}[m])$. The component $\boldsymbol{r}[j-1]$ is called the randomizer of L_j and ℓ_j.

Lemma 3.1 ([18]) *A special piecewise secure* $\mathsf{AKGS} = (\mathsf{Garble}, \mathsf{Eval})$ *for a function class* $\mathcal{F}$ *is also piecewise secure. The* RevSamp *algorithm (required in piecewise security) obtained for a special piecewise secure* AKGS *is linear in* $\gamma, \ell_2, \ldots,$
ℓ_{m+1} *and perfectly recovers* ℓ_1 *even if the randomness of* Garble *is not uniformly sampled. More specifically, we have the following:*

$$\begin{aligned}
&\mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, \ell_1, \ldots, \ell_m) \\
&= \ell_1 \mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, 1, 0, \ldots, 0) + \mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, 0, \ell_2, \ldots, \ell_m) \qquad (3.1) \\
&\mathsf{RevSamp}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, \gamma, \ell_2, \ldots, \ell_m) \\
&= (\mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, 1, 0, \ldots, 0))^{-1} (\gamma - \mathsf{Eval}((M, 1^N, 1^T, 1^S, p), \boldsymbol{x}, 0, \ell_2, \ldots, \ell_m)) \qquad (3.2)
\end{aligned}$$

Note that, Equation (3.1) follows from the linearity of Eval and Equation (3.2) ensures that RevSamp perfectly computes ℓ_1 (which can be verified by Equation (3.1) with $\gamma = zM|_{N,T,S}(\boldsymbol{x}) + \beta$).

Lemma 3.2 ([18]) *A piecewise secure* $\mathsf{AKGS} = (\mathsf{Garble}, \mathsf{Eval})$ *is also special piecewise secure after an appropriate change of variable for the randomness used by* Garble.

4 (1-SK, 1-CT, 1-Slot)-FE for Unbounded AWS in L

In this section, we build a *secret-key*, 1-slot FE scheme for the *unbounded* attribute-weighted sum functionality in L. At a high level, the scheme satisfies the following properties:

– The setup is *independent* of any parameters, other than the security parameter λ. Specifically, the *length* of vectors and attributes, *number* of Turing machines and their *sizes* are not fixed a-priori during setup. These parameters are flexible and can be chosen at the time of key generation or encryption.
– A secret key is associated with a tuple $(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})$, where $\boldsymbol{M} = (M_k)_{k \in \mathcal{I}_{\boldsymbol{M}}}$ is a tuple of Turing machines with indices k from an index set $\mathcal{I}_{\boldsymbol{M}}$. For each $k \in \mathcal{I}_{\boldsymbol{M}}, M_k \in \mathsf{L}$, i.e., M_k is represented by a deterministic log-space bounded Turing machine (with an arbitrary number of states).
– Each ciphertext encodes a tuple of public-private attributes $(\boldsymbol{x}, \boldsymbol{z})$ of lengths N and n respectively. The runtime T and space bound S for all the machines in $\boldsymbol{M}$ are associated with $\boldsymbol{x}$ which is the input of each machine M_k.
– Finally, decrypting a ciphertext $\mathsf{CT}_{\boldsymbol{x}}$ that encodes $(\boldsymbol{x}, \boldsymbol{z})$ with a secret key $\mathsf{SK}_{\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}}}$ that is tied to $(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})$ reveals the value $\sum_{k \in \mathcal{I}_{\boldsymbol{M}}} \boldsymbol{z}[k] \cdot M_k(\boldsymbol{x})$ whenever $\mathcal{I}_{\boldsymbol{M}} \subseteq [n]$.

We build an FE scheme for the functionality sketched above (also described in Definition 3.2) and prove it to be simulation secure against a *single* ciphertext and secret key query, where the key can be asked either before or after the ciphertext query. Accordingly, we denote the scheme as $\mathsf{SK\text{-}UAWS}^{\mathsf{L}}_{(1,1,1)} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where the index $(1,1,1)$ represents in order the number of secret keys, ciphertexts and slots supported. Below, we list the ingredients for our scheme.

1. $\mathsf{IPFE} = (\mathsf{IPFE.Setup}, \mathsf{IPFE.KeyGen}, \mathsf{IPFE.Enc}, \mathsf{IPFE.Dec})$: a *secret-key*, *function-hiding* IPFE based on G, where $\mathsf{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, g_1, g_2, e)$ is pairing group tuple of prime order p. We can instantiate this from [18].
2. $\mathsf{AKGS} = (\mathsf{Garble}, \mathsf{Eval})$: a special piecewise-secure AKGS for the function class $\mathcal{M} = \{M|_{N,T,S} : \mathbb{Z}_p^N \to \mathbb{Z}_p \mid M \in \mathsf{TM}, N, T, S \geq 1, p \text{ prime}\}$ describing the set of time/space bounded Turing machines. In our construction, the Garble algorithm would run implicitly under the hood of IPFE and thus, it is not invoked directly in the scheme.

We are now ready to describe the $\mathsf{SK\text{-}UAWS}^{\mathsf{L}}_{(1,1,1)} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

Setup(1^λ)**:** On input the security parameter, fix a prime integer $p \in \mathbb{N}$ and define the slots for two IPFE master secret keys as follows:

$$\begin{aligned}\mathcal{S}_{\text{1-UAWS}} &= \{\mathsf{index}_1, \mathsf{index}_2, \mathsf{init}, \mathsf{rand}, \mathsf{rand}^{\mathsf{temp}}, \mathsf{rand}^{\mathsf{comp}}, \mathsf{rand}^{\mathsf{temp,comp}}, \mathsf{acc}, \mathsf{sim}, \mathsf{sim}^{\mathsf{temp}}, \mathsf{sim}^{\mathsf{comp}}\} \\ &\quad \bigcup \{\mathsf{tb}_\tau, \mathsf{tb}_\tau^{\mathsf{temp}}, \mathsf{tb}_\tau^{\mathsf{comp}}, \mathsf{tb}_\tau^{\mathsf{temp,comp}} \mid \tau \in \mathcal{T}\}, \\ \widetilde{\mathcal{S}}_{\text{1-UAWS}} &= \{\mathsf{index}_1, \mathsf{index}_2, \mathsf{init}, \mathsf{rand}, \mathsf{rand}^{\mathsf{temp}}, \mathsf{rand}^{\mathsf{temp,comp}}, \mathsf{acc}, \mathsf{acc}^{\mathsf{temp}}, \mathsf{sim}, \mathsf{sim}^{\mathsf{temp}}\}\end{aligned}$$

Finally, it returns $\mathsf{MSK} = (\mathsf{IPFE.MSK}, \mathsf{IPFE.}\widetilde{\mathsf{MSK}})$.

KeyGen$(\mathsf{MSK}, (\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}}))$**:** On input the master secret key $\mathsf{MSK} = (\mathsf{IPFE.MSK}, \mathsf{IPFE.}\widetilde{\mathsf{MSK}})$ and a function tuple $\boldsymbol{M} = (M_k)_{k \in \mathcal{I}_{\boldsymbol{M}}}$ indexed w.r.t. an index set $\mathcal{I}_{\boldsymbol{M}} \subset \mathbb{N}$ of arbitrary size, parse $M_k = (Q_k, \boldsymbol{y}_k, \delta_k) \in \mathsf{TM}$ $\forall k \in \mathcal{I}_{\boldsymbol{M}}$ and sample the set of elements

$$\left\{\beta_k \leftarrow \mathbb{Z}_p \mid \sum_k \beta_k = 0 \mod p\right\}_{k \in \mathcal{I}_{\boldsymbol{M}}}$$

For all $k \in \mathcal{I}_{\boldsymbol{M}}$, do the following:

1. For $M_k = (Q_k, \boldsymbol{y}_k, \delta_k)$, compute its transition blocks $\mathbf{M}_{k,\tau} \in \{0,1\}^{Q_k \times Q_k}$, $\forall \tau \in \mathcal{T}$.
2. Sample independent random vectors $\boldsymbol{r}_{k,f} \leftarrow \mathbb{Z}_p^{Q_k}$ and a random element $\pi_k \in \mathbb{Z}_p$.
3. For the following vector $\boldsymbol{v}_{k,\mathsf{init}}$, compute a secret key $\mathsf{IPFE.SK}_{k,\mathsf{init}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.MSK}, [\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2)$:

vector	index_1	index_2	init	rand	acc	tb_τ	the other indices
$\boldsymbol{v}_{k,\mathsf{init}}$	π_k	$k \cdot \pi_k$	$\boldsymbol{r}_{k,f}[1]$	0	β_k	0	0

4. For each $q \in [Q_k]$, compute the following secret keys

$$\mathsf{IPFE.SK}_{k,q} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.MSK}, [\![\boldsymbol{v}_{k,q}]\!]_2) \qquad \text{and}$$
$$\widetilde{\mathsf{IPFE.SK}}_{k,q} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.}\widetilde{\mathsf{MSK}}, [\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2),$$

where the vectors $\boldsymbol{v}_{k,q}, \widetilde{\boldsymbol{v}}_{k,q}$ are defined as follows:

vector	index_1	index_2	init	rand	acc	tb_τ	the other indices
$\boldsymbol{v}_{k,q}$	π_k	$k \cdot \pi_k$	0	$-\boldsymbol{r}_{k,f}[q]$	0	$(\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q]$	0

vector	index_1	index_2	rand	acc	the other indices
$\widetilde{\boldsymbol{v}}_{k,q}$	π_k	$k \cdot \pi_k$	$-\boldsymbol{r}_{k,f}[q]$	$\boldsymbol{y}_k[q]$	0

Finally, it returns the secret key as

$$\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})} = \left((\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}}), \left\{\mathsf{IPFE.SK}_{k,\mathsf{init}}, \{\mathsf{IPFE.SK}_{k,q}, \widetilde{\mathsf{IPFE.SK}}_{k,q}\}_{q\in[Q_k]}\right\}_{k\in\mathcal{I}_{\boldsymbol{M}}}\right).$$

Enc$(\mathsf{MSK}, (\boldsymbol{x}, 1^T, 1^{2^S}), \boldsymbol{z})$**:** On input the master secret key $\mathsf{MSK} = (\mathsf{IPFE.MSK}, \widetilde{\mathsf{IPFE.MSK}})$, a public attribute $\boldsymbol{x} \in \{0,1\}^N$ for some arbitrary $N \geq 1$ with time and space complexity bounds given by $T, S \geq 1$ (as $1^T, 1^{2^S}$) respectively, and the private attribute $\boldsymbol{z} \in \mathbb{Z}_p^n$ for some arbitrary $n \geq 1$, it does the following:

1. Sample a random vector $\boldsymbol{r}_{\boldsymbol{x}} \leftarrow \mathbb{Z}_p^{[0,T]\times[N]\times[S]\times\{0,1\}^S}$.
2. For each $k \in [n]$, do the following:
   (a) Sample a random element $\rho_k \leftarrow \mathbb{Z}_p$.
   (b) Compute a ciphertext $\mathsf{IPFE.CT}_{k,\mathsf{init}} \leftarrow \mathsf{IPFE.Enc}(\mathsf{IPFE.MSK}, [\![\boldsymbol{u}_{k,\mathsf{init}}]\!]_1)$ for the vector $\boldsymbol{u}_{k,\mathsf{init}}$:

vector	index_1	index_2	init	rand	acc	tb_τ	the other indices
$\boldsymbol{u}_{k,\mathsf{init}}$	$-k \cdot \rho_k$	ρ_k	$\boldsymbol{r}_{\boldsymbol{x}}[(0,1,1,\mathbf{0}_S)]$	0	1	0	0

   (c) For all $t \in [T], i \in [N], j \in [S], \boldsymbol{W} \in \{0,1\}^S$, do the following:
      (i) Compute the transition coefficients $c_\tau(\boldsymbol{x}; t, i, j, \boldsymbol{W}; \boldsymbol{r}_{\boldsymbol{x}}), \forall \tau \in \mathcal{T}$ using $\boldsymbol{r}_{\boldsymbol{x}}$.
      (ii) Compute the ciphertext $\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}} \leftarrow \mathsf{IPFE.Enc}(\mathsf{IPFE.MSK}, [\![\boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}]\!]_1)$ for the vector $\boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}$:

vector	index_1	index_2	init	rand	acc	tb_τ	the other indices
$\boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}$	$-k \cdot \rho_k$	ρ_k	0	$\boldsymbol{r}_{\boldsymbol{x}}[(t-1,i,j,\boldsymbol{W})]$	0	$c_\tau(\boldsymbol{x}; t, i, j, \boldsymbol{W}; \boldsymbol{r}_{\boldsymbol{x}})$	0

   (d) For $t = T+1$, compute the ciphertext $\widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}} \leftarrow \widetilde{\mathsf{IPFE.Enc}}(\widetilde{\mathsf{IPFE.MSK}}, [\![\widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}]\!]_1)$ for the vector $\widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}$:

vector	index_1	index_2	rand	acc	the other indices
$\widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}$	$-k \cdot \rho_k$	ρ_k	$\boldsymbol{r}_{\boldsymbol{x}}[(T,i,j,\boldsymbol{W})]$	$\boldsymbol{z}[k]$	0

3. Finally, it returns the ciphertext as

$$\mathsf{CT}_{(\boldsymbol{x},T,S)} = \Bigg((\boldsymbol{x},T,S), \Big\{\mathsf{IPFE.CT}_{k,\mathsf{init}}, \{\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}\}_{t\in[T]}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}}\Big\}_{k\in[n],i\in[N],j\in[S],\boldsymbol{W}\in\{0,1\}^S}\Bigg).$$

Dec$(\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}, \mathsf{CT}_{(\boldsymbol{x},T,S)})$**:** On input a secret key $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$ and a ciphertext $\mathsf{CT}_{(\boldsymbol{x},T,S)}$, do the following:

1. Parse $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$ and $\mathsf{CT}_{(\boldsymbol{x},T,S)}$ as follows:

$$\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})} = \left(((M_k)_{k\in\mathcal{I}_{\boldsymbol{M}}}, \mathcal{I}_{\boldsymbol{M}}), \left\{\mathsf{IPFE.SK}_{k,\mathsf{init}}, \{\mathsf{IPFE.SK}_{k,q}, \widetilde{\mathsf{IPFE.SK}}_{k,q}\}_{q\in[Q_k]}\right\}_{k\in\mathcal{I}_{\boldsymbol{M}}}\right),$$
$$M_k = (Q_k, \boldsymbol{y}_k, \delta_k),$$
$$\mathsf{CT}_{(\boldsymbol{x},T,S)} = \Bigg((\boldsymbol{x},T,S), \Big\{\mathsf{IPFE.CT}_{k,\mathsf{init}}, \{\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}\}_{t\in[T]}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}}\Big\}_{k\in[n],i\in[N],j\in[S],\boldsymbol{W}\in\{0,1\}^S}\Bigg), \boldsymbol{x} \in \{0,1\}^N.$$

2. Output $\perp$, if $\mathcal{I}_{\boldsymbol{M}} \not\subseteq [n]$. Else, select the sequence of ciphertexts for the indices $k \in \mathcal{I}_{\boldsymbol{M}}$ as

$$\mathsf{CT}_{(\boldsymbol{x},T,S)} = \Big((\boldsymbol{x}, T, S), \Big\{ \mathsf{IPFE.CT}_{k,\mathsf{init}}, \{\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}\}_{t\in[T]}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}} \Big\}_{k\in\mathcal{I}_{\boldsymbol{M}}, i\in[N], j\in[S], \boldsymbol{W}\in\{0,1\}^S} \Big)$$

3. Recall that $\forall k \in \mathcal{I}_{\boldsymbol{M}}, \mathcal{C}_{M_k,N,S} = [N] \times [S] \times \{0,1\}^S \times [Q_k]$, and that we denote any element in it as $\theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S}$ where the only component in the tuple θ_k depending on k is $q \in [Q_k]$[5]. Invoke the IPFE decryption to compute all label values as:

$$\forall k \in \mathcal{I}_{\boldsymbol{M}} : [\![\ell_{k,\mathsf{init}}]\!]_{\mathrm{T}} = \mathsf{IPFE.Dec}(\mathsf{IPFE.SK}_{k,\mathsf{init}}, \mathsf{IPFE.CT}_{k,\mathsf{init}})$$

$$\forall k \in \mathcal{I}_{\boldsymbol{M}}, t \in [T], \theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S} : [\![\ell_{k,t,\theta_k}]\!]_{\mathrm{T}} = \mathsf{IPFE.Dec}(\mathsf{IPFE.SK}_{k,q}, \mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}})$$

$$\forall k \in \mathcal{I}_{\boldsymbol{M}}, \theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S} : [\![\ell_{k,T+1,\theta_k}]\!]_{\mathrm{T}} = \mathsf{IPFE.Dec}(\widetilde{\mathsf{IPFE.SK}}_{k,q}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}})$$

4. Next, invoke the AKGS evaluation and obtain the combined value

$$[\![\mu]\!]_{\mathrm{T}} = \prod_{k\in\mathcal{I}_{\boldsymbol{M}}} \mathsf{Eval}\left(\left(M_k, 1^N, 1^T, 1^{2^S}, p \right), \boldsymbol{x}, [\![\ell_{k,\mathsf{init}}]\!]_{\mathrm{T}}, \left\{ [\![\ell_{k,t,\theta_k}]\!]_{\mathrm{T}} \right\}_{t\in[T+1], \theta_k\in\mathcal{C}_{M_k,N,S}} \right)$$

5. Finally, it returns $\mu = \mathsf{DLog}_{g_{\mathrm{T}}}([\![\mu]\!]_{\mathrm{T}})$, where $g_{\mathrm{T}} = e(g_1, g_2)$. Similar to [2], we assume that the desired attribute-weighted sum lies within a specified polynomial-sized domain so that discrete logarithm can be solved via brute-force.

Correctness: Correctness follows from that of IPFE and AKGS. The first step is to observe that all the AKGS label values are correctly computed as functions of the input $\boldsymbol{x}$. This holds by the correctness of IPFE and AKGS encoding of the iterated matrix-vector product representing any TM computation. The next (and final) correctness follows from the linearity of $\mathsf{AKGS.Eval}$.

In more detail, for all $k \in \mathcal{I}_{\boldsymbol{M}}, \theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S}$, let $L_{k,\mathsf{init}}, L_{k,t,\theta_k}$ be the label functions corresponding to the AKGS garbling of $M_k = (Q_k, \boldsymbol{y}_k, \delta_k)$. By the definitions of vectors $\boldsymbol{v}_{k,\mathsf{init}}, \boldsymbol{u}_{\mathsf{init}}$ and the correctness of IPFE, we have

$$\begin{aligned}\ell_{k,\mathsf{init}} &= (-k\rho_k\pi_k + k\pi_k\rho_k) + \boldsymbol{r}_{\boldsymbol{x}}[(0, 1, 1, \mathbf{0}_S)]\boldsymbol{r}_{k,f}[1] + \beta_k \\ &= \boldsymbol{r}_0[(1, 1, \mathbf{0}_S, 1)] + \beta_k = \boldsymbol{e}^T_{(1,1,\mathbf{0}_S,1)}\boldsymbol{r}_0 + \beta_k = L_{k,\mathsf{init}}(\boldsymbol{x}).\end{aligned}$$

Next, $\forall k \in \mathcal{I}_{\boldsymbol{M}}, t \in [T], q \in [Q_k]$, the structures of $\boldsymbol{v}_{k,q}, \boldsymbol{u}_{t,i,j,\boldsymbol{W}}$ and the correctness of IPFE yields

$$\begin{aligned}\ell_{k,t,i,j,\boldsymbol{W},q} &= (-k\rho_k\pi_k + k\pi_k\rho_k) - \boldsymbol{r}_{\boldsymbol{x}}[(t-1, i, j, \boldsymbol{W})]\boldsymbol{r}_{k,f}[q] + \sum_{\tau\in\mathcal{T}} c_\tau(\boldsymbol{x}; t, i, j, \boldsymbol{W}; \boldsymbol{r}_{\boldsymbol{x}})(\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] \\ &= -\boldsymbol{r}_{t-1}[(i, j, \boldsymbol{W}, q)] + \sum_{\tau\in\mathcal{T}} c_\tau(\boldsymbol{x}; t, i, j, \boldsymbol{W}; \boldsymbol{r}_{\boldsymbol{x}})(\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q] = L_{k,t,i,j,\boldsymbol{W},q}(\boldsymbol{x})\end{aligned}$$

[5] For simplicity of notations, we enumerate the states of each M_k as $1, \ldots, q$, i.e., $[Q_k] = [Q]$ for some $Q \in \mathbb{N}$.

Finally, $\forall k \in \mathcal{I}_{\boldsymbol{M}}, q \in [Q_k]$, the vectors $\widetilde{\boldsymbol{v}}_{k,q}, \widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}$ and the $\widetilde{\mathsf{IPFE}}$ correctness again yields

$$\begin{aligned}\ell_{k,T+1,i,j,\boldsymbol{W},q} &= (-k\rho_k\pi_k + k\pi_k\rho_k) - \boldsymbol{r}_{\boldsymbol{x}}[(T,i,j,\boldsymbol{W})]\boldsymbol{r}_{k,f}[q] + \boldsymbol{z}[k]\boldsymbol{y}_k[q] \\ &= -\boldsymbol{r}_T[(i,j,\boldsymbol{W},q)] + \boldsymbol{z}[k]\left(1_{[N]\times[S]\times\{0,1\}^S} \otimes \boldsymbol{y}_k\right)[(i,j,\boldsymbol{W},q)] \\ &= L_{k,T+1,i,j,\boldsymbol{W},q}(\boldsymbol{x}).\end{aligned}$$

The above label values are computed in the exponent of the target group $\mathbb{G}_{\mathrm{T}}$. Once all these are generated correctly, the linearity of Eval implies that the garbling can be evaluated in the exponent of $\mathbb{G}_{\mathrm{T}}$. Thus, this yields

$$\begin{aligned}[\![\mu]\!]_{\mathrm{T}} &= \prod_{k\in\mathcal{I}_{\boldsymbol{M}}} \mathsf{Eval}\left(\left(M_k, 1^N, 1^T, 1^{2^S}, p\right), \boldsymbol{x}, [\![\ell_{k,\mathsf{init}}]\!]_{\mathrm{T}}, \left\{[\![\ell_{k,t,\theta_k}]\!]_{\mathrm{T}}\right\}_{t\in[T+1],\theta_k\in\mathcal{C}_{M_k,N,S}}\right) \\ &= [\![\sum_{k\in\mathcal{I}_{\boldsymbol{M}}} \mathsf{Eval}((M_k, 1^N, 1^T, 1^{2^S}, p), \boldsymbol{x}, \ell_{k,\mathsf{init}}, \{\ell_{k,t,\theta_k}\}_{t\in[T+1],\theta_k\in\mathcal{C}_{M_k,N,S}})]\!]_{\mathrm{T}} \\ &= [\![\sum_{k\in\mathcal{I}_{\boldsymbol{M}}} (\boldsymbol{z}[k]\cdot M_k|_{N,T,S}(\boldsymbol{x}) + \beta_k)]\!]_{\mathrm{T}} = [\![\sum_{k\in\mathcal{I}_{\boldsymbol{M}}} \boldsymbol{z}[k]\cdot M_k|_{N,T,S}(\boldsymbol{x})]\!]_{\mathrm{T}} = [\![\boldsymbol{M}(\boldsymbol{x})^\top \boldsymbol{z}]\!]_{\mathrm{T}}\end{aligned}$$

Theorem 4.1 *Assuming the* SXDH *assumption holds in* $\mathcal{G}$ *and the* IPFE *is function hiding secure, the above construction of* (1-SK, 1-CT, 1-*Slot*)-*FE for* UAWS *is adaptively simulation secure.*

The security analysis is provided in the full version.

5 1-Slot FE for Unbounded AWS for L

In this section, we construct a *public key* 1-slot FE scheme for the *unbounded* attribute-weighted sum functionality for L. The scheme satisfies the same properties as of the $\mathsf{SK\text{-}UAWS}^{\mathsf{L}}_{(1,1,1)}$. However, the *public key* scheme supports releasing polynomially many secret keys and a single challenge ciphertext, hence we denote the scheme as $\mathsf{PK\text{-}UAWS}^{\mathsf{L}}_{(\mathsf{poly},1,1)}$.

Along with the AKGS for Logspace Turing machines we require a *function-hiding slotted* $\mathsf{IPFE} = (\mathsf{IPFE.Setup}, \mathsf{IPFE.KeyGen}, \mathsf{IPFE.Enc}, \mathsf{IPFE.SlotEnc}, \mathsf{IPFE.Dec})$ based on G, where $\mathsf{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, g_1, g_2, e)$ is pairing group tuple of prime order p. We now describe the $\mathsf{PK\text{-}UAWS}^{\mathsf{L}}_{(\mathsf{poly},1,1)} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

Setup(1^λ)**:** On input the security parameter, fix a prime integer $p \in \mathbb{N}$ and define the slots for generating two pair of IPFE master keys as follows:

$$\begin{aligned}\mathcal{S}_{\mathsf{pub}} &= \left\{\mathsf{index}_1, \mathsf{index}_2, \mathsf{pad}, \mathsf{init}^{\mathsf{pub}}, \mathsf{rand}^{\mathsf{pub}}, \mathsf{acc}^{\mathsf{pub}}\right\} \cup \{\mathsf{tb}^{\mathsf{pub}}_\tau | \tau \in \mathcal{T}\}, \\ \mathcal{S}_{\mathsf{copy}} &= \{\mathsf{init}^{\mathsf{copy}}, \mathsf{rand}^{\mathsf{copy}}\} \cup \{\mathsf{tb}^{\mathsf{copy}}_\tau | \tau \in \mathcal{T}\}, \\ \mathcal{S}_{\mathsf{priv}} &= \mathcal{S}_{\mathsf{copy}} \cup \mathcal{S}_{\text{1-UAWS}} \cup \{\mathsf{pad}^{\mathsf{copy}}, \mathsf{pad}^{\mathsf{temp}}, \mathsf{acc}^{\mathsf{perm}}, \mathsf{sim}^{\mathsf{copy}}\}, \\ \widetilde{\mathcal{S}}_{\mathsf{pub}} &= \{\mathsf{index}_1, \mathsf{index}_2, \mathsf{rand}^{\mathsf{pub}}, \mathsf{acc}^{\mathsf{pub}}\}, \\ \widetilde{\mathcal{S}}_{1,\mathsf{copy}} &= \{\mathsf{rand}^{\mathsf{copy}}_1, \mathsf{acc}^{\mathsf{copy}}_1\}, \widetilde{\mathcal{S}}_{2,\mathsf{copy}} = \{\mathsf{rand}^{\mathsf{copy}}_2, \mathsf{acc}^{\mathsf{copy}}_2\}, \\ \widetilde{\mathcal{S}}_{\mathsf{priv}} &= \widetilde{\mathcal{S}}_{1,\mathsf{copy}} \cup \widetilde{\mathcal{S}}_{2,\mathsf{copy}} \cup \widetilde{S}_{\text{1-UAWS}} \cup \{\mathsf{sim}^{\mathsf{copy}}\}\end{aligned}$$

It generates $(\mathsf{IPFE.MPK}, \mathsf{IPFE.MSK}) \leftarrow \mathsf{IPFE.Setup}(\mathcal{S}_{\mathsf{pub}}, \mathcal{S}_{\mathsf{priv}})$ and $(\mathsf{IPFE.}\widetilde{\mathsf{MPK}}, \mathsf{IPFE.}\widetilde{\mathsf{MSK}}) \leftarrow \mathsf{IPFE.Setup}(\widetilde{\mathcal{S}}_{\mathsf{pub}}, \widetilde{\mathcal{S}}_{\mathsf{priv}})$ and returns $\mathsf{MSK} = (\mathsf{IPFE.MSK}, \mathsf{IPFE.}\widetilde{\mathsf{MSK}})$ and $\mathsf{MPK} = (\mathsf{IPFE.MPK}, \mathsf{IPFE.}\widetilde{\mathsf{MPK}})$.

KeyGen$(\mathsf{MSK}, (\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}}))$**:** On input the master secret key $\mathsf{MSK} = (\mathsf{IPFE.MSK}, \mathsf{IPFE.}\widetilde{\mathsf{MSK}})$ and a function tuple $\boldsymbol{M} = (M_k)_{k \in \mathcal{I}_{\boldsymbol{M}}}$ indexed w.r.t. an index set $\mathcal{I}_{\boldsymbol{M}} \subset \mathbb{N}$ of arbitrary size, it parses $M_k = (Q_k, \boldsymbol{y}_k, \delta_k) \in \mathsf{TM}\ \forall k \in \mathcal{I}_{\boldsymbol{M}}$ and samples the set of elements

$$\left\{ \alpha, \beta_k \leftarrow \mathbb{Z}_p \mid k \in \mathcal{I}_{\boldsymbol{M}}, \sum_k \beta_k = 0 \mod p \right\}.$$

It computes a secret key $\mathsf{IPFE.SK}_{\mathsf{pad}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.MSK}, [\![\boldsymbol{v}_{\mathsf{pad}}]\!]_2)$ for the following vector $\boldsymbol{v}_{\mathsf{pad}}$:

vector	index_1	index_2	pad	$\mathsf{init}^{\mathsf{pub}}$	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	$\mathsf{tb}_\tau^{\mathsf{pub}}$	in $\mathcal{S}_{\mathsf{priv}}$
$\boldsymbol{v}_{\mathsf{pad}}$	0	0	α	0	0	0	0	0

For all $k \in \mathcal{I}_{\boldsymbol{M}}$, do the following:

1. For $M_k = (Q_k, \boldsymbol{y}_k, \delta_k)$, compute transition blocks $\mathbf{M}_{k,\tau} \in \{0,1\}^{Q_k \times Q_k}$, $\forall \tau \in \mathcal{T}_k$.
2. Sample independent random vector $\boldsymbol{r}_{k,f} \leftarrow \mathbb{Z}_p^{Q_k}$ and a random element $\pi_k \in \mathbb{Z}_p$.
3. For the following vector $\boldsymbol{v}_{k,\mathsf{init}}$, compute a secret key $\mathsf{IPFE.SK}_{k,\mathsf{init}} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.MSK}, [\![\boldsymbol{v}_{k,\mathsf{init}}]\!]_2)$:

vector	index_1	index_2	pad	$\mathsf{init}^{\mathsf{pub}}$	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	$\mathsf{tb}_\tau^{\mathsf{pub}}$	in $\mathcal{S}_{\mathsf{priv}}$
$\boldsymbol{v}_{k,\mathsf{init}}$	π_k	$k \cdot \pi_k$	0	$\boldsymbol{r}_{k,f}[1]$	0	β_k	0	0

4. For each $q \in [Q_k]$, compute the following secret keys

$$\mathsf{IPFE.SK}_{k,q} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.MSK}, [\![\boldsymbol{v}_{k,q}]\!]_2) \quad \text{and}$$
$$\mathsf{IPFE.}\widetilde{\mathsf{SK}}_{k,q} \leftarrow \mathsf{IPFE.KeyGen}(\mathsf{IPFE.}\widetilde{\mathsf{MSK}}, [\![\widetilde{\boldsymbol{v}}_{k,q}]\!]_2)$$

where the vectors $\boldsymbol{v}_{k,q}, \widetilde{\boldsymbol{v}}_{k,q}$ are defined as follows:

vector	index_1	index_2	pad	$\mathsf{init}^{\mathsf{pub}}$	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	$\mathsf{tb}_\tau^{\mathsf{pub}}$	in $\mathcal{S}_{\mathsf{priv}}$
$\boldsymbol{v}_{k,q}$	π_k	$k \cdot \pi_k$	0	0	$-\boldsymbol{r}_{k,f}[q]$	0	$(\mathbf{M}_{k,\tau}\boldsymbol{r}_{k,f})[q]$	0

vector	index_1	index_2	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	in $\widetilde{\mathcal{S}}_{\mathsf{priv}}$
$\widetilde{\boldsymbol{v}}_{k,q}$	k	$k \cdot \pi_k$	$-\boldsymbol{r}_{k,f}[q]$	$\alpha \cdot \boldsymbol{y}_k[q]$	0

Finally, it returns the secret key as

$$\mathsf{SK}_{(\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}})} = \left((\boldsymbol{M}, \mathcal{I}_{\boldsymbol{M}}), \mathsf{IPFE.SK}_{\mathsf{pad}}, \left\{ \mathsf{IPFE.SK}_{k,\mathsf{init}}, \left\{ \mathsf{IPFE.SK}_{k,q}, \mathsf{IPFE.}\widetilde{\mathsf{SK}}_{k,q} \right\}_{q \in [Q_k]} \right\}_{k \in \mathcal{I}_{\boldsymbol{M}}} \right).$$

Enc$(\mathsf{MPK}, (\boldsymbol{x}, 1^T, 1^{2^S}), \boldsymbol{z})$**:** On input the master public key $\mathsf{MPK} = (\mathsf{IPFE.MPK}, \mathsf{IPFE.}\widetilde{\mathsf{MPK}})$, a public attribute $\boldsymbol{x} \in \{0,1\}^N$ for some arbitrary $N \geq 1$ with time and space complexity bounds given by $T, S \geq 1$ (as $1^T, 1^{2^S}$) respectively, and the private attribute $\boldsymbol{z} \in \mathbb{Z}_p^n$ for some arbitrary $n \geq 1$, it samples $s \leftarrow \mathbb{Z}_p$ and compute a ciphertext $\mathsf{IPFE.CT}_{\mathsf{pad}} \leftarrow \mathsf{IPFE.Enc}(\mathsf{IPFE.MPK}, [\![\boldsymbol{u}_{\mathsf{pad}}]\!]_1)$ for the vector $\boldsymbol{u}_{\mathsf{pad}}$:

vector	index_1	index_2	pad	$\mathsf{init}^{\mathsf{pub}}$	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	$\mathsf{tb}_{\tau}^{\mathsf{pub}}$	in $\mathcal{S}_{\mathsf{priv}}$
$\boldsymbol{u}_{\mathsf{pad}}$	0	0	s	0	0	0	0	0

Next, it does the following:

1. Sample a random vector $\boldsymbol{r}_{\boldsymbol{x}} \leftarrow \mathbb{Z}_p^{[0,T]\times[N]\times[S]\times\{0,1\}^S}$.
2. For each $k \in [n]$, do the following:
   (a) Sample a random element $\rho_k \leftarrow \mathbb{Z}_p$.
   (b) Compute a ciphertext $\mathsf{IPFE.CT}_{k,\mathsf{init}} \leftarrow \mathsf{IPFE.SlotEnc}(\mathsf{IPFE.MPK}, [\![\boldsymbol{u}_{k,\mathsf{init}}]\!]_1)$ for the vector $\boldsymbol{u}_{k,\mathsf{init}}$:

vector	index_1	index_2	pad	$\mathsf{init}^{\mathsf{pub}}$	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	$\mathsf{tb}_{\tau}^{\mathsf{pub}}$	in $\mathcal{S}_{\mathsf{priv}}$
$\boldsymbol{u}_{k,\mathsf{init}}$	$-k\cdot\rho_k$	ρ_k	0	$s\cdot\boldsymbol{r}_{\boldsymbol{x}}[(0,1,1,\mathbf{0}_S)]$	0	s	0	$\perp$

   (c) For all $t \in [T], i \in [N], j \in [S], \boldsymbol{W} \in \{0,1\}^S$, do the following:
      (i) Compute the transition coefficients $c_\tau(\boldsymbol{x}; t, i, j, \boldsymbol{W}; \boldsymbol{r}_{\boldsymbol{x}}), \forall \tau \in \mathcal{T}$ using $\boldsymbol{r}_{\boldsymbol{x}}$.
      (ii) Compute $\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}} \leftarrow \mathsf{IPFE.SlotEnc}(\mathsf{IPFE.MPK}, [\![\boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}]\!]_1)$ for the vector $\boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}$:

vector	index_1	index_2	pad	$\mathsf{init}^{\mathsf{pub}}$	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	$\mathsf{tb}_{\tau}^{\mathsf{pub}}$	in $\mathcal{S}_{\mathsf{priv}}$
$\boldsymbol{u}_{k,t,i,j,\boldsymbol{W}}$	$-k\cdot\rho_k$	ρ_k	0	0	$s\cdot\boldsymbol{r}_{\boldsymbol{x}}[(t-1,i,j,\boldsymbol{W})]$	0	$s\cdot c_\tau(\boldsymbol{x};t,i,j,\boldsymbol{W};\boldsymbol{r}_{\boldsymbol{x}})$	$\perp$

   (d) For $t = T+1$, and for all $i \in [N], j \in [S], \boldsymbol{W} \in \{0,1\}^S$, compute $\widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}} \leftarrow \mathsf{IPFE.SlotEnc}(\mathsf{IPFE.}\widetilde{\mathsf{MPK}}, [\![\widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}]\!]_1)$ for the vector $\widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}$:

vector	index_1	index_2	$\mathsf{rand}^{\mathsf{pub}}$	$\mathsf{acc}^{\mathsf{pub}}$	in $\widetilde{\mathcal{S}}_{\mathsf{priv}}$
$\widetilde{\boldsymbol{u}}_{k,T+1,i,j,\boldsymbol{W}}$	$-k\cdot\rho_k$	ρ_k	$s\cdot\boldsymbol{r}_{\boldsymbol{x}}[(T,i,j,\boldsymbol{W})]$	$s\cdot\boldsymbol{z}[k]$	$\perp$

3. Finally, it returns the ciphertext as

$$\mathsf{CT}_{(\boldsymbol{x},T,S)} = \Big((\boldsymbol{x},T,S), n, \mathsf{IPFE.CT}_{\mathsf{pad}}, \Big\{ \mathsf{IPFE.CT}_{k,\mathsf{init}}, \{\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}\}_{t\in[T]}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}} \Big\}_{k\in[n],i\in[N],j\in[S],\boldsymbol{W}\in\{0,1\}^S} \Big).$$

Dec$(\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}, \mathsf{CT}_{(\boldsymbol{x},T,S)})$**:** On input a secret key $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$ and a ciphertext $\mathsf{CT}_{(\boldsymbol{x},T,S)}$, do the following:

1. Parse $\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})}$ and $\mathsf{CT}_{(\boldsymbol{x},T,S)}$ as follows:

$$\mathsf{SK}_{(\boldsymbol{M},\mathcal{I}_{\boldsymbol{M}})} = \Big(((M_k)_{k\in\mathcal{I}_{\boldsymbol{M}}}, \mathcal{I}_{\boldsymbol{M}}), \mathsf{IPFE.SK}_{\mathsf{pad}}, \Big\{ \mathsf{IPFE.SK}_{k,\mathsf{init}}, \{\mathsf{IPFE.SK}_{k,q}, \widetilde{\mathsf{IPFE.SK}}_{k,q}\}_{q\in[Q_k]} \Big\}_{k\in\mathcal{I}_{\boldsymbol{M}}} \Big), M_k = (Q_k, \boldsymbol{y}_k, \delta_k),$$

$$\mathsf{CT}_{(\boldsymbol{x},T,S)} = \Big((\boldsymbol{x},T,S), n, \mathsf{IPFE.CT}_{\mathsf{pad}}, \Big\{ \mathsf{IPFE.CT}_{k,\mathsf{init}}, \{\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}\}_{t\in[T]}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}} \Big\}_{k\in[n],i\in[N],j\in[S],\boldsymbol{W}\in\{0,1\}^S} \Big).$$

2. Output $\perp$, if $\mathcal{I}_{\boldsymbol{M}} \not\subset [n]$. Else, select the sequence of ciphertexts for the indices $k \in \mathcal{I}_{\boldsymbol{M}}$ as

$$\mathsf{CT}_{(\boldsymbol{x},T,S)} = \Big((\boldsymbol{x},T,S), \Big\{ \mathsf{IPFE.CT}_{k,\mathsf{init}}, \{\mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}\}_{t\in[T]}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}} \Big\}_{k\in\mathcal{I}_{\boldsymbol{M}},i\in[N],j\in[S],\boldsymbol{W}\in\{0,1\}^S} \Big).$$

3. Use the IPFE decryption to obtain $[\![\mu_{\mathsf{pad}}]\!]_{\mathrm{T}} \leftarrow \mathsf{IPFE.Dec}(\mathsf{IPFE.SK}_{\mathsf{pad}}, \mathsf{IPFE.CT}_{\mathsf{pad}})$.
4. Recall that $\forall k \in \mathcal{I}_{\boldsymbol{M}}, \mathcal{C}_{M_k,N,S} = [N] \times [S] \times \{0,1\}^S \times [Q_k]$, and that we denote any element in it as $\theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S}$ where the only component in the tuple θ_k depending on k is $q \in [Q_k]$. Invoke the IPFE decryption to compute all label values as:

$$\begin{aligned}
&\forall k \in \mathcal{I}_{\boldsymbol{M}} : [\![\ell_{k,\mathsf{init}}]\!]_{\mathrm{T}} = \mathsf{IPFE.Dec}(\mathsf{IPFE.SK}_{k,\mathsf{init}}, \mathsf{IPFE.CT}_{k,\mathsf{init}}) \\
&\forall k \in \mathcal{I}_{\boldsymbol{M}}, t \in [T], \theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S} : \\
&\qquad [\![\ell_{k,t,\theta_k}]\!]_{\mathrm{T}} = \mathsf{IPFE.Dec}(\mathsf{IPFE.SK}_{k,q}, \mathsf{IPFE.CT}_{k,t,i,j,\boldsymbol{W}}) \\
&\forall k \in \mathcal{I}_{\boldsymbol{M}}, \theta_k = (i, j, \boldsymbol{W}, q) \in \mathcal{C}_{M_k,N,S} : \\
&\qquad [\![\ell_{k,T+1,\theta_k}]\!]_{\mathrm{T}} = \mathsf{IPFE.Dec}(\widetilde{\mathsf{IPFE.SK}}_{k,q}, \widetilde{\mathsf{IPFE.CT}}_{k,T+1,i,j,\boldsymbol{W}})
\end{aligned}$$

5. Next, invoke the AKGS evaluation procedure and obtain the combined value

$$[\![\mu]\!]_{\mathrm{T}} = \prod_{k \in \mathcal{I}_{\boldsymbol{M}}} \mathsf{Eval}\left(\left(M_k, 1^N, 1^T, 1^{2^S}, p\right), \boldsymbol{x}, [\![\ell_{k,\mathsf{init}}]\!]_{\mathrm{T}}, \left\{[\![\ell_{k,t,\theta_k}]\!]_{\mathrm{T}}\right\}_{t \in [T+1], \theta_k \in \mathcal{C}_{M_k,N,S}}\right)$$

6. Finally, it returns μ' such that $[\![\mu]\!]_{\mathrm{T}} = ([\![\mu_{\mathsf{pad}}]\!]_{\mathrm{T}})^{\mu'}$, where $g_{\mathrm{T}} = e(g_1, g_2)$. Similar to [2], we assume that the desired attribute-weighted sum lies within a specified polynomial-sized domain so that μ' can be searched via brute-force.

The correctness of our $\mathsf{PK\text{-}UAWS}^{\mathsf{L}}_{(\mathsf{poly},1,1)}$ can be shown similarly to our secret key scheme of the previous section. Please see the full version of the paper for details.

Theorem 5.1 *Assuming the* SXDH *assumption holds in* $\mathcal{G}$ *and the* IPFE *is function hiding secure, the above construction of* 1-Slot FE *for* UAWS *is adaptively simulation secure.*

The description of the simulator and the proof of the above theorem is given in the full version.

References

1. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: PKC 2015. pp. 733–751. Springer (2015)
2. Abdalla, M., Gong, J., Wee, H.: Functional encryption for attribute-weighted sums from k-Lin. In: CRYPTO 2020. pp. 685–716. Springer (2020)
3. Agrawal, S., Libert, B., Maitra, M., Titiu, R.: Adaptive simulation security for inner product functional encryption. In: PKC 2020. pp. 34–64. Springer (2020)
4. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: CRYPTO 2016. pp. 333–362. Springer (2016)
5. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: FOCS 2011. pp. 120–129. IEEE Computer Society (2011)

6. Attrapadung, N.: Dual system encryption framework in prime-order groups via computational pair encodings. In: ASIACRYPT 2016. pp. 591–623. Springer (2016)
7. Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: EUROCRYPT 2014. pp. 533–556. Springer (2014)
8. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC 2011. pp. 253–273. Springer (2011)
9. Chen, J., Gay, R., Wee, H.: Improved dual system ABE in prime-order groups via predicate encodings. In: EUROCRYPT 2015. pp. 595–624. Springer (2015)
10. Chen, J., Gong, J., Kowalczyk, L., Wee, H.: Unbounded ABE via bilinear entropy expansion, revisited. In: EUROCRYPT 2018. pp. 503–534. Springer (2018)
11. Datta, P., Dutta, R., Mukhopadhyay, S.: Functional encryption for inner product with full function privacy. In: PKC 2016. pp. 164–195. Springer (2016)
12. Datta, P., Okamoto, T., Takashima, K.: Adaptively simulation-secure attribute-hiding predicate encryption. In: ASIACRYPT 2018. pp. 640–672. Springer (2018)
13. Datta, P., Pal, T.: (compact) adaptively secure fe for attribute-weighted sums from k-lin. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 434–467. Springer (2021)
14. Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: ICALP 2014. pp. 650–662. Springer (2014)
15. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: EUROCRYPT 2010. pp. 62–91. Springer (2010)
16. Lewko, A.B., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: TCC 2010. pp. 455–479. Springer (2010)
17. Lewko, A.B., Waters, B.: Unbounded HIBE and attribute-based encryption. In: EUROCRYPT 2011. pp. 547–567. Springer (2011)
18. Lin, H., Luo, J.: Compact adaptively secure abe from k-Lin: Beyond NC^1 and towards NL. In: EUROCRYPT 2020. pp. 247–277. Springer (2020)
19. Lin, H., Vaikuntanathan, V.: Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In: FOCS 2016. pp. 11–20. IEEE (2016)
20. Okamoto, T., Takashima, K.: Fully secure functional encryption with general relations from the decisional linear assumption. In: CRYPTO 2010. pp. 191–208. Springer (2010)
21. Okamoto, T., Takashima, K.: Fully secure unbounded inner-product and attribute-based encryption. In: ASIACRYPT 2012. pp. 349–366. Springer (2012)
22. O'Neill, A.: Definitional issues in functional encryption. IACR Cryptology ePrint Archive, Report 2010/556 (2010)
23. Tomida, J., Takashima, K.: Unbounded inner product functional encryption from bilinear maps. Japan Journal of Industrial and Applied Mathematics **37**(3), 723–779 (2020)
24. Waters, B.: Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In: CRYPTO 2009. pp. 619–636. Springer (2009)
25. Wee, H.: Dual system encryption via predicate encodings. In: TCC 2014. pp. 616–637. Springer (2014)
26. Wee, H.: Attribute-hiding predicate encryption in bilinear groups, revisited. In: TCC 2017. pp. 206–233. Springer (2017)