Multi-Client Functional Encryption for Linear Functions in the Standard Model from LWE

Benoît Libert, Radu Titiu

To cite this version:


HAL Id: hal-02352139
https://inria.hal.science/hal-02352139
Submitted on 6 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Multi-Client Functional Encryption for Linear Functions in the Standard Model from LWE

Benoît Libert\textsuperscript{1,2} and Radu Ţiţiu\textsuperscript{2,3}

\textsuperscript{1} CNRS, Laboratoire LIP, France
\textsuperscript{2} ENS de Lyon, Laboratoire LIP (U. Lyon, CNRS, ENSL, INRIA, UCBL), France
\textsuperscript{3} Bitdefender, Bucharest, Romania

Abstract. Multi-client functional encryption (MCFE) allows $\ell$ clients to encrypt ciphertexts $(C_{t,1}, C_{t,2}, \ldots, C_{t,\ell})$ under some label. Each client can encrypt his own data $X_i$ for a label $t$ using a private encryption key $ek_i$ issued by a trusted authority in such a way that, as long as all $C_{t,i}$ share the same label $t$, an evaluator endowed with a functional key $dk_f$ can evaluate $f(X_1, X_2, \ldots, X_\ell)$ without learning anything else on the underlying plaintexts $X_i$. Functional decryption keys can be derived by the central authority using the master secret key. Under the Decision Diffie-Hellman assumption, Chotard \textit{et al.} (Asiacrypt 2018) recently described an adaptively secure MCFE scheme for the evaluation of linear functions over the integers. They also gave a decentralized variant (DMCFE) of their scheme which does not rely on a centralized authority, but rather allows encryptors to issue functional secret keys in a distributed manner. While efficient, their constructions both rely on random oracles in their security analysis. In this paper, we build a standard-model MCFE scheme for the same functionality and prove it fully secure under adaptive corruptions. Our proof relies on the Learning-With-Errors (LWE) assumption and does not require the random oracle model. We also provide a decentralized variant of our scheme, which we prove secure in the static corruption setting (but for adaptively chosen messages) under the LWE assumption.

Keywords. Multi-client functional Encryption, inner product evaluation, LWE, standard model, decentralization.

1 Introduction

Functional encryption (FE) \cite{Boneh2010, Boneh2005} is a modern paradigm that overcomes the all-or-nothing nature of ordinary encryption schemes. In FE, the master secret key $msk$ allows deriving a sub-key $dk_f$ associated with a specific function $f$. If a ciphertext $C$ encrypts a message $X$ under the master public key $mpk$, when $dk_f$ is used to decrypt $C$, the decryptor only obtains $f(X)$ and nothing else about $X$. Functional encryption is an extremely general concept as it subsumes identity-based encryption \cite{Boneh2001, Boneh2006}, searchable encryption \cite{Boneh2007}, attribute-based encryption \cite{Boneh2005, Boneh2010}, broadcast encryption \cite{Boneh2005} and many others.

As formalized by Boneh, Sahai and Waters \cite{Boneh2010}, FE only allows evaluating a function $f$ over data provided by a single sender whereas many natural applications require to compute over data coming from distinct distrustful parties. A
straightforward solution to handle multiple senders is to distribute the generation of ciphertexts by means of a multi-party computation (MPC) protocol. Unfortunately, jointly generating a ciphertext incurs potentially costly interactions between the senders who should be online at the same time and have their data ready to be submitted. Ideally, the participants should be able to supply their input without interacting with one another and go off-line immediately after having sent their contribution. This motivates the concepts of multi-input [39,38] and multi-client [44,38] functional encryption, which support the evaluation of multivariate functions over data coming from distinct sources.

1.1 (Decentralized) Multi-Client FE

**Multi-client functional encryption.** As defined in [44,38], multi-client functional encryption (MCFE) allows computing over input vectors \((X_1, \ldots, X_\ell)\) of which each coordinate \(X_i\) may be sent by a different client. Each ciphertext \(C_i\) is associated with a client index \(i\) and a tag \(t\) (also called “label”): on input of a vector of ciphertexts \((C_1 = \text{Encrypt}(1, X_1, t), \ldots, C_\ell = \text{Encrypt}(\ell, X_\ell, t))\), where \(C_i\) is generated by client \(i\) using a secret encryption key \(ek_i\) for each \(i \in [\ell]\), anyone holding a functional decryption key \(dk_f\) for an \(\ell\)-ary function can compute \(f(X_1, \ldots, X_\ell)\) as long as all \(C_i\) are labeled with the same tag \(t\) (which may be a time-specific information or a dataset name). No further information than \(f(X_1, \ldots, X_\ell)\) is revealed about individual inputs \(X_i\) and nothing can be inferred by combining ciphertexts generated for different tags. MCFE can thus be seen as a multi-party computation (MPC) where each ciphertext \(C_i\) can be generated independently of others and no communication is needed between data providers.

**Decentralized multi-client functional encryption.** Most FE flavors involve a single central authority that should not only be trusted by all users, but also receives the burden of generating all functional secret keys. In decentralized FE systems [53,25], multiple authorities can operate independently without even being aware of one another.

Like its single-client counterpart, multi-client FE requires a trusted entity, which is assigned the task of generating a master key \(msk\) as well as handing out encryption keys \(ek_i\) to all clients and functional decryption keys \(dk_f\) to all decryptors. In some applications, clients may be reluctant to rely on a single point of trust. This motivates the design of a decentralized version of MCFE, as introduced by Chotard et al. [28]. Decentralized multi-client functional encryption (DMCFE) obviates the need for a centralized authority by shifting the task of generating functional secret keys to the clients themselves. In a setup phase, the clients \(S_1, \ldots, S_\ell\) first generate public parameters by running an interactive protocol but no further interaction is needed among clients when it comes to generating functional secret keys later on. When a decryptor wishes to obtain a functional secret key for an \(\ell\)-ary function \(f\), it interacts with each client \(i\) independently so as to obtain partial functional decryption keys \(dk_{f,i}\). The decryptor can then fold \(\{dk_{f,i}\}_{i=1}^\ell\) into a functional decryption key \(dk_f\) for \(f\). By
doing so, each client has full control over his individual data and the functions for which secret keys are given out. Importantly, no interaction among senders is required beyond the setup phase, where public parameters are generated.

As a motivating example, Chotard et al. [28] consider the use-case of a financial analyst that is interested in mining several companies’ private data so as to better understand the dynamics of an economical sector. These companies have some incentives to collaborate, but they do not want their clients’ data to be abused (in which case, they would risk heavy fines owing to the EU General Data Protection Regulation). After having interactively set up DMCFE parameters, each company can encrypt its own data with respect to a time-stamp. Then, the analyst can contact each company to obtain partial functional keys and reconstruct a key that only reveals a weighted aggregate of companies’ private inputs provided they are labeled with the same time-stamp.

Chotard et al. [28] described a DMCFE scheme that allows evaluating linear functions over encrypted data: namely, if \((X_1, \ldots, X_\ell) \in \mathbb{Z}^\ell\) are the individual contributions sent by \(\ell\) senders, a functional secret key \(d_kf\) for the integer vector \(y = (y_1, \ldots, y_\ell) \in \mathbb{Z}^\ell\) allows computing \(\sum_{i=1}^{\ell} y_i \cdot X_i\) from \(\{C_i = \text{Encrypt}(i, X_i, t)\}_{i=1}^{\ell}\), where \(C_i\) is generated by the \(i\)-th sender. In the decentralized setting, each sender can also generate a partial functional secret key \(d_{kf,i}\) for \(y = (y_1, \ldots, y_\ell) \in \mathbb{Z}^\ell\) using their secret encryption key \(e_k\).

1.2 Our Contributions

The MCFE scheme of Chotard et al. [28] was proved fully secure (as opposed to selectively secure) in the random oracle model under the standard Decision Diffie-Hellman assumption in groups without a bilinear maps. Its decentralized variant was proved secure under the Symmetric eXternal Diffie-Hellman (SXDH) assumption in groups endowed with an asymmetric bilinear map. While efficient, the schemes of [28] both require the random oracle model. Chotard et al. thus left open the problem of designing a (D)MCFE system under well-studied hardness assumptions without using random oracles: even in the centralized setting, the only known MCFE candidates in the standard model [44, 38] relies on indistinguishability obfuscation. They also left open the problem of instantiating their schemes under the LWE assumption or any other assumption than DDH.

In this paper, we address both problems. For linear functions over the integers (i.e., the same functionality as [28]), we construct the first MCFE scheme in the standard model and prove it fully secure under the Learning-With-Errors assumption [62] in the adaptive corruption setting (note that only static corruptions were considered in [44, Section 2.3]). This construction turns out to be the first standard-model realization of an MCFE system with labels – albeit for a restricted functionality – that does not require obfuscation. Next, we extend our centralized system to obtain the first labeled DMCFE scheme without random oracles. Like [28], our decentralized solution is only proved secure in the static corruption setting although we can handle adaptive corruptions in its centralized version. Both constructions are proved secure under the LWE assumption with sub-exponential approximation factors. Our security proofs stand in the standard
model in the sense of the same security definitions as those considered in [28].

We leave it as an open problem to achieve security under an LWE assumption with polynomial approximation factor. Another natural open question is the feasibility of (D)MCFE beyond linear functions under standard assumptions.

### 1.3 Challenges and Techniques

We start from the observation that the DDH-based MCFE scheme of Chotard et al. [28] can be interpreted as relying on (a variant of) the key-homomorphic pseudorandom function [19] of Naor, Pinkas and Reingold [60]. Namely, the scheme of [28] encrypts \( x_i \in \mathbb{Z}_q \) for the tag \( t \) by computing \( C_i = g^{x_i} \cdot H_{t,1}^i \cdot H_{t,2}^i \), where \((s_i, t_i) \in \mathbb{Z}_q^2\) is the \( i \)-th sender’s secret key and \((H_{t,1}, H_{t,2}) = H(t) \in \mathbb{G}^2\) is derived from a random oracle in a DDH-hard group \( \mathbb{G} = \langle g \rangle \).

The security proof of [28] crucially exploits the entropy of the secret key \((s_i, t_i)\) in a hybrid argument over all encryption queries. To preserve this entropy, they need to prevent the encryption oracle from leaking too much about uncorrupted users’ secret keys \{\((s_i, t_i)\)\}. For this purpose, they rely on the DDH assumption to modify the random oracle \( H : \{0, 1\}^* \rightarrow \mathbb{G}^2\) in such a way that, in all encryption queries but one, the hash value \( H(t) \in \mathbb{G}^2\) lives in a one-dimensional subspace. In order to transpose this technique in the standard model, we would need a programmable hash function [47] that ranges over a one-dimensional subspace of \( \mathbb{G}^2\) on polynomially-many inputs while mapping an extra input outside this subspace with noticeable probability. The results of Hanaoka et al. [46] hint that such programmable hash functions are hardly instantiable in prime-order DDH groups. While the multi-linear setting [34] allows bypassing the impossibility results of [46], it is not known to enable standard assumptions.

A natural idea is to replace the random-oracle-based key-homomorphic PRF of [60] by an LWE-based key-homomorphic PRF [19,12]. However, analogously to Chotard et al. [28],\(^4\) we aim at an MCFE system that can be proved secure in a game where the adversary is allowed to corrupt senders adaptively. In order to deal with the adaptive corruption of senders, we thus turn to the adaptively secure distributed PRF proposed by Libert, Stehlé and Titiu [56]. The latter can be seen as instantiating the programmable hash function of Freire et al. [34] in the context of homomorphic encryption (FHE). Their PRF maps an input \( x \) to \([A(x)^\top \cdot s]_p\), where\(^5\) \( s \in \mathbb{Z}^n\) is the secret key and \( A(x) \in \mathbb{Z}_q^{n \times m}\) is derived from public matrices using the Gentry-Sahai-Waters FHE [37]. More precisely, the matrix \( A(x)\) is obtained as the product of GSW ciphertexts dictated by the output of an admissible hash function [16] applied to the PRF input. The security proof of [56] uses the property that, with noticeable probability, the input-dependent matrix \( A(x)\) is a GSW encryption of 1 for the challenge input \( x^*\): namely, \( A(x^*)\) is a matrix of of the form \( A(x^*) = A \cdot R^* + G\), where \( G \in \mathbb{Z}_q^{n \times m}\) is

---

\(^4\) While their decentralized scheme is only proved secure under static corruptions, its centralized version is proved secure under adaptive corruptions.

\(^5\) Introduced in [13], the notation \([x]_p\) stands for the rounded value \([(p/q) \cdot x] \in \mathbb{Z}_q\), where \( x \in \mathbb{Z}_q\), and \( p < q\).
the gadget matrix of Micciancio and Peikert \cite{58} and \( R^* \in \mathbb{Z}^{m \times m} \) is a small-norm matrix. At the same time, all evaluation queries are associated with a matrix \( A(x) \) consisting of a GSW encryption of 0 (i.e., a matrix \( A(x) = A \cdot R \), for a small-norm \( R \in \mathbb{Z}^{m \times m} \). Then, the proof of \cite{56} appeals to the lossy mode of LWE \cite{40} and replaces the uniform matrix \( A^T \in \mathbb{Z}_q^{n \times n} \) by a lossy matrix of the form \( \hat{A}^T \cdot C + E \), where \( E \in \mathbb{Z}^{m \times n} \) is a short integer matrix with Gaussian entries, \( C \in \mathbb{Z}_q^{n_1 \times m} \) is random, and \( \hat{A} \in \mathbb{Z}_q^{n \times m} \) has rank \( n_1 \ll n \). In all evaluation queries, the smallness of \( s \in \mathbb{Z}^n \) then ensures that the values \( [A(x)^\top \cdot s]_p \) always reveal the same information about \( s \), which amounts to the product \( C \cdot s \in \mathbb{Z}_q^{n_1} \).

Since \( A(x^*) \) depends on \( G \) for the challenge input \( x^* \), the function \( [A(x^*)^\top \cdot s]_p \) is in fact an injective function of \( s \), meaning that it has high min-entropy.

Our MCFE scheme relies on the lossy mode of LWE in a similar way to \cite{56}, except that we add a Gaussian noise instead of using the Learning-With-Rounding technique \cite{13}. The \( i \)-th sender uses his secret key \( s_i \in \mathbb{Z}^n \) to encrypt a short integer vector as \( x_i \in \mathbb{Z}^{n_0} \) as \( C_i = G_0^\top \cdot x_i + A(t)^\top \cdot s_i + \text{noise} \in \mathbb{Z}_q^n \), where \( A(t) \in \mathbb{Z}_q^{m \times m} \) is a tag-dependent matrix derived as a product of GSW ciphertexts indexed by the bits of \( t \) and \( G_0 \in \mathbb{Z}_q^{m \times m} \) is a gadget matrix for which the lattice \( A^\top (G_0) \) has a short public basis. A functional secret key for the vector \( y = (y_1, \ldots, y_q)^\top \) consists of \( dk_y = \sum_{i=1}^q y_i \cdot s_i \in \mathbb{Z}^n \) and allows computing \( G_0^\top \cdot (\sum_{i=1}^q y_i \cdot x_i) + \text{small} \in \mathbb{Z}_q^n \) from \( \sum_{i=1}^q y_i \cdot C_i \in \mathbb{Z}_q^n \) and eventually recovering the linear function \( \sum_{i=1}^q y_i \cdot x_i \in \mathbb{Z}^{n_0} \) of \( X = [x_1 \ldots | x_q] \in \mathbb{Z}_q^{n_0 \times t} \).

At this point, adapting the security proof of \cite{56} is non-trivial. We cannot rely on the DPRF of \cite{56} in a modular way as it would require a DPRF where partial evaluations are themselves pseudorandom so long as the adversary does not obtain the underlying secret key shares: in our setting, a challenge ciphertext contains a bunch of partial evaluations (one for each message slot) rather than a threshold recombination of such evaluations. We emphasize that, in the LWE-based DPRF of \cite{56}, partial evaluations are not proven pseudorandom: \cite{56} only proves – via a deterministic randomness extraction argument – the pseudorandomness of the final PRF value obtained by combining partial evaluations. They cannot apply (and neither can we) a randomness extractor to individual partial DPRF evaluations as it would destroy their key homomorphic property. Instead of relying on the pseudorandomness of partial evaluations, we actually prove a milder indistinguishability property which suffices for our purposes.

The first step is to make sure that all encryption queries will involve a lossy matrix \( A(t)^\top = R_t \cdot \hat{A}^\top \cdot C + E_t \), for small-norm \( R_t \in \mathbb{Z}^{m \times m} \) and \( E_t \in \mathbb{Z}^{m \times n} \), so that honest senders’ ciphertexts are of the form \( C_i = G_0^\top \cdot x_i + R_t \cdot \hat{A}^\top \cdot C \cdot s_i + \text{noise} \) and thus leak nothing about \( s_i \in \mathbb{Z}^n \) beyond \( C \cdot s_i \in \mathbb{Z}^{n_1} \). The difficulty arises in the challenge queries \((i, t^*, x_{0,i}^*, x_{1,i}^*)\), where \( A(t^*) \in \mathbb{Z}_q^{n \times m} \) is not a lossy matrix and we must find a way to replace \( C_i^* = G_0^\top \cdot x_{0,i}^* + A(t^*)^\top \cdot s_i + \text{noise} \) by \( C_i^* = G_0^\top \cdot x_{1,i}^* + A(t^*)^\top \cdot s_i + \text{noise} \) without the adversary noticing. In \cite{56},
To do this, we need to modify the proof of \cite{56}, by introducing a matrix \( \hat{M} \). The alternative secret keys are used to generate partial functional secret keys whereas the second one is used for all legal functional key queries.\footnote{The standard Leftover Hash Lemma cannot be applied since the source \( \{ A(x^*)^\top \cdot s \}_p \) is not guaranteed to be independent of the seed. A deterministic extractor based on \( k \)-wise independent functions \cite{32} is thus needed in \cite{56}.}

The DPRF of \cite{56} is thus needed in \cite{56}.

In order to build a DMCFE system, we proceed analogously to \cite{28} and combine two instances of our centralized MCFE scheme. The first one is only used to generate partial functional secret keys whereas the second one is used exactly as in the centralized system. As in \cite{28}, we first have the senders run an interactive protocol allowing them to jointly generate public parameters for the two MCFE instances. At the end of this protocol (which may involve costly MPC operations, but is only executed once), each sender holds an encryption key

\[
A(t^*)^\top = R^* \cdot A^\top + G_0^\top \cdot V = R^* \cdot \hat{A}^\top \cdot C + G_0^\top \cdot V + \text{noise},
\]

for a statistically random matrix \( V \in \mathbb{Z}_{q_0}^{n_0 \times n} \) included in the public parameters. In the proof, the simulator generates a statistically uniform matrix \( U = [\chi] \) where \( C \in \mathbb{Z}_{q_1}^{n_1 \times n} \) is used to build the lossy matrix \( A^\top = \hat{A}^\top \cdot C + E \), together with a trapdoor \( T_U \) for \( \Lambda^1(U) \). The idea of embedding a trapdoor in the LWE secret of a lossy matrix is borrowed from \cite{55}). Using \( T_U \), the simulator can sample a short matrix \( T \in \mathbb{Z}^{n_0 \times m} \) satisfying \( U \cdot T = [1_{n_0}] \mod q \), allowing it to define an alternative secret key \( s_i' = s_i + T \cdot (x_{0,i}^\top - x_{1,i}^\top) \in \mathbb{Z}^n \). As long as \( s_i \) is sampled from a Gaussian distribution with sufficiently large standard deviation, \( s_i' \) and \( s_i \) are negligibly far apart in terms of statistical distance (note that, as in \cite{69,14}, the simulator can guess \( x_{0,i}^\top - x_{1,i}^\top \) upfront without affecting the polynomial running time of the reduction since we are in the middle of a purely statistical argument).

The alternative secret keys \( \{ s_i' \}_{i=1}^\ell \) further satisfy \( \sum_{i=1}^\ell y_i \cdot s_i' = \sum_{i=1}^\ell y_i \cdot s_i \) for all legal functional key queries \( y = (y_1, \ldots, y_\ell) \) made by the adversary. The definition of \( s_i' \) finally ensures that \( C \cdot s_i' = C \cdot s_i \mod q \), meaning that \( s_i' \) is compatible with all encryption queries for which \( A(t) \) is lossy. From (1), the condition \( V \cdot T = [1_{n_0}] \mod q \) then implies that the challenge ciphertext can be interpreted as an encryption of \( x_{1,i}^\top \) since \( C_i^\top = G_0^\top \cdot x_{1,i}^\top + A(t^*)^\top \cdot s_i' + \text{noise} \) is statistically close to \( C_i^\top = G_0^\top \cdot x_{0,i}^\top + A(t^*)^\top \cdot s_i + \text{noise} \).

We insist that our construction and proof are not merely obtained by plugging the DPRF of \cite{56} into the high-level design principle of \cite{28}. In particular, we do not rely on the pseudorandomness of partial PRF evaluations, but rather prove a milder indistinguishability property in some transition in our sequence of games. To do this, we need to modify the proof of \cite{56}, by introducing a matrix \( V \) and embedding a trapdoor in the matrix \( U \) obtained by stacking up \( V \) and the secret matrix \( C \) of the lossy mode of LWE.
\( \mathbf{e}_i = (s_i, t_i) \) consisting of encryption keys for the two underlying instances. In order to have the \( i \)-th sender \( S_i \) generate a partial functional secret key \( \mathbf{dk}_{f,i} \) for a vector \( \mathbf{y} = (y_1, \ldots, y_\ell) \)\(^T \), we exploit the fact that our centralized scheme allows encrypting vectors. Namely, the decryptor obtains from \( S_i \) an MCFE encryption of the vector \( y_i \cdot s_i \in \mathbb{Z}^n \) under the encryption key \( t_i \) of the first instance.

### 1.4 Related Work

Functional encryption was implicitly introduced by Sahai and Waters in [64], where they also constructed a scheme for threshold functions. Constructions of FE for point functions (known as identity-based encryption) [18,30] existed already, but were not viewed through the lens of FE until later. Subsequent works saw constructions for several more advanced functionalities such as inner product functions [51,7], Boolean formulas [45,52,61,67,54], membership checking [21] and even finite state automaton [68]. Recently, the landscape of functional encryption improved considerably. Gorbunov et al. [43] and Garg et al. [35] provided the first constructions of attribute-based encryption for all circuits; Goldwasser et al. [42] constructed succinct simulation-secure single-key FE scheme for all circuits and also obtained FE for Turing machines [41]. In a breakthrough result, Garg et al. [35] designed indistinguishability-secure multi-key FE schemes for all circuits. However, while the constructions of [43,42] rely on standard assumptions, the assumptions underlying the other constructions [35,41] are still ill-understood and have not undergone much cryptanalytic effort.

**FE for Simple Circuits.** Abdalla, Bourse, De Caro and Pointcheval [3] considered the question of building FE for linear functions (a functionality dubbed IPFE for “inner product functional encryption”). Here, a ciphertext \( C \) encrypts a vector \( \mathbf{y} \in D^\ell \) over some ring \( D \), a secret key for the vector \( \mathbf{x} \in D^\ell \) allows computing \( \langle \mathbf{x}, \mathbf{y} \rangle \) and nothing else about \( \mathbf{y} \). Abdalla et al. [3] described two constructions under the Decision Diffie-Hellman (DDH) and Learning-With-Errors (LWE) assumptions, respectively. On the downside, Abdalla et al. [3] only proved their schemes to be secure against selective adversaries. Namely, in the security game, the adversary chooses two vectors \( \mathbf{x}_0, \mathbf{x}_1 \in D^\ell \) and expects to receive an encryption of one of these in the challenge phase. Selective security forces the adversary to declare \( \mathbf{x}_0, \mathbf{x}_1 \) before seeing the public key and before obtaining any private key. Agrawal, Libert and Stehlé subsequently upgraded the constructions of [3] so as to prove security against adaptive adversaries, which may choose \( \mathbf{x}_0, \mathbf{x}_1 \) after having seen the public key and obtained a number of private keys. Agrawal et al. [8] described several IPFE schemes under well-established assumptions which include the standard Decision Diffie-Hellman (DDH) assumption, the Decision Composite Residuosity (DCR) assumption and the LWE assumption. Under the DCR and LWE assumptions, the schemes of [8] can evaluate both inner products over the integers and modulo a prime or composite number. The IPFE constructions of [3,8] served as building blocks for FE schemes handling general functionalities [9] in the bounded collusion setting [63,43]. Quite recently, the IPFE functionality [3,8] was extended into FE schemes supporting the evaluation
of quadratic functions over encrypted data [57,11]. The schemes of [57,11] are only proved secure against selective adversaries and they can only compute functions which have their output confined in a small interval. For the time being, the only known FE schemes that support the evaluation of more general functions than quadratic polynomials either require fancy tools like obfuscation [35], or are restricted to bounded collusions [43,9].

**Multi-Input and Multi-Client Functional Encryption.** Goldwasser et al. [39,38] introduced the concept of multi-input functional encryption (MIFE). MIFE and MCFE are both more interesting in the secret-key setting than in the public-key setting, where much more information inevitably leaks about the data (see, e.g., [39,5,28]). Similarly to MCFE, MIFE operates over input vectors \((X_1,\ldots,X_\ell)\) comprised of messages sent by distinct parties, but without assigning a tag to ciphertexts: each user \(i\) can encrypt \(X_i\) as \(C_i = \text{Encrypt}(X_i)\) in such a way that anyone equipped with a functional secret key \(\text{sk}_f\) for an \(\ell\)-argument function \(f\) can compute \(f(X_1,\ldots,X_n)\) given multiple ciphertexts \(\{C_i = \text{Encrypt}(X_i)\}_{i=1}^\ell\).

Brakerski et al. [23] gave a transformation for constructing adaptively secure general-purpose MIFE schemes for a constant \(n\) from any general-purpose private-key single-input scheme. Like MCFE, MIFE for general functionalities necessarily rely on indistinguishability obfuscation or multilinear maps, so that instantiations under standard assumptions are currently lacking. Under the SXDH assumption, Abdalla et al. [5] managed to construct a MIFE scheme for the inner product functionality. In their scheme, each input slot encrypts a vector \(x_i \in \mathbb{Z}_p^m\) while each functional secret key \(\text{sk}_y\) corresponds to a vector \(y \in \mathbb{Z}_p^{\ell \cdot m}\), where \(\ell\) is the total number of slots. On input of encrypted data \(X = (x_1,\ldots,x_\ell)\) such that \(x_i\) is encrypted by sender \(i\) in the \(i\)-th slot, their multi-input inner product functionality computes \((X,y)\) using \(\text{sk}_y\). Function-hiding MIFE schemes were described in [31,4]. Abdalla et al. [4] notably gave a generic single-input to multi-input transformation, which yields MIFE constructions for the inner product functionality under the DDH, LWE and DCR assumptions.

Besides syntactical differences, MCFE departs from MIFE in the amount of information leaked about plaintexts. The MIFE model [39,38] allows any slot of any ciphertext to be combined with any other slot of any other ciphertext. As soon as senders encrypt more than one ciphertext per slot, a given functional secret key can thus compute a much larger number of values. As discussed in [28], this feature incurs a much more important information leakage, especially when many functional secret keys are given out. In contrast, the multi-client setting only allows functional secret keys to operate over ciphertexts that share the same tag. As long as tags are single-use (e.g., a timestamp), this allows clients to retain a more accurate control over the information leaked about their data.

The first MCFE realization was proposed in [44,38] and relies on the DDH assumption and on indistinguishability obfuscation to handle general circuits. The notion of aggregator-oblivious encryption (AOE) [66,24,49,15] allows an untrusted aggregator to compute sums of encrypted values without learning anything else about individual inputs. As such, AOE can be seen as a form of MCFE with single-key security (namely, the only key revealed to the aggregator.
is for the vector \((1, 1, \ldots, 1)^\top\) for the evaluation of inner products. So far, all non-interactive AOE constructions \([66, 49, 15]\) rely on the random oracle model.

The first efficient MCFE scheme with multi-key security was described by Chotard \textit{et al.} \([28]\) who also introduced the concept of decentralized MCFE. Their schemes both rely on DDH-like assumptions in the random oracle model. At the time of writing, we are not aware of any (D)MCFE construction based on a well-studied assumption in the standard model.

**Decentralized Functional Encryption.** The first examples of decentralized FE schemes were given in the context of attribute-based encryption (ABE) \([26, 27]\). Lewko and Waters \([53]\) gave the first ciphertext-policy ABE where users’ attributes may be certified by completely independent authorities. Boneh and Zhandry \([22]\) suggested distributed broadcast encryption systems, which dispense with the need for an authority handing out keys to registered users. Chandran \textit{et al.} \([25]\) considered decentralized general-purpose FE using obfuscation. The decentralization of multi-client FE was first considered by Chotard \textit{et al.} \([28]\) in a model where all clients run an interactive protocol to generate public parameters, but eliminate any interaction beyond the setup phase.

Abdalla \textit{et al.} \([2]\) described generic transformations providing DMCFE schemes from any MCFE system satisfying extra properties. While applying their compilers to \([4]\) yields DMCFE schemes in the standard model, the resulting ciphertexts are not labeled. Without labels, the functionality leaks much more information about encrypted messages for a given functional key since there is no restriction on the way slots from different ciphertexts can be combined together (any slot from any ciphertext can be combined with any other slot from any other ciphertext). In this paper, our goal is to support labels, which is significantly more challenging and was only achieved in the random oracle model so far.

Chotard \textit{et al.} \([29]\) gave a technique to remove the restriction that forces the adversary to make challenge queries for all uncorrupted ciphertext slots. Their technique upgrades any MCFE scheme satisfying our definition (which is the definition introduced in \([28]\) and called “pos-IND” security in \([2]\)) so as to prove security under a stronger definition where the adversary can obtain incomplete ciphertexts. Their technique builds on a “secret-sharing layer” (SSL) primitive which is only known to exist assuming pairings and random oracles as their SSL scheme \([29, \text{Section 4.2}]\) is implicitly based on the Boneh-Franklin IBE \([18]\). Abdalla \textit{et al.} \([2]\) suggested a different technique to handle incomplete ciphertexts without using pairings, but they either require random oracles or they do not support labels (except in a model with static corruptions and selective security).

Chotard \textit{et al.} \([29]\) also showed how to transform the ROM-based scheme from \([28]\) in such a way that users are allowed multiple encryption queries for each slot-label pair. Their technique is not generic and only works for their DDH-based construction (as they mention in Section 6.2). Finally, \([2, 29]\) both give generic compilers from MCFE to DMCFE. Abdalla \textit{et al.} \([2]\) obtain DMCFE under adaptive corruptions, but they need to start from an MCFE which computes inner products modulo an integer \(L\) (instead of inner products over \(\mathbb{Z}\)). Hence, their compiler does not imply DMCFE from LWE in the standard model. As it
turns out, neither [2, 29] implies MCFE with labels in the standard model from LWE (nor any standard assumption), even for the security definition of [28]. In a concurrent and independent work [1], Abdalla et al. provide a solution to this problem via a generic construction of labeled MCFE from single-input IPFE schemes evaluating modular inner products. While their construction satisfies a stronger security notion than ours (which allows multiple encryption queries for the same slot-label pair), their scheme of [1, Section 3] requires longer ciphertext than ours as each slot takes a full IPFE ciphertext of linear size in \( \ell \) if \( \ell \) is the number of slots.

In their construction and in ours, handling incomplete ciphertexts expands partial ciphertexts by a factor \( O(\ell) \). In our most efficient schemes, we still need to assume that the adversary obtains challenge ciphertexts for all clients as in [28]. In Appendix A, we show that a variant of the compiler of Abdalla et al. [2] allows proving security in the standard model, even when the adversary is allowed to obtain incomplete challenge ciphertexts. Our compiler relies on pseudorandom functions satisfying a specific security definition in the multi-instance setting. The concurrent work of Abdalla et al. [1] achieves a similar result using any PRF satisfying a standard security definition.

2 Background

2.1 Lattices

For any \( q \geq 2 \), let \( \mathbb{Z}_q \) denote the ring of integers modulo \( q \). For a vector \( \mathbf{x} \in \mathbb{R}^n \) denote \( \| \mathbf{x} \| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \) and \( \| \mathbf{x} \|_\infty = \max_i | x_i | \). If \( \mathbf{M} \) is a matrix over \( \mathbb{R} \), then \( \| \mathbf{M} \| := \sup_{\mathbf{x} \neq 0} \| \mathbf{M} \mathbf{x} \| / \| \mathbf{x} \| \) and \( \| \mathbf{M} \|_\infty := \sup_{\mathbf{x} \neq 0} \| \mathbf{M} \mathbf{x} \|_\infty \). For a finite set \( S \), we let \( U(S) \) denote the uniform distribution over \( S \). If \( X \) and \( Y \) are distributions over the same domain, then \( \Delta(X, Y) \) denotes their statistical distance. Let \( \Sigma \in \mathbb{R}^{n \times n} \) be a symmetric positive-definite matrix, and \( c \in \mathbb{R}^n \). We define the Gaussian function on \( \mathbb{R}^n \) by \( \rho_{\Sigma, c}(\mathbf{x}) = \exp(-\pi(x - c)^\top \Sigma^{-1}(x - c)) \) and if \( \Sigma = \sigma^2 \cdot \mathbf{I}_n \) and \( \mathbf{c} = \mathbf{0} \) we denote it by \( \rho_{\Sigma} \).

For an \( n \)-dimensional lattice \( \Lambda \subset \mathbb{R}^n \) and for any lattice vector \( \mathbf{x} \in \Lambda \) the discrete gaussian is defined as \( \rho_{\Lambda, \Sigma, c}(\mathbf{x}) = \frac{\rho_{\Sigma, c}(\mathbf{x})}{\rho_{\Sigma}(\mathbf{0})} \).

For an \( n \)-dimensional lattice \( \Lambda \), we define \( \eta_\varepsilon(\Lambda) \) as the smallest \( r > 0 \) such that \( \rho_{1/r}(\Lambda \setminus \mathbf{0}) \leq \varepsilon \) with \( \Lambda \) denoting the dual of \( \Lambda \), for any \( \varepsilon \in (0, 1) \). For a matrix \( \mathbf{A} \in \mathbb{Z}_q^{n \times m} \), we define \( \Lambda^1(\mathbf{A}) = \{ \mathbf{x} \in \mathbb{Z}^m : \mathbf{A} \cdot \mathbf{x} \equiv \mathbf{0} \mod q \} \) and \( \Lambda(\mathbf{A}) = \Lambda^\top \cdot \mathbb{Z}^n + q \mathbb{Z}^m \). For an arbitrary vector \( \mathbf{u} \in \mathbb{Z}_q^n \), we also define the shifted lattice \( \Lambda^\mathbf{u}(\mathbf{A}) = \{ \mathbf{x} \in \mathbb{Z}^m : \mathbf{A} \cdot \mathbf{x} \equiv \mathbf{u} \mod q \} \).

**Definition 2.1 (LWE).** Let \( m \geq n \geq 1 \), \( q \geq 2 \) and \( \alpha \in (0, 1) \) be functions of a security parameter \( \lambda \). The LWE problem consists in distinguishing between the distributions \( (\mathbf{A}, \mathbf{As} + e) \) and \( U(\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^n) \), where \( \mathbf{A} \sim U(\mathbb{Z}_q^{m \times n}) \), \( s \sim U(\mathbb{Z}_q^n) \) and \( e \sim D_{\mathbb{Z}_{\lambda \cdot n \cdot q}} \). For an algorithm \( \mathcal{A} : \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^n \rightarrow \{0, 1\} \), we define:

\[
\text{Adv}_{q, m, n, \mathbf{A}, \mathbf{As} + e}(\mathcal{A}) = |\Pr[\mathcal{A}(\mathbf{A}, \mathbf{As} + e) = 1] - \Pr[\mathcal{A}(\mathbf{A}, \mathbf{u}) = 1]|,
\]
where the probabilities are over $A \sim U(\mathbb{Z}_q^{m \times n})$, $s \sim U(\mathbb{Z}_q^n)$, $u \sim U(\mathbb{Z}_q^m)$ and $e \sim D_{\mathbb{Z}_{m,q}}$ and the internal randomness of $A$. We say that $\text{LWE}_{q,m,n,\alpha}$ is hard if, for any ppt algorithm $A$, the advantage $\text{Adv}_{\text{LWE}}(A)$ is negligible.

Micciancio and Peikert [58] described a trapdoor mechanism for LWE. Their technique uses a “gadget” matrix $G \in \mathbb{Z}_q^{n \times w}$, with $w = n \log q$, for which anyone can publicly sample short vectors $x \in \mathbb{Z}^w$ such that $G \cdot x = 0$.

**Lemma 2.2 ([58, Section 5]).** Let $m \geq 3n \log q$. There exists a ppt algorithm \text{GenTrap} that outputs a statistically uniform matrix $A \in \mathbb{Z}_q^{n \times m}$, together with a trapdoor $T_A \in \mathbb{Z}_q^{m \times m}$ for $A^\perp(A)$, such that $\max_j \|\tilde{t}_j\| \leq O(\sqrt{n \log q})$, where $\tilde{t}_j$ are the corresponding Gram-Schmidt vectors.

It is known [58] that, for any $u \in \mathbb{Z}_q^m$, a trapdoor for $A \in \mathbb{Z}_q^{n \times m}$ allows sampling from $D_{A^\perp(A),s,\omega}(\sqrt{\log m})$ for $s = O(\sqrt{n \log q})$. Since

$$\eta_{2^{-m}}(A^\perp(A)) \leq \max_j \|\tilde{t}_j\| \cdot \omega(\sqrt{\log m}) \leq s \cdot \omega(\sqrt{\log m})$$

for large enough $s = O(\sqrt{n \log q})$, the magnitude of a vector $x$ sampled from $D_{A^\perp(A),s,\omega}(\sqrt{\log m})$, is bounded by $\|x\| \leq \sqrt{m} \cdot \omega(\sqrt{\log m})$.

**Remark 2.3.** For $m \geq 3n \log q$, we can thus sample a statistically uniform matrix $A$ from $\mathbb{Z}_q^{n \times m}$ together with a trapdoor, which allows finding small solutions of $A \cdot x = u \mod q$, with $\|x\| \leq \sqrt{m} \cdot \omega(\sqrt{\log m}) = O(\sqrt{mn \log q}) \cdot \omega(\sqrt{\log m})$.

We sometimes rely on the so-called “noise flooding” technique via the next lemma.

**Lemma 2.4 ([40, Lemma 3]).** Let $y \in \mathbb{Z}^m$. The statistical distance between $D_{\mathbb{Z}_q^m,s}$ and $y + D_{\mathbb{Z}_q^m,s}$ is at most $\Delta(D_{\mathbb{Z}_q^m,s}, y + D_{\mathbb{Z}_q^m,s}) \leq m \cdot \|y\|_\infty$.

**Lemma 2.5 ([36, Theorem 4.1]).** There is a ppt algorithm that, given a basis $B$ of an $n$-dimensional lattice $\Lambda = L(B)$, a parameter $s > \|B\| \cdot \omega(\sqrt{\log n})$, and a center $c \in \mathbb{R}^n$, outputs a sample from a distribution statistically close to $D_{\Lambda,s,c}$.

### 2.2 Admissible Hash Functions

Admissible hash functions were introduced by Boneh and Boyen [16] as a combinatorial tool for partitioning-based security proofs for which Freire et al. [34] gave a simplified definition. Jager [48] considered the following generalization in order to simplify the analysis of reductions under decisional assumption.

**Definition 2.6 ([48]).** Let $\ell(\lambda), L(\lambda) \in \mathbb{N}$ be functions of a security parameter $\lambda \in \mathbb{N}$. Let $\text{AHF} : \{0,1\}^\ell \to \{0,1\}^L$ be an efficiently computable function. For every $K \in \{0,1,\bot\}^L$, let the partitioning function $P_K : \{0,1\}^\ell \to \{0,1\}$ such that

$$P_K(X) := \begin{cases} 0 & \forall i \in [L] \quad (\text{AHF}(X)_i = K_i) \lor (K_i = \bot) \\ 1 & \text{otherwise} \end{cases}$$
We say that AHF is a balanced admissible hash function if there exists an efficient algorithm \( \text{AdmSmp}(1^\lambda, Q, \delta) \) that takes as input \( Q \in \text{poly}(\lambda) \) and a non-negligible \( \delta(\lambda) \in (0, 1) \) and outputs a key \( K \in \{0, 1\}^L \) such that, for all \( X^{(1)}, \ldots, X^{(Q)}, X^* \in \{0, 1\}^\ell \) such that \( X^* \notin \{X^{(1)}, \ldots, X^{(Q)}\} \), we have

\[
\gamma_{\text{max}}(\lambda) \geq \Pr_K \left[ P_K(X^{(1)}) = \cdots = P_K(X^{(Q)}) = 1 \land P_K(X^*) = 0 \right] \geq \gamma_{\text{min}}(\lambda),
\]

where \( \gamma_{\text{max}}(\lambda) \) and \( \gamma_{\text{min}}(\lambda) \) are functions such that

\[
\tau(\lambda) = \gamma_{\text{min}}(\lambda) \cdot \delta(\lambda) - \frac{\gamma_{\text{max}}(\lambda) - \gamma_{\text{min}}(\lambda)}{2}
\]

is a non-negligible function of \( \lambda \).

Intuitively, the condition that \( \tau(\lambda) \) be non-negligible requires \( \gamma_{\text{min}}(\lambda) \) to be noticeable and the difference of \( \gamma_{\text{max}}(\lambda) - \gamma_{\text{min}}(\lambda) \) to be small.

It is known [48] that balanced admissible hash functions exist for \( \ell, L = \Theta(\lambda) \).

**Theorem 2.7 ([48, Theorem 1])**. Let \( (C_\ell)_{\ell \in \mathbb{N}} \) be a family of codes \( C_\ell : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell \) with minimal distance \( c : L \) for some constant \( c \in (0, 1/2) \). Then, \( (C_\ell)_{\ell \in \mathbb{N}} \) is a family of balanced admissible hash functions. Furthermore, \( \text{AdmSmp}(1^\lambda, Q, \delta) \) outputs a key \( K \in \{0, 1\}^\ell \) for which \( \eta = \left\lfloor \frac{\ln(2Q + Q/\delta)}{\ln((1-c)^c)} \right\rfloor \) components are not \( \perp \) and \( \gamma_{\text{max}} = 2^{-\eta}, \gamma_{\text{min}} = (1 - Q(1-c))^{\eta} \cdot 2^{-\eta} \), so that \( \tau = (2\delta - (2\delta + 1) \cdot Q \cdot (1-c)^\eta)/2^{\eta+1} \) is a non-negligible function of \( \lambda \).

**Lemma 2.8 ([50, Lemma 8],[6, Lemma 28])**. Let \( K \leftarrow \text{AdmSmp}(1^\lambda, Q, \delta) \), an input space \( X \) and the mapping \( \gamma \) that maps a \( (Q + 1) \)-uple \( (X^*, X_1, \ldots, X_Q) \) in \( X^{Q+1} \) to a probability value in \([0, 1]\), given by:

\[
\gamma(X^*, X_1, \ldots, X_Q) := \Pr_K \left[ P_K(X^{(1)}) = \cdots = P_K(X^{(Q)}) = 1 \land P_K(X^*) = 0 \right].
\]

We consider the following experiment where we first execute the PRF security game, in which the adversary eventually outputs a guess \( b \in \{0, 1\} \) of the challenger’s bit \( b \in \{0, 1\} \) and wins with advantage \( \varepsilon \). We denote by \( X^* \in X \) the challenge input and \( X_1, \ldots, X_Q \in X \) the evaluation queries. At the end of the game, we flip a fair random coin \( b'' \leftarrow U(\{0, 1\}) \). If the condition \( P_K(X^{(1)}) = \cdots = P_K(X^{(Q)}) = 1 \land P_K(X^*) = 0 \) is satisfied we define \( b'' = \hat{b} \). Otherwise, we define \( b'' = b'' \). Then, we have \( | \Pr[b' = b] - 1/2 | \geq \gamma_{\text{min}} \cdot \varepsilon - \gamma_{\text{max}} - \gamma_{\text{min}} \), where \( \gamma_{\text{min}} \) and \( \gamma_{\text{max}} \) are the maximum and minimum of \( \gamma(X) \) for any \( X \in X^{Q+1} \).

### 2.3 Randomness Extraction

The Leftover Hash Lemma was used by Agrawal et al. [6] to re-randomize matrices over \( \mathbb{Z}_q \) by multiplying them with small-norm matrices.

**Lemma 2.9 ([6])**. Let integers \( m, n \) such that \( m > 2n \cdot \log q \), for some prime \( q > 2 \). Let \( A, U \leftarrow U(\mathbb{Z}_q^{n \times m}) \) and \( R \leftarrow U(\{-1, 1\}^{m \times m}) \). The distributions \( (A, AR) \) and \( (A, U) \) are within \( 2^{-\Omega(n)} \) statistical distance.
2.4 Multi-Client Functional Encryption

We recall the syntax of multi-client functional encryption as introduced in [44].

Definition 2.10. A multi-client functional encryption (MCFE) scheme for a message space $\mathcal{M}$ and tag space $\mathcal{T}$ is a tuple $(\text{Setup}, \text{Encrypt}, \text{DKeygen}, \text{Decrypt})$ of efficient algorithm with the following specifications:

- **Setup**($\text{cp}, 1^\ell$): Takes in global parameters $\text{cp}$ and a pre-determined number of users $1^\ell$, where $\text{cp}$ specifies a security parameter $1^\lambda$. It outputs a set of public parameters $\text{mpk}$, a master secret key $\text{msk}$, and a set of encryption keys $\{\text{ek}_i\}_{i=1}^\ell$. We assume that $\text{mpk}$ is included in all encryption keys $\text{ek}_i$.

- **Encrypt**($\text{ek}_i, x_i, t$): Takes as input the encryption key $\text{ek}_i$ of user $i \in [\ell]$, a message $x_i$, and a tag $t \in \mathcal{T}$. It output a ciphertext $C_{t,i}$.

- **DKeygen**($\text{msk}, f$): Takes as input the master secret key $\text{msk}$ and an $\ell$-argument function $f : \mathcal{M}^\ell \rightarrow \mathcal{R}$. It outputs a functional decryption key $\text{dk}_f$.

- **Decrypt**($\text{dk}_f, t, C$): Takes as input a functional decryption key $\text{dk}_f$, a tag $t$, and an $\ell$-vector of ciphertexts $C = (C_{t,1}, \ldots, C_{t,\ell})$. It outputs a function evaluation $f(x) \in \mathcal{R}$ or an error message $\perp$.

**Correctness.** For any set of public parameters $\text{cp}$, any $(\text{mpk}, \text{msk}, \{\text{ek}_i\}_{i=1}^{\ell}) \leftarrow \text{Setup}(\text{cp}, 1^\ell)$, any vector $x \in \mathcal{M}^\ell$ any tag $t \in \mathcal{T}$ and any function $f : \mathcal{M}^\ell \rightarrow \mathcal{R}$, if $C_{t,i} \leftarrow \text{Encrypt}(\text{ek}_i, x_i, t)$ for all $i \in [\ell]$ and $\text{dk}_f \leftarrow \text{DKeygen}(\text{msk}, f)$, we have $\text{Decrypt}(\text{dk}_f, t, C_{t}) = (C_{t,1}, \ldots, C_{t,\ell}) = f(x)$ with overwhelming probability.

We now recall the security definition given in [44] for an adaptively secure MCFE, and then we will give the definition that we use in this work. These two definitions are in fact equivalent.

Definition 2.11 (IND-sec). For an MCFE scheme with $\ell$ senders, consider the following game between an adversary $A$ and a challenger $C$. The game involves a set $\mathcal{HS}$ of honest senders (initialized to $\mathcal{HS} := [\ell]$) and a set $\mathcal{CS}$ (initialized to $\mathcal{CS} := \emptyset$) of corrupted senders.

- **Initialization:** The challenger $C$ chooses $\text{cp}$ and runs $(\text{mpk}, \text{msk}, \{\text{ek}_i\}_{i=1}^{\ell}) \leftarrow \text{Setup}(\text{cp}, 1^\ell)$. Then, it chooses a random bit $b \leftarrow \{0,1\}$ and gives the master public key $\text{mpk}$ to the adversary.

- **Encryption queries:** The adversary $A$ can adaptively make encryption queries $Q\text{Encrypt}(i, x^0, x^1, t)$, to which the challenger replies with $\text{Encrypt}(\text{ek}_i, x^b, t)$. For any given pair $(i,t)$, only one query is allowed and subsequent queries involving the same $(i,t)$ are ignored.

- **Functional decryption key queries:** The adversary can adaptively obtain functional decryption keys by making queries of the form $Q\text{DKeygen}(f)$. The challenger returns $\text{dk}_f \leftarrow \text{DKeygen}(\text{msk}, f)$.

- **Corruption queries:** For any user $i \in \mathcal{HS}$, the adversary can adaptively make queries $Q\text{Corrupt}(i)$, to which the challenger replies with $\text{ek}_i$ and updates $\mathcal{HS}$ and $\mathcal{CS}$ by setting $\mathcal{CS} := \mathcal{CS} \cup \{i\}$ and $\mathcal{HS} := \mathcal{HS} \setminus \{i\}$.

- **Finalize:** The adversary makes its guess $b' \in \{0,1\}$; $A$ wins the game if $b = b'$, where $b$ is defined to be $\beta := b'$ except in the following situations.
1. An encryption query $Q\text{Encrypt}(i, x^0, x^1, t)$ has been made for an index $i \in CS$ with $x^0 \neq x^1$.

2. For some label $t$, an encryption query $Q\text{Encrypt}(i, x^0_i, x^1_i, t)$ has been asked for $i \in HS$, but encryption queries $Q\text{Encrypt}(j, x^0_j, x^1_j, t)$ have not been asked for all $j \in HS$.

3. For a label $t$ and some function $f$ queried to $Q\text{DKeygen}$, there exists a pair of vectors $(x^0, x^1)$ such that $f(x^0) \neq f(x^1)$, where
   - $x^0_i = x^1_i$ for all $i \in CS$;
   - $Q\text{Encrypt}(i, x^0_i, x^1_i, t)$ have been asked for all $i \in HS$.

In any of the above cases, $A$’s output is replaced by a random $\beta \leftarrow U\{0, 1\}$.

An MCFE scheme provides IND security if, for any efficient adversary $A$, we have $\text{Adv}^{\text{IND}}(A) := |\Pr[\beta = 1 | b = 1] - \Pr[\beta = 1 | b = 0]| \in \text{neg}(\lambda)$.

In the following, it will be convenient to work with the following security definition, which is equivalent to Definition 2.11.

**Definition 2.12 (1-challenge IND-sec).** For an MCFE scheme with $\ell$ senders, we consider the following game between an adversary $A$ and a challenger $C$. The game involves a set $HS$ (initialized to $HS := [\ell]$), of honest senders and a set $CS$ (initialized to $CS := \emptyset$), of corrupted senders.

**Initialization:** The challenger $C$ generates $cp$ and runs $(\text{mpk}, \text{msk}, \{\text{ek}_i\}_{i=1}^{\ell}) \leftarrow \text{Setup}(cp, 1^\ell)$. Then, it chooses a random bit $b \leftarrow \{0, 1\}$ and gives the master public key $\text{mpk}$ to the adversary $A$.

**Encryption queries:** The adversary can adaptively make encryption queries $Q\text{Encrypt}(i, x, t)$, to which the challenger replies with $\text{Encrypt}(\text{ek}_i, x, t)$. Any further query involving the same pair $(i, t)$ is ignored.

**Challenge queries:** The adversary adaptively makes challenge queries of the form $Q\text{Encrypt}(i, x^0_i, x^1_i, t^*)$. The challenger replies with $\text{Encrypt}(\text{ek}_i, x^0_i, x^1_i, t^*)$. Only one tag $t^*$ can be involved in a challenge query. If $t^*$ denotes the tag of the first query, the challenger only replies to subsequent challenge queries for the same label $t^*$. Moreover, only one query $(i, t^*)$ is allowed for each $i \in [\ell]$ and subsequent queries involving the same $i \in [\ell]$ are ignored.

**Functional decryption key queries:** The adversary can adaptively obtain functional decryption keys via queries $Q\text{DKeygen}(f)$. At each query, the challenger returns $\text{dk}_f \leftarrow \text{DKeygen}(\text{msk}, f)$.

**Corruption queries:** For any user $i \in HS$, the adversary can adaptively make queries $Q\text{Corrupt}(i)$, to which the challenger replies with $\text{ek}_i$ and updates $CS$ and $CS$ by setting $CS := CS \cup \{i\}$ and $HS := HS \setminus \{i\}$.

**Finalize:** The adversary outputs a bit $b' \in \{0, 1\}$. The adversary $A$ wins if $\beta = b'$, where $\beta$ is defined as $\beta := b'$, unless of the situations below occurred.

1. A challenge query $Q\text{Encrypt}(i, x^0_i, x^1_i, t^*)$ has been made for an index $i \in CS$ with $x^0_i \neq x^1_i$.

2. An encryption query $Q\text{Encrypt}(i, x, t^*)$ has been made for the challenge tag $t^*$ for some index $i \in [\ell]$. 

14
We say that an MCFE scheme provides $A$ \textit{adversary} we use the same syntax as Chotard. This allows the adversary to make both an encryption query $\text{Setup} \{S_i\}_{i=1}^{\ell}$ multi-client functional encryption (DMCFE) scheme between $\ell$ senders $\{S_i\}_{i=1}^{\ell}$ and a functional decryptor $FD$ is specified by the following components.

\begin{itemize}
  \item \textbf{Setup}(cp, $t^\ell$): This is an interactive protocol between the senders $\{S_i\}_{i=1}^{\ell}$, which allows them to generate their own secret keys $sk_i$ and encryption keys $ek_i$, for $i \in [\ell]$, as well as a set of public parameters $mpk$.
\end{itemize}

3. For the challenge tag $t^*$, a challenge query $\text{CQEncrypt}(i, x_i^{*0}, x_i^{*1}, t^*)$ has been asked for some $i \in HS$, but challenge queries $\text{CQEncrypt}(j, x_j^{*0}, x_j^{*1}, t^*)$ have not been asked for all $j \in HS$.

4. For the challenge tag $t^*$ and some function $f$ queried to $\text{QDKeygen}$, there exists a pair of vectors $(x_i^{*0}, x_i^{*1})$ such that $f(x_i^{*0}) \neq f(x_i^{*1})$, where
- $x_i^{0} = x_i^{1}$ for all $i \in CS$,
- $\text{CQEncrypt}(i, x_i^{*0}, x_i^{*1}, t^*)$ have been asked for all $i \in HS$.

If any of these events occurred, $A$’s output is overwritten by $\beta \leftarrow U(\{0, 1\})$.

We say that an MCFE scheme provides $1\text{Ch-IND}$ security if, for any efficient adversary $A$, we have $\text{Adv}^{1\text{Ch-IND}}(A) := |\text{Pr}[\beta = b] - \frac{1}{2}| \in \text{negl}(\lambda)$.

In Appendix B, we show that $1\text{Ch-IND}$ security implies $\text{IND}$ security. We also note that condition 2 of “Finalize” could be:

2'. Both $\text{QEncrypt}(i, x, t^*)$ and $\text{CQEncrypt}(i, x_i^{*0}, x_i^{*1}, t^*)$ have been made for an index $i$ and the challenge label $t^*$, such that $x_i^{*0} \neq x_i^{*1}$

This allows the adversary to make both an encryption query $\text{QEncrypt}(i, x, t^*)$ and a challenge query $\text{CQEncrypt}(i, x_i^{*0}, x_i^{*1}, t^*)$ where $x_i^{*0} = x_i^{*1}$. In Proposition B.2 in Appendix, we show that replacing condition 2 by condition 2' does not make the adversary any stronger.

Our first construction is proven secure under Definition 2.12. Abdalla et al. [2] and Chotard et al. [29] independently showed constructions that can be proven secure in the sense of a stronger definition which eliminates restriction 3 from the “Finalize” stage. In Appendix A, we show that a variant of the compiler of [2, Section 4.2] is secure in the standard model. Recently, Abdalla et al. [1] independently obtained a similar result. While their PRF-based compiler [1] can rely on any PRF, we obtain a tighter reduction using a specific PRF described in [56]. Chotard et al. [29] additionally show how to enable repetitions by allowing multiple encryption queries for the same pair $(i, t)$. However, they need random oracles for this purpose.

2.5 Decentralized Multi-Client Functional Encryption

We use the same syntax as Chotard et al. [28] with the difference that we explicitly assume common public parameters $\text{cp}$. As in [28], we assume that each function $f$ can be injectively encoded as a tag $t_f$ (called “label” in [28]) taken as input by the partial functional key generation algorithm.

\begin{definition}
For a message space $M$ and tag space $T$, a decentralized multi-client functional encryption (DMCFE) scheme between $\ell$ senders $\{S_i\}_{i=1}^{\ell}$ and a functional decryptor $FD$ is specified by the following components.

\begin{itemize}
  \item \textbf{Setup}(cp, $t^\ell$)
\end{itemize}

This is an interactive protocol between the senders $\{S_i\}_{i=1}^{\ell}$, which allows them to generate their own secret keys $sk_i$ and encryption keys $ek_i$, for $i \in [\ell]$, as well as a set of public parameters $mpk$.
Correctness. For any $\lambda \in \mathbb{N}$, any $(\text{mpk},\{\text{sk}_i\}_{i=1}^\ell,\{\text{ek}_i\}_{i=1}^\ell) \leftarrow \text{Setup}(\text{cp},1^\lambda)$, any $x \in \mathcal{M}^\lambda$, any tag $t \in \mathcal{T}$ and any function $f : \mathcal{M}^\ell \rightarrow \mathcal{R}$, if $C_{t,i} \leftarrow \text{Encrypt}(\text{ek}_i, x_i, t)$ for all $i \in [\ell]$ and $\text{dk}_f \leftarrow \text{DKeygenComb}\left(\{\text{DKeygenShare}(\text{sk}_i,t_f)\}_{i},t_f\right)$, with overwhelming probability, we have $\text{Decrypt}\left(\text{dk}_f,t,C\right) = f(x)$.

Definition 2.14 (IND-sec for DMCFE). For a DMCFE scheme with $\ell$ senders, we consider the following game between an adversary and a challenger. It involves a set $\mathcal{HS}$ of honest senders (initialized to $\mathcal{HS} := [\ell]$) and a set $\mathcal{CS}$ (initialized to $\mathcal{CS} := \emptyset$) of the corrupted senders.

**Initialization:** The challenger $\mathcal{C}$ generates $\text{cp}$ and runs $(\text{mpk},\{\text{sk}_i\}_{i=1}^\ell,\{\text{ek}_i\}_{i=1}^\ell) \leftarrow \text{Setup}(\text{cp},1^\lambda)$. Then, it flips a fair coin $b \leftarrow \{0,1\}$ and gives the master public key mpk to the adversary $A$.

**Encryption queries:** The adversary $A$ can adaptively make encryption queries $\text{QEncrypt}(i,x^0_i,x^1_i,t)$, to which the challenger replies with $\text{Encrypt}(\text{ek}_i,x_i^b,t)$. For any given pair $(i,t)$, only one query is allowed and subsequent queries involving the same $(i,t)$ are ignored.

**Functional decryption key queries:** Via queries $\text{QDKeygen}(i,f)$, $A$ can adaptively obtain partial functional decryption keys on behalf of uncorrupted senders. At each query, the challenger returns $\text{dk}_f \leftarrow \text{DKeygenShare}(\text{sk}_i,t_f)$ if $i \in \mathcal{HS}$ (if $i \in \mathcal{CS}$, the oracle returns $\perp$).

**Corruption queries:** For any user $i \in \mathcal{HS}$, the adversary can adaptively make queries $\text{QCorrupt}(i)$ and the challenger replies by returning $(\text{sk}_i,\text{ek}_i)$. It also updates the sets $\mathcal{HS}$ and $\mathcal{CS}$ by setting $\mathcal{CS} := \mathcal{CS} \cup \{i\}$ and $\mathcal{HS} := \mathcal{HS} \setminus \{i\}$.

**Finalize:** The adversary outputs a bit $b' \in \{0,1\}$. The adversary $A$ wins if $b = b'$, where $b$ is defined as $b := b'$, unless of the situations below occurred.

1. An encryption query $\text{QEncrypt}(i,x^0_i,x^1_i,t)$ has been made for an index $i \in \mathcal{CS}$ with $x^0_i \neq x^1_i$.
2. For some label $t$, an encryption query $\text{QEncrypt}(i,x^0_i,x^1_i,t)$ has been asked for $i \in \mathcal{HS}$, but encryption queries $\text{QEncrypt}(j,x^0_j,x^1_j,t)$ have not been asked for all $j \in \mathcal{HS}$.
3. For a tag $t$ and some function $f$ queried to QDKeygen$(i, \cdot)$ for all $i \in HS$, there exists a pair of vectors $(x^0, x^1)$ such that $f(x^0) \neq f(x^1), \text{where}$

- $x^0_i = x^1_i$ for all $i \in CS$;
- $QEncrypt(i, x^0_i, x^1_i, t)$ have been asked for all $i \in HS$.

If any of these events occurred, $A$’s output is overwritten by $\beta \leftarrow U\{0, 1\}$.

We say that a DMCFE scheme provides IND security if, for any efficient adversary $A$, we have $\text{Adv}^{\text{IND}}_A := |\Pr[\beta = 1 | b = 1] - \Pr[\beta = 1 | b = 0]| \in \text{negl}(\lambda)$.

The above definition captures adaptive corruptions in that the QCorrupt$(\cdot)$ oracle may be invoked at any time during the game. In the static corruption setting, all queries to QCorrupt$(\cdot)$ should be made at once before the initialization phase. In this case, the sets $HS$ and $CS$ are thus determined before the generation of $(mpk, \{sk_i\}_{i=1}^L, \{ek_i\}_{i=1}^L)$. We denote by sta-IND-sec the latter security game.

Our scheme of Section 4 will be proven secure under static corruptions. We insist that only corruptions are static: the encryption oracle can be queried on adaptively chosen messages $(x^0, x^1)$, which is stronger than the selective security game, where the challenge messages have to be declared upfront.

3 Our MCFE Scheme for Linear Functions

The scheme encrypts $x_i \in \mathbb{Z}^n$ as a vector $C_{i,i} = G_0^T \cdot x_i + A(\tau)^T \cdot s_i + e_i$, where $G_0$ is a gadget matrix; $\tau = AHF(t) \in \{0, 1\}^L$ is an admissible hash of the tag $t$; and $e_i$ is a Gaussian noise. This is done in a way that a functional secret key $s_i = \sum_{i=1}^{\ell} y_i \cdot s_i \in \mathbb{Z}^n$ allows computing $\sum_{i=1}^{\ell} y_i \cdot x_i$ from $\{C_{i,i}\}_{i=1}^\ell$ by using the public trapdoor of the lattice $\Lambda^T(G_0)$.

We derive $A(\tau)$ from a set of $2L$ public matrices $\{A_{i,0}, A_{i,1}\}_{i=1}^L$ and an additional matrix $V \in \mathbb{Z}_q^{n \times n}$. Like [56], our proof interprets each $A_{i,b} \in \mathbb{Z}_q^{n \times m}$ as a GSW ciphertext $i-b = A \cdot R_{i,b} + \mu_{i,b} \cdot G$, where $R_{i,b} \in \{-1, 1\}^{m \times m}$, $\mu_{i,b} \in \{0, 1\}$ and $G \in \mathbb{Z}_q^{m \times m}$ is the gadget matrix of $[58]$. Then, we homomorphically compute $A(\tau)$ as an FHE ciphertext $A \cdot R_{\tau} + (\prod_{i=1}^L \mu_{i,\tau[i]} \cdot G)$, for some small-norm $R_{\tau} \in \mathbb{Z}^{m \times m}$, which in turn multiplied by $G^{-1}(V^T \cdot G_0)$ in such a way that $A(\tau) = A \cdot R_{\tau} + (\prod_{i=1}^L \mu_{i,\tau[i]} \cdot (V^T \cdot G_0))$. Via a careful choice of $\{\mu_{i,b}\}_{i \in [L], b \in \{0, 1\}}$, the properties of admissible hash functions imply that $\prod_{i=1}^L \mu_{i,x[i]}$ vanishes in all encryption queries but evaluates to 1 on the challenge tag $\tau$. In order to prevent the encryption oracle from leaking too much about $s_i \in \mathbb{Z}^n$, we proceed as in [56] and replace the random $A \in \mathbb{Z}_q^{n \times m}$ by a lossy matrix $A^T = \hat{A}^T \cdot C + E$, where $\hat{A} \leftarrow U(\mathbb{Z}_q^{n_1 \times m}), C \leftarrow U(\mathbb{Z}_q^{n_1 \times n})$ and for a small-norm $E \in \mathbb{Z}^{m \times n}$.

Our construction and proof depart from [56] in that we use an additional multiplication by $G^{-1}(V^T \cdot G_0)$ in order to introduce a matrix $V \in \mathbb{Z}_q^{n \times n}$ in the expression of $A(\tau)$. In addition, unlike [56], we do not rely on a randomness extraction argument to exploit the entropy of $A(\tau)^T \cdot s_i + e_i$ in the challenge phase. Instead, we use a trapdoor for the matrix $U = [X]_1$ to “equivocate” the challenge ciphertexts and explain them as an encryption of $x^1_{0,i}$ instead of $x^0_{0,i}$.17
Another difference with [56] is that the product \( \mathbf{A}(\tau) \) of GSW ciphertexts \( \{ \mathbf{A}_{i,\tau[i]} \}_{i=1}^{L} \) is evaluated in a sequential manner\(^7\) (as in the “right-spine” PRF construction of [12]) in order for the noise matrix \( \mathbf{R}_{\tau} \) to retain small entries.

We note that, as was suggested by [56, Section 4.3] in the context of distributed PRFs, a technique due to Yamada [70] can be used to compute \( \mathbf{A}(\tau) \) as a function of \( \Theta(\log^2 \lambda) \) (instead of \( \Theta(\lambda) \)) public matrices. For simplicity, we chose to describe the scheme using public parameters containing a linear number of matrices here.

### 3.1 Description

In the following description, we assume public parameters

\[
\text{cp} := \left( \lambda, \ell_{\max}, X, Y, n_0, n_1, n, m, \alpha, \alpha_1, \sigma, \ell_t, L, q, \text{AHF} \right),
\]

consisting of a security parameter \( \lambda \) and the following quantities:

- \((X, Y, \ell_{\max}, n_0, n_1, n, m)\), which are all in \( \text{poly}(\lambda) \)

\[
X = 1, \ n_1 = \lambda^d, \ q = 2^{\lambda^d - 1}, \ \alpha = 2^{d^2 - 1}, \ \alpha_1 = 2^{\lambda^d + d \log \lambda}, \ n_0 = o(\lambda^d - 2),
\]

\[
n = O(\lambda^{d-1}), \ \sigma = 2^{d^2 - 2}, \ \text{and} \ \ n_0 \cdot \ell_{\max} = O(\lambda^d - 2) \quad \text{where} \ d \ \text{is a constant;}
\]

- The description of a balanced admissible hash function \( \text{AHF} : \{0,1\}^{\ell_t} \rightarrow \{0,1\}^{L} \), for a suitable parameter \( L \in \Theta(\lambda) \).

- The description of a tag space \( T = \{0,1\}^{\ell_t} \), for some \( \ell_t \in \text{poly}(\lambda) \), such that tags may be arbitrary strings (e.g., time period numbers or dataset names).

- The description of a balanced admissible hash function \( \text{AHF} : \{0,1\}^{\ell_t} \rightarrow \{0,1\}^{L} \), for a suitable parameter \( L \in \Theta(\lambda) \).

- The description of a tag space \( T = \{0,1\}^{\ell_t} \), for some \( \ell_t \in \text{poly}(\lambda) \), such that tags may be arbitrary strings (e.g., time period numbers or dataset names).

- A real \( \alpha > 0 \) and a Gaussian parameter \( \sigma > 0 \), which specifies an interval \([\beta, \beta] = [\sigma \sqrt{n}, \sigma \sqrt{n}]\) where the coordinates of users’ secret keys will live (with probability exponentially close to 1).

Letting \( \ell \in \text{poly}(\lambda) \), with \( \ell \leq \ell_{\max} \), be the number of users, our function space is the set of all functions \( f_y : \mathbb{Z}_{n_0}^{\ell} \rightarrow \mathbb{Z}_{n_0}^{n_0} \) indexed by an integer vector \( y \in \mathbb{Z}^{\ell} \) of infinity norm \( \| y \|_\infty < Y \).

We define \( \mathbf{G}_0 \in \mathbb{Z}_{q}^{n_0 \times m} \) to be the gadget matrix

\[
\mathbf{G}_0 = [\mathbf{I}_{n_0} \otimes (1, 2, 4, \ldots, 2^{\log q})] \ | \ 0^{n_0} \ | \ldots \ | \ 0^{n_0} \in \mathbb{Z}_{q}^{n_0 \times m}
\]

where the product \( \mathbf{I}_{n_0} \otimes (1, 2, 4, \ldots, 2^{\log q}) \) is padded with \( m - n_0 \cdot \log q \) zero columns. We similarly denote by \( \mathbf{G} \in \mathbb{Z}_{q}^{n \times m} \) the gadget matrix of rank \( n \):

\[
\mathbf{G} = [\mathbf{I}_{n} \otimes (1, 2, 4, \ldots, 2^{\log q})] \ | \ 0^{n} \ | \ldots \ | \ 0^{n} \in \mathbb{Z}_{q}^{n \times m}
\]

Our MCFE construction goes as follows.

\(^7\) In [56], the multiplication of ciphertexts \( \{ \mathbf{A}_{i,\tau[i]} \}_{i=1}^{L} \) was computed in a parallel fashion \( \mathbf{A}_0 \cdot \prod_{i=1}^{L} \mathbf{G}^{-1}(\mathbf{A}_{i,\tau[i]}) \) because their initial proof required the matrices \( \{ \mathbf{A}_{i,b} \}_{i,b} \) to be generated in such a way that \( \mathbf{G}^{-1}(\mathbf{A}_{i,b}) \) was invertible over \( \mathbb{Z}_q \).
Setup(cp, 1^l): On input of cp and a number of users ℓ, do the following.

1. Choose random matrices \( A_{i,b} \leftarrow U(\mathbb{Z}_q^{n \times m}) \), for each \( i \in [L] \), \( b \in \{0,1\} \).
2. Choose a uniformly random matrix \( V \leftarrow U(\mathbb{Z}_q^{n \times n}) \).
3. For each \( i \in [ℓ] \), sample \( s_i \leftarrow D_{\mathbb{Z}_q^s} \) and define \( e_{k_i} = s_i \in \mathbb{Z}^n \).

Output the master secret key \( msk := \{ e_{k_i} \}_{i=1}^ℓ \) and the public parameters

\[
mpk := (cp, V, \{ A_{i,0}, A_{i,1} \in \mathbb{Z}_q^{n \times m} \}_{i=1}^L).
\]

DKeygen(msk, \( f_y \)): Given the master secret key \( msk := \{ e_{k_i} \}_{i=1}^ℓ \) and a linear function \( f_y : \mathbb{Z}_q^{n_0 \times ℓ} \rightarrow \mathbb{Z}_q^n \) defined by an integer vector \( y = (y_1, \ldots, y_ℓ)^T \in \mathbb{Z}^ℓ \) which maps an input \( X = [x_1 | \ldots | x_ℓ] \in \mathbb{Z}_q^{n_0 \times ℓ} \) to \( f_y(X) = X \cdot y \in \mathbb{Z}_q^{n_0} \), parse each \( e_{k_i} \) as a vector \( s_i \in \mathbb{Z}^n \). Then, compute and output the functional secret key \( d_{k_y} := (y, s_y) \), where \( s_y = \sum_{i=1}^ℓ y_i s_i \in \mathbb{Z}^n \).

Encrypt(\( e_{k_i}, x_i, t \)): Given \( e_{k_i} = s_i \in \mathbb{Z}^n \), \( x_i \in [-X,X]^{n_0} \), and \( t \in \{0,1\}^ℓ \),

1. Compute \( τ = AHF(t) \in \{0,1\}^L \) and parse it as \( τ = τ[1] \ldots τ[L] \).
2. Define \( W = G_0^T \cdot V \in \mathbb{Z}_q^{m \times n} \) and compute

\[
A(τ) = A_{L,τ[L]} \cdot G^{-1} \left( A_{L-1,τ[L-1]} \cdot G^{-1} \left( \ldots A_2,τ[2] \right) \cdot G^{-1} \left( A_1,τ[1]\right) \right)
\]

\[
\cdot G^{-1}(W^T) \in \mathbb{Z}_q^{n \times m}.
\]

3. Sample a noise vector \( e_i \leftarrow D_{\mathbb{Z}_q[m \cdot q]} \). Then, compute and output

\[
C_{t,i} = G_0^T \cdot x_i + A(τ)^T \cdot s_i + e_i \in \mathbb{Z}_q^m.
\]

Decrypt(\( d_{k_y}, t, C_t \)): On input of a functional secret key \( d_{k_y} = (y, s_y) \) for a vector \( y = (y_1, \ldots, y_ℓ)^T \in [-Y,Y]^ℓ \), a tag \( t \in \{0,1\}^ℓ \), and an \( ℓ \)-vector of ciphertexts \( C_t = (C_{t,1}, \ldots, C_{t,ℓ}) \in (\mathbb{Z}_q^m)\), conduct the following steps.

1. Compute \( τ = AHF(t) \in \{0,1\}^L \) and parse it as \( τ = τ[1] \ldots τ[L] \).
2. Compute \( A(τ) \in \mathbb{Z}_q^{n \times m} \) as per (2).
3. Compute \( f_{t,y} = \sum_{i=1}^ℓ y_i \cdot C_{t,i} - A(τ)^T \cdot s_y \mod q \).
4. Interpret \( f_{t,y} \in \mathbb{Z}_q^m \) as a vector of the form \( f_{t,y} = G_0^T \cdot z + \tilde{e} \mod q \), for some error vector \( \tilde{e} \in [-B,B]^m \). Using the public trapdoor of \( A_1(G_0) \), compute and output the underlying vector \( z \in [-ℓ \cdot X \cdot Y, ℓ \cdot X \cdot Y]^n_0 \).

The following lemma is proved in Appendix C.1.

Lemma 3.1 (Correctness). Assume that \( m = \omega(\sqrt{\log ℓ}) \), \( Y \cdot ℓ \cdot αq \cdot \log q < q/2 \) and \( ℓ \cdot X \cdot Y < q/2 \). Then, for any \( (mpk, msk, \{ e_{k_i} \}_{i=1}^ℓ) \leftarrow \text{Setup}(cp, 1^λ) \), any message \( X = [x_1 | \ldots | x_ℓ] \in [-X,X]^{n_0 \times ℓ} \), any \( y \in [-Y,Y]^ℓ \), any tag \( t \in \{0,1\}^ℓ \), algorithm Decrypt(\( d_{k_y}, t, C_t \)) outputs \( X \cdot y \in \mathbb{Z}_q^{n_0} \) with probability exponentially close to 1, where \( C_{t,i} \leftarrow \text{Encrypt}(e_{k_i}, x_i, t) \) and \( d_{k_y} \leftarrow \text{DKeygen}(msk, f_y) \).
3.2 Security

We now prove the security of the scheme in the sense of Definition 2.12 (and thus Definition 2.11 modulo some loss of tightness in the reduction).

For the current parameters $n_1 = \lambda^d$, $q = 2^{\lambda^{d-1}}$, and $\alpha_1 = 2^{-\lambda^{d-1}+d\log \lambda}$, $\alpha_1 q = \Omega(\sqrt{n_1})$, we know from [62] that \text{LWE}_{q,n_1,\alpha_1} is at least as hard as \text{GapSVP}_\gamma, with $\gamma = \tilde{O}(n_1/\alpha_1) = \tilde{O}(2^{\lambda^{d-1}})$. The best known algorithms [65] for solving \text{GapSVP}_\gamma run in $2^{\tilde{O}(\sqrt{d})}$, which for our parameters is $2^{\tilde{O}(\lambda)}$.

**Theorem 3.2.** The above MCFE schemes provides adaptive security under the \text{LWE}_{q,m,n_1,\alpha_1} assumption.

**Proof.** The proof considers a sequence of games. In each game, we denote by $W_i$ the event that $b' = b$. For each $i$, the adversary’s advantage function in Game$_i$ is $\text{Adv}_i(A) := |\Pr[b' = b] - 1/2| = \frac{1}{2} \cdot |\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$.

**Game$_0$:** This is the real security game. We denote by $t^*$ the tag of the challenge phase while $t(1), \ldots, t(Q)$ are the tags involved in encryption queries. Namely, for each $j \in [Q]$, $t(j)$ stands for the $j$-th distinct tag involved in an encryption query. Since up to $\ell$ encryption queries $(i, x_i, t)$ are allowed for each tag $t$, the adversary can make a total of $\ell \cdot Q$ encryption queries. The game begins with the challenger initially choosing encryption keys $\{e_k\}_{i=1}^t$ by sampling $e_k = s_i \leftarrow D_{Z^{m},\sigma}$ for each $i \in [t]$. In addition, the challenger flips a fair coin $b \leftarrow U([0,1])$ which will determine the response to challenge queries. At each encryption query each $i \in [t]$, the adversary obtains $e_k$ and the challenger updates a set $\mathcal{CS} := \mathcal{CS} \cup \{i\}$, which is initially empty. At each encryption query $(i, x_i^{(j)}, t^{(j)})$, the challenger samples $e_i^{(j)} \leftarrow D_{Z^{m},\sigma}$ and returns $C_{t,i} = G_0^T \cdot x_i^{(j)} + A(\tau^{(j)})^T \cdot s_i + e_i^{(j)} \in Z_q^m$, where $\tau^{(j)} = \text{AHF}(t^{(j)})$. In the challenge phase, the adversary $\mathcal{A}$ chooses a fresh tag $t^*$ and two vectors of messages $X_0^* = [x_{0,1}^* \ldots x_{0,t}^*] \in [-X,X]^n \times \ell$ and $X_1^* = [x_{1,1}^* \ldots x_{1,t}^*] \in [-X,X]^n \times \ell$ subject to the constraint that, for any private key query $y \in [-Y,Y]^t$ made by $\mathcal{A}$, we must have $X_0^* \cdot y = X_1^* \cdot y$ over $\mathbb{Z}$. In addition, the invariant that $x_{0,i}^* = x_{1,i}^*$ for any $i \in \mathcal{CS}$ must be satisfied at any time during the game. In response to a challenge query $(i, x_{0,i}^*, x_{1,i}^*, t^*)$, the challenger generates a challenge ciphertext $C_{t^*,i}$, where

$$C_{t^*,i} = G_0^T \cdot x_{b,i}^* + A(\tau^*)^T \cdot s_i + e_i^*,$$

where $\tau^* = \text{AHF}(t^*)$ and $e_i^* \leftarrow D_{Z^{m},\sigma}$ for all $i \in [t]$.

When $\mathcal{A}$ halts, it outputs $\hat{b} \in \{0,1\}$ and the challenger defines $b' := \hat{b}$. We have $\text{Adv}(A) := |\Pr[W_0] - 1/2|$, where $W_0$ is event that $b' = b$.

**Game$_i$:** This game is identical to Game$_0$ except for the following changes. First, the challenger runs $K \leftarrow \text{AdmSmp}(1^\lambda, Q, \delta)$ to generate a key $K \in \{0,1,\perp\}^L$
for a balanced admissible hash function $\text{AHF} : \{0, 1\}^L \rightarrow \{0, 1\}^L$. When the adversary halts and outputs $\hat{b} \in \{0, 1\}$, the challenger checks if the conditions

$$
P_K(t^{(1)}) = \cdots = P_K(t^{(Q)}) = 1 \land P_K(t^*) = 0 \label{eq:4}
$$

are satisfied. If conditions (4) do not hold, the challenger ignores $A$’s output $\hat{b} \in \{0, 1\}$ and overwrites it with a random bit $b'' \leftrightarrow \{0, 1\}$ to define $b' = b''$. If conditions (4) are satisfied, the challenger sets $b' = \hat{b}$. By Lemma 2.8,

$$
| \Pr[W_1] - 1/2 | \geq \gamma_{\min} \cdot \text{Adv}(A) - \frac{1}{2} \cdot (\gamma_{\max} - \gamma_{\min}) = \tau,
$$

where $\tau(\lambda)$ is a noticeable function.

**Game 2:** In this game, we modify the generation of $\text{mpk}$ in the following way. Initially, the challenger samples a uniformly random matrix $A \leftarrow U(\mathbb{Z}^{n \times m}_q)$. Next, for each $i \in [L]$, it samples $R_{i,0}, R_{i,1} \leftarrow U([-1, 1])^{m \times m}$ and defines $\{A_{i,0}, A_{i,1}\}_{i=1}^L$ as follows for all $i \in [L]$ and $j \in \{0, 1\}$:

$$A_{i,j} := \begin{cases} A \cdot R_{i,j} & \text{if } (j \neq K_i) \land (K_i \neq \perp) \\ A \cdot R_{i,j} + G & \text{if } (j = K_i) \lor (K_i = \perp) \end{cases} \label{eq:5}
$$

Since $A \in \mathbb{Z}^{n \times m}_q$ was chosen uniformly, the Leftover Hash Lemma ensures that $\{A_{i,0}, A_{i,1}\}_{i=1}^L$ are statistically independent and uniformly distributed over $\mathbb{Z}^{n \times m}_q$. It follows that $| \Pr[W_2] - \Pr[W_1] | \leq L \cdot 2^{-\lambda}$.

We note that, at each encryption query $(i, x^{(j)}_i, t^{(j)})$, the admissible hash function maps $t^{(j)}$ to $\tau^{(j)} = \text{AHF}(t^{(j)})$, which is itself mapped to a GSW encryption

$$A(\tau^{(j)}) = A \cdot R_{\tau^{(j)}} + (\prod_{i=1}^L \mu_i) \cdot W^T, \label{eq:6}
$$

of a product $\prod_{i=1}^L \mu_i$, for some small norm matrix $R_{\tau^{(j)}} \in \mathbb{Z}^{m \times m}$, where

$$
\mu_i := \begin{cases} 0 & \text{if } (\text{AHF}(t^{(j)})_i \neq K_i) \land (K_i \neq \perp) \\ 1 & \text{if } (\text{AHF}(t^{(j)})_i = K_i) \lor (K_i = \perp) \end{cases}
$$

If conditions (4) are satisfied, at each encryption query $(i, x^{(j)}_i, t^{(j)})$, the admissible hash function ensures that $\tau^{(j)} = \text{AHF}(t^{(j)})$ satisfies

$$A(\tau^{(j)}) = A \cdot R_{\tau^{(j)}}, \quad \forall j \in [Q], \label{eq:7}
$$

for some small norm $R_{\tau^{(j)}} \in \mathbb{Z}^{m \times m}$. Moreover, the challenge tag $t^*$ is mapped to an $L$-bit string $\tau^* = \text{AHF}(t^*)$ such that

$$A(\tau^*) = A \cdot R_{\tau^*} + W^T = A \cdot R_{\tau^*} + V^T \cdot G_0 \label{eq:8}
$$

21
Game$_3$: In this game, we modify the distribution of mpk and replace the uniform matrix $A \in \mathbb{Z}_q^{n \times m}$ by a lossy matrix such that

$$A^\top = \hat{A}^\top \cdot C + E \in \mathbb{Z}_q^{m \times n},$$  \hspace{1cm} (9)

where $\hat{A} \leftarrow U(\mathbb{Z}_q^{n \times m})$, $C \leftarrow U(\mathbb{Z}_q^{1 \times n})$ and $E \leftarrow D_{\mathbb{Z}_q^{m \times n}, \alpha q}$, for $n_1 \ll n$. The matrix (9) is thus “close” to a matrix $\hat{A}^\top \cdot C$ of much lower rank than $n$.

Under the LWE assumption in dimension $n_1$ with error rate $\alpha_1$, this change should not significantly affect $A$’s behavior and a straightforward reduction shows that $|\Pr[W_3] - \Pr[W_2]| \leq n \cdot \text{Adv}^\text{LWE}_{n_1, \alpha_1}(\lambda)$, where the factor $n$ comes from the use of an LWE assumption with $n$ secrets.

Game$_4$: In this game, we modify the encryption oracle. At each encryption query $(i, x_i^{(j)}, t^{(j)})$, the challenger generates the ciphertext by computing:

$$C_{t,i} = G_0^\top \cdot x_i^{(j)} + R_{t^{(j)}}^\top \cdot \hat{A}^\top \cdot C \cdot s_i + e_i^{(j)} \in \mathbb{Z}_q^m,$$

and for each challenge query $(i, x_{0,i}^*, x_{1,i}^*, t^*)$ the challenger replies with:

$$C_{t^*,i} = G_0^\top \cdot x_{b,i}^* + (R_{t^*}^\top \cdot \hat{A}^\top \cdot C + G_0^\top \cdot V) \cdot s_i + e_i^* \in \mathbb{Z}_q^m$$  \hspace{1cm} (11)

where $e_i^{(j)} \leftarrow D_{\mathbb{Z}_q^{m \times q}}$ and $e_i^* \leftarrow D_{\mathbb{Z}_q^{m \times q}}$. The only difference between Game$_3$ and Game$_4$ is thus that the terms $R_{t^{(j)}}^\top \cdot E \cdot s_i + e_i^{(j)}$ and $R_{t^*}^\top \cdot E \cdot s_i + e_i^*$ are replaced by $e_i^{(j)}$ and $e_i^*$ respectively, at each encryption or challenge query. However, the smudging lemma (Lemma 2.4) ensures that the two distributions are statistically close as long as $\alpha$ is sufficiently large with respect to $\alpha_1$ and $\sigma$. Concretely, Lemma 3.3 implies $|\Pr[W_4] - \Pr[W_3]| \leq \ell \cdot (Q + 1) \cdot 2^{-\Omega(\lambda)}$.

Game$_5$: This game is like Game$_4$ but we modify the challenge oracle. Instead of encrypting $X_b^* = [x_{b,1}^* | \ldots | x_{b,\ell}^*]$ as in (11), the challenger encrypts a linear combination of $X_b^*$ and $X_0^*$. It initially chooses a uniformly random $\gamma \leftarrow U(\mathbb{Z}_q)$ and, at each challenge query $(i, x_{0,i}^*, x_{1,i}^*, t^*)$, computes $C_{t^*,i}$ as

$$C_{t^*,i} = G_0^\top \cdot ((1 - \gamma) \cdot x_{b,i}^* + \gamma \cdot x_{0,i}^*) + (R_{t^*}^\top \cdot \hat{A}^\top \cdot C + G_0^\top \cdot V) \cdot s_i + e_i^*,$$

with $e_i^* \leftarrow D_{\mathbb{Z}_q^{m \times q}}$, for all $i \in [\ell]$. Lemma 3.4 shows that Game$_4$ and Game$_5$ are negligibly far apart as $|\Pr[W_5] - \Pr[W_4]| \leq 2^{-\Omega(\lambda)}$.

In Game$_5$, we clearly have $\Pr[W_5] = 1/2$ since the challenge ciphertexts $(C_{t,1}, \ldots, C_{t,\ell})$ reveal no information about $b \in \{0,1\}$.  \hspace{1cm} □

Lemma 3.3. Let $R_\tau \in \mathbb{Z}_q^{m \times m}$ be as in equation (6). Let $E \leftarrow D_{\mathbb{Z}_q^{m \times n}, \alpha q}$ and $s \leftarrow D_{\mathbb{Z}_q^{m \times \tau}, \sigma}$. If $\alpha q = \omega(\sqrt{\log n})$, $\sigma = \omega(\sqrt{\log n})$ and $\alpha \geq 2^L \cdot m^2 \cdot n^{-3/2} \cdot \alpha_1 \cdot \sigma$, we have the statistical distance upper bound $\Delta(D_{\mathbb{Z}_q^{n \times q}}, R_\tau^\top \cdot E \cdot s + D_{\mathbb{Z}_q^{n \times q}}) \leq 2^{-\lambda}$.

(The proof is given in Appendix C.2.)
Lemma 3.4. We have $|\Pr[W_5] - \Pr[W_4]| \leq 2^{-\Omega(\lambda)}$.

Proof. To prove the result, we resort to a technique of guessing in advance the difference $X_{i-b} - X_b^*$, which was previously used in [69,14] and can be seen as complexity leveraging with respect to a statistical argument. We consider the following variants of $\text{Game}_1$ and $\text{Game}_5$, respectively.

We define $\text{Game}'_4$ and $\text{Game}_5'$ simultaneously by using an index $k \in \{4, 5\}$:

$\text{Game}'_4$: This game is like $\text{Game}_k$ with one difference in the setup phase. To generate $mpk$, the challenger $B$ generates a statistically uniform $U \in \mathbb{Z}^{(n_0+n_1) \times n}_q$ with a trapdoor $T_U$ for the lattice $A^\perp(U)$. Then, $B$ parses $U$ as

$$U = \begin{bmatrix} V \\ C \end{bmatrix} \in \mathbb{Z}^{(n_0+n_1) \times n}_q,$$

where $V \in \mathbb{Z}^{n_0 \times n}_q$ and $C \in \mathbb{Z}^{n_1 \times n}_q$ are statistically independent and uniform over $\mathbb{Z}_q$. Next, it computes

$$A^T = \hat{A}^T \cdot C + E \in \mathbb{Z}_q^{m \times n},$$

where $\hat{A} \leftarrow U(\mathbb{Z}_q^{n_1 \times m})$ and $E \leftarrow D_{\mathbb{Z}_q^{n \times n}, \alpha, q}$. The obtained matrix $A \in \mathbb{Z}_q^{n \times m}$ is then used to generate $\{A_{i,j}\}_{i \in [L], j \in [0,1]}$ as per (20). The upper part $V \in \mathbb{Z}_q^{n_0 \times n}$ of $U$ is included in $mpk$, the distribution of which is statistically close to that of $\text{Game}_k$: we indeed have $|\Pr[W_k^*] - \Pr[W_k]| \leq 2^{-\Omega(\lambda)}$.

We do the same as above and define $\text{Game}'_4''$ and $\text{Game}_5''$ simultaneously by using an index $k \in \{4, 5\}$:

$\text{Game}_5'$: This game is identical to $\text{Game}'_4$ with the following difference. At the outset of the game, the challenger randomly chooses $\Delta X \leftarrow U([-2X, 2X]^{n_0 \times \ell})$ as a guess for the difference $X_{i-b} - X_b^*$ between the challenge messages $X_0^*, X_1^*$. In the challenge phase, the challenger checks if $\Delta X = X_{i-b} - X_b^*$. If not, it aborts and replaces $A'$’s output $b$ with a random bit $b'' \leftarrow U(\{0,1\})$.

If the guess for $X_{i-b} - X_b^*$ was successful (we call $\text{Guess}$ this event), the challenger proceeds exactly as it did in $\text{Game}_k'$. Since the choice of $\Delta X \leftarrow U([-2X, 2X]^{n_0 \times \ell})$ is completely independent of $A$’s view, we clearly have $\Pr[\text{Guess}] = 1/(4X)^{n_0 \ell}$. Since $\text{Game}_5''$ is identical to $\text{Game}'_4$ when $\text{Guess}$ occurs, this implies $\text{Adv}_4''(A) = (4X)^{n_0 \ell} \cdot \text{Adv}_4'(A)$. Indeed,

$$\text{Adv}_4''(A) = \frac{1}{2} \cdot \Pr[b' = 1 \mid b = 1, \text{Guess}] \cdot \Pr[\text{Guess}] + \frac{1}{2} \cdot \Pr[\text{Guess}] = \frac{1}{2} \cdot \Pr[\text{Guess}] \cdot | \Pr[b' = 1 \mid b = 1, \text{Guess}] - \Pr[b' = 1 \mid b = 0, \text{Guess}] |$$

$$= \Pr[\text{Guess}] \cdot \text{Adv}_4'(A) = \frac{1}{(4X)^{n_0 \ell}} \cdot \text{Adv}_4'(A)$$

and we can similarly show that $\text{Adv}_5''(A) = (4X)^{n_0 \ell} \cdot \text{Adv}_5'(A)$. 

23
We first claim that, conditionally on \( \cdot \), which in turn implies \( \cdot \). This implies that \( \cdot \), which is exactly the distribution from Game 3.5.

Indeed, our choice of \( \cdot \) assuming that \( \cdot \), is replaced by alternative encryption keys \( \{ek_i\}_{i=1}^\ell \), which are generated as follows. After having sampled \( \cdot \) for all \( i \in [\ell] \), the challenger \( B \) chooses \( \gamma \leftarrow U(\mathbb{Z}_q) \) and uses the trapdoor \( T_U \) for \( A^\perp(U) \) to sample a small-norm matrix \( T \in \mathbb{Z}^{n \times n_0} \) satisfying

\[
U \cdot T = \left[ \begin{array}{c} \gamma \cdot 1_{n_0} \\ 0^{n_1 \times n_0} \end{array} \right] \mod q, \tag{12}
\]

so that \( V \cdot T = \gamma \cdot 1_{n_0} \mod q \) and \( C \cdot T = 0^{n_1 \times n_0} \mod q \). For each \( i \in [\ell] \), \( B \) then defines the alternative key \( ek'_i = s'_i \) of user \( i \) to be

\[
s'_i = s_i + T \cdot \Delta x_i \in \mathbb{Z}^n \quad \forall i \in [\ell], \tag{13}
\]

where \( \Delta x_i \) is the \( i \)-th column of \( \Delta X \) (i.e., the guess for \( x_{i-b,i} - x_{i}^* \)). These modified encryption keys \( \{ek'_i = s'_i\}_{i=1}^\ell \) are used to answer all encryption queries and to generate the challenge ciphertext. At each corruption query \( i \), the adversary is also given \( ek'_i \) instead of \( ek_i \).

We finally claim that, from the adversary's view \( \cdot \) is identical to \( Game''_4 \). To see this, we first argue that trading \( \{ek_i\}_{i=1}^\ell \) for \( \{ek'_i\}_{i=1}^\ell \) has no incidence on queries made by a legitimate adversary:

- We have \( C \cdot s'_i = C \cdot s_i \mod q \), so that encryption queries obtain the same responses no matter which key set is used among \( \{ek_i\}_{i=1}^\ell \) and \( \{ek'_i\}_{i=1}^\ell \).
- We have \( \sum_{i=1}^\ell s'_i \cdot y_i = \sum_{i=1}^\ell s_i \cdot y_i \) so long as the adversary only obtains private keys for vectors \( y \in \mathbb{Z}^\ell \) such that \( (X_0^* - X^\perp) \cdot y = 0 \) (over \( \mathbb{Z} \)).
- For any corrupted user \( i \in CS \), it should be the case that \( x_{0,i}^* = x_{1,i}^* \), meaning that \( s'_i = s_i \) as long as \( \cdot \) occurs.

This implies that \( Game''_5 \) is identical to \( Game''_4 \), except that users' secret keys are defined via (13) and thus have a slightly different distribution. Lemma 3.5 shows that the statistical distance between the distributions of \( \{s'_i\}_{i=1}^\ell \) and \( \{s_i\}_{i=1}^\ell \) is at most \( 2^{-\lambda} \cdot (4X)^{-n_0 \ell} \). This implies that \( Game'_4 \) and \( Game''_4 \) are statistically close assuming that \( \cdot \) occurs. When \( \cdot \) does not occur, both games output a random \( b' \leftarrow U((0, 1)) \), so that \( |Pr[W'_0] - Pr[W''_0]| \leq 2^{-\lambda} \cdot (4X)^{-n_0 \ell} \).

We finally claim that, from the adversary's view \( Game''_5 \) is identical to \( Game''_4 \). Indeed, our choice of \( T \) ensures that \( V \cdot T = \gamma \cdot 1_{n_0} \mod q \), so that we have

\[
V \cdot s'_i = V \cdot s_i + \gamma \cdot (x_{1-b,i}^* - x_{b,i}^*) \mod q.
\]

This implies

\[
C_{t,i} = G_{0}^\top \cdot x_{b,i}^* + (R_{0}^\top \cdot \hat{A}^\top \cdot C + G_{0}^\top \cdot V) \cdot s'_i + e_i^* = G_{0}^\top \cdot ((1 - \gamma) \cdot x_{b,i}^* + \gamma \cdot x_{1-b,i}^*) + (R_{0}^\top \cdot \hat{A}^\top \cdot C + G_{0}^\top \cdot V) \cdot s_i + e_i^*
\]

which is exactly the distribution from \( Game''_4 \).

Putting the above altogether, we find

\[
|Pr[W'_0] - Pr[W''_0]| \leq 2^{-\Omega(\lambda)} \cdot (4X)^{-n_0 \ell},
\]

which in turn implies \( |Pr[W_4] - Pr[W_5]| \leq 2^{-\Omega(\lambda)} \), as claimed.

\[
\square
\]

**Lemma 3.5.** If \( \sigma \geq 2^\lambda \cdot n_0 \cdot (4X)^{n_0 \ell + 1} \cdot \omega(n^2 \sqrt{\log n}) \), then we have the inequality

\[
\Delta(\cdot) \leq 2^{-\lambda} \cdot (4X)^{-n_0 \ell}. \tag{14}
\]

(The proof is in Appendix C.3.)
4 A DMCFE Scheme for Linear Functions

As in [28], our DMCFE scheme combines two instances of the underlying centralized scheme of Section 3. While the second instance is used exactly in the same way as in the centralized construction, the first instance is used for the sole purpose of generating partial functional secret keys without having the senders communicate with one another. As in [28], the senders have to initially run an interactive protocol in order to jointly generate public parameters for the two schemes. Note that this protocol is the only step that requires interaction among senders and it is only executed once. This interactive step ends with each sender holding an encryption key \( \mathbf{ek}_i = (\mathbf{s}_i, \mathbf{t}_i) \) comprised of encryption keys for the two MCPE instances. The distributed protocol also ensures that a functional secret key \( \mathbf{t} = \sum_{i=1}^{\ell_i} \mathbf{t}_i \) for the all-one vector \((1, 1, \ldots, 1)^\top \in \mathbb{Z}^\ell\) be made publicly available for the first MCPE instance. Later on, when a decryptor wishes to obtain a partial functional secret key \( \mathbf{dkf}_i \) for a vector \( \mathbf{y} = (y_1, \ldots, y_\ell)^\top \) from the \( i \)-th sender \( \mathcal{S}_i \), the latter can generate an MCPE encryption of the vector \( y_i \cdot \mathbf{s}_i \in \mathbb{Z}^n \) under his secret key \( \mathbf{t}_i \). Having obtained partial functional secret keys \( \mathbf{dkf}_i \) from all senders \( \{\mathcal{S}_i\}_{i=1}^{\ell} \), the decryptor can then use the functional secret key \( \mathbf{t} = \sum_{i=1}^{\ell} \mathbf{t}_i \) to compute \( \mathbf{s}_y = \sum_{i=1}^{\ell} y_i \cdot \mathbf{s}_i \in \mathbb{Z}^n \).

4.1 Description

We assume global public parameters

\[
\mathbf{cp} := \left( \lambda, \ell_{\text{max}}, X, \bar{X}, Y, \bar{Y}, n_0, n_1, n, \bar{n}, m, \bar{m}, \alpha, \alpha_1, \bar{\alpha}_1, \bar{\alpha}, \ell_f, \ell, L, q, \bar{q}, \text{AHF}_L, \text{AHF}_{\ell_f} \right),
\]

which specify a security parameter \( \lambda \) and the following quantities

- Let \( \ell_{\text{max}} = \lambda^k, n_1 = \lambda^d, \bar{d} = 3d + k - 1, q = 2^{\lambda^d+1} + \lambda, \bar{q} = 2^{\lambda^d+1} + \lambda, \bar{n}_1 = \lambda^d, \alpha_1 = 2^{-\lambda^d+1+d \log \lambda}, \bar{\alpha}_1 = 2^{-\lambda^d+1+d \log \lambda}, \alpha = 2^{-\lambda^d+1}, n_0 \cdot \ell_{\text{max}} = O(\lambda^{d-2}), n_0 = O(\lambda^d), n = O(\lambda^{d+1}), \bar{n} = O(\lambda^{d+1+k-2}), X = 1, \bar{Y} = 1, \sigma = 2^{\lambda^d-1-2\lambda}, \bar{\sigma} = 2^{\lambda^d-1-2\lambda}, \bar{X} = 2\ell \cdot Y \cdot \sigma \sqrt{n}, \text{ and the rest of the parameters } Y, m, \bar{m} \text{ are all in } \text{poly}(\lambda).

- A tag length \( \ell_t \in \Theta(\lambda) \) and a length \( \ell_f \in \Theta(\lambda) \) of function labels.

- Dimensions \( n, m, n_0, n_1, n, \bar{n}, m, \bar{m} \in \text{poly}(\lambda) \) such that \( n > 3 \cdot (n_0 + n_1) \cdot \lceil \log q \rceil, m > 2 \cdot n \cdot \lceil \log q \rceil, \bar{n} > 3 \cdot (n + \bar{n}_1) \cdot \lceil \log \bar{q} \rceil \) and \( \bar{m} > 2 \cdot n \cdot \lceil \log \bar{q} \rceil \).

- The description of balanced admissible hash functions \( \text{AHF}_L : \{0, 1\}^{\ell_t} \to \{0, 1\}^L \) and \( \text{AHF}_{\ell_f} : \{0, 1\}^{\ell_f} \to \{0, 1\}^L \), for a suitable \( L \in \Theta(\lambda) \).

- A real \( \alpha > 0 \) and a Gaussian parameter \( \sigma > 0 \), which will specify an interval \([\bar{\beta}, \beta] = [-\sigma \sqrt{n}, \sigma \sqrt{n}] \) where the coordinates of the secret will live (with probability exponentially close to 1).

We define \( \mathbf{G} \in \mathbb{Z}_q^{n \times \bar{m}} \) to be the gadget matrix

\[
\mathbf{G} = [\mathbf{I}_n \otimes (1, 2, 4, \ldots, 2\lceil \log \bar{q} \rceil)] \cdot 0^n \mid \ldots \mid 0^n \in \mathbb{Z}_q^{n \times \bar{m}}
\]
where $I_n \otimes (1, 2, 4, \ldots, 2^{[\log \bar{q}]})$ is padded with $m-n \cdot [\log \bar{q}]$ zero columns.

**Setup**($cp, 1^\ell$): On input of a number of users $\ell < \ell_{\text{max}}$, the senders $\{S_i\}_{i=1}^\ell$ run an interactive protocol at the end of which the following quantities are made publicly available.

- Random matrices $A_{i,b} \leftarrow U(\mathbb{Z}_q^{m \times m})$, for each $i \in [L], b \in \{0,1\}$.
- Random matrices $B_{i,b} \leftarrow U(\mathbb{Z}_q^{n \times \bar{m}})$, for each $i \in [L], b \in \{0,1\}$.
- Random matrices $V \leftarrow U(\mathbb{Z}_q^{n \times \bar{n}})$.
- The sum $t = \sum_{i=1}^\ell t_i \in \mathbb{Z}^n$ of Gaussian vectors $t_i \leftarrow D_{\mathbb{Z}^n, \sigma}$ for $i \in [\ell]$. In addition, for each $i \in [\ell]$, the $i$-th sender $S_i$ privately obtains the following:

  - The $i$-th term $t_i \in \mathbb{Z}^n$ of the sum $t = \sum_{i=1}^\ell t_i$.
  - A Gaussian vector $s_i \leftarrow D_{\mathbb{Z}^n, \sigma}$, which is used to define $S_i$’s encryption key $ek_i = (s_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}^n$ and the corresponding secret key $sk_i = (s_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}^n$.

The master public key is defined to be

$$mpk := (cp, V, \{A_{i,0}, A_{i,1} \in \mathbb{Z}_q^{m \times m}\}_{i=1}^L, \{B_{i,0}, B_{i,1} \in \mathbb{Z}_q^{\bar{n} \times \bar{m}}\}_{i=1}^L, t),$$

while $S_i$ obtains $ek_i = (s_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}^n$ and $sk_i = (s_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}^n$ for each $i \in [\ell]$.

**DKeygenShare**($sk_i, t_f$): Given the secret key $sk_i = (s_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}^n$ and the label $t_f$ of a linear function $f' : \mathbb{Z}_q^{n_0} \times \ell \rightarrow \mathbb{Z}_q^{n_0}$ described by a vector $y = (y_1, \ldots, y_\ell) \in [-Y, Y]^\ell$, conduct the following steps.

1. Compute $\tau_f = \tau_f[1] \ldots \tau_f[L] = \text{AHF}_f(t_f) \in \{0,1\}^L$ as well as

   $$B(\tau_f) = B_{\ell_f, \tau_f[L]} \bar{G}^{-1} \left( B_{\ell_f-1, \tau_f[L-1]} \bar{G}^{-1} \left( \ldots B_{2, \tau_f[2]} \bar{G}^{-1}(B_{1, \tau_f[1]}) \right) \right) \bar{G}^{-1}(W^\top) \in \mathbb{Z}_q^{\bar{n} \times \bar{m}},$$

   where $\bar{W} = \bar{G} \cdot \bar{V} \in \mathbb{Z}_q^{\bar{n} \times \bar{n}}$.

2. Sample a noise vector $e_{f,i} \leftarrow D_{\mathbb{Z}_{\bar{q}}, \alpha_{\bar{q}}}$. Then, compute

   $$dk_{f,i} = \bar{G}^\top \cdot (y_i \cdot s_i) + B(\tau_f)^\top \cdot t_i + e_{f,i} \in \mathbb{Z}_{\bar{q}}^\bar{m}.$$  \hspace{1cm} (15)

Output the partial functional decryption key $dk_{f,i} \in \mathbb{Z}_{\bar{q}}^\bar{m}$.

**DKeygenComb**($\{dk_{f,i}\}_{i=1}^\ell, t_f$): Given the label of a function described by a vector $y = (y_1, \ldots, y_\ell) \in [-Y, Y]^\ell$ and $\ell$ partial functional keys $\{dk_{f,i}\}_{i=1}^\ell$ where $dk_{f,i} \in \mathbb{Z}_{\bar{q}}^\bar{m}$ for each $i \in [\ell]$, conduct the following steps.

1. Compute $\tau_f = \text{AHF}_f(t_f) \in \{0,1\}^L$ and parse it as $\tau_f = \tau_f[1] \ldots \tau_f[L].$

2. Compute $B(\tau_f) \in \mathbb{Z}_q^{\bar{n} \times \bar{m}}$ as per (14).

3. Compute $d_{f,i} = \sum_{i=1}^\ell dk_{f,i} - B(\tau_f)^\top \cdot t \mod \bar{q}$, where $t \in \mathbb{Z}^n$ is taken from $mpk$.  

26
4. Interpret $d_i \in \mathbb{Z}_q^m$ as a vector of the form $d_i = G^T \cdot s_i + \tilde{e}_f \mod q$, for some error vector $\tilde{e}_f \in [-B, B]^m$. Using the public trapdoor of $\Lambda^2(G)$, compute the underlying $s_i \in [-\ell \cdot \beta \cdot Y, \ell \cdot \beta \cdot Y]^n$.

Output the functional secret key $dk_y = (y, s_y)$.

$\text{Encrypt}(ek_i, x_i, t)$: Given $ek_i = s_i \in \mathbb{Z}^n$, $x_i \in [-X, X]^n_0$, and $t \in \{0, 1\}^{\ell_t}$,

1. Compute $\tau = \text{AHF}(t) \in \{0, 1\}^L$ and parse it as $\tau = \tau[1] \ldots \tau[L]$.
2. Letting $W = G_0^T \cdot V \in \mathbb{Z}_q^{m \times n}$, compute

$$A(\tau) = A_{L, \tau[L]} \cdot G^{-1} \left( A_{L-1, \tau[L-1]} \cdot G^{-1} \left( \cdots A_{2, \tau[2]} \cdot G^{-1} \left( A_{1, \tau[1]} \right) \right) \right) \cdot G^{-1}(W^T) \in \mathbb{Z}_q^{n \times m}. \quad (16)$$

3. Sample a noise vector $e_i \leftarrow D_{\mathbb{Z}_q^{\ell m}}$. Then, compute and output

$$C_{t,i} = G_0^T \cdot x_i + A(\tau)^T \cdot s_i + e_i \in \mathbb{Z}_q^m.$$

$\text{Decrypt}(dk_y, t, C_i)$: On input of a functional secret key $dk_y = (y, s_y)$ for a vector $y = (y_1, \ldots, y_l)^T \in [-Y, Y]^l$, a tag $t \in \{0, 1\}^{\ell_t}$, and an $L$-vector of ciphertexts $C_i = (C_{t,1}, \ldots, C_{t,L}) \in (\mathbb{Z}_q^m)^L$, conduct the following steps.

1. Compute $\tau = \text{AHF}(t) \in \{0, 1\}^L$ and parse it as $\tau = \tau[1] \ldots \tau[L]$.
2. Compute $A(\tau) \in \mathbb{Z}_q^{n \times m}$ as per (16).
3. Compute $f_{i,y} = \sum_{i=1}^L y_i \cdot C_{t,i} - A(\tau)^T \cdot s_y \mod q$.
4. Interpret $f_{i,y} \in \mathbb{Z}_q^m$ as a vector of the form $f_{i,y} = G_0^T \cdot z + \tilde{e} \mod q$, for some error vector $\tilde{e} \in [-B, B]^m$. Using the public trapdoor of $\Lambda^2(G_0)$, compute and output the underlying vector $z \in [-\ell \cdot X \cdot Y, \ell \cdot X \cdot Y]^n_0$.

The scheme’s correctness is implied by that of the two underlying centralized schemes. In turn, these are correct by Lemma 3.1 and the choice of parameters.

4.2 Security

The proof of Theorem 4.1 is given in Appendix D. It proceeds with a sequence of games where the first (resp. last) game is the real experiment of Definition 2.14 where the challenger’s bit is $b = 0$ (resp. $b = 1$).

In order to reduce the security of the centralized scheme to that of its decentralized variant, the proof first moves to a game where the partial functional key generation oracle of Definition 2.14 can be simulated using the functional key generation oracle of Definition 2.11. To this end, it relies on the security of the first MCFE instance. The reduction has the particularity that it has to send dummy encryption queries to its challenger in order to comply with Condition 2 of the “finalize” step in Definition 2.11. The next step is to move to a game where encryption queries $(i, x_{i,0}, x_{i,1}, t)$ are answered by returning encryptions of $x_{i,1}$ instead of $x_{i,0}$. For this purpose, we rely on the security of the second MCFE instance, which is possible since the partial key generation oracle can be simulated using the centralized key generation oracle. In order to make sure that
the simulation does not break Condition 3 in the “finalize” step of Definition 2.11, the reduction only invokes its centralized key generation oracle in the very last partial key generation query involving an uncorrupted user. We note that a similar technique was used in [28], where keys were computed modulo a prime. Here, our reduction has to apply the same strategy over the integers. However, the technique still works as long as the message space of the first MCFE instance is large enough to contain functional secret keys $\sum_{i=1} y_i \cdot s_i$ in each coordinate. The final transition restores the partial key generation oracle of Definition 2.14 to its original output distribution. To this end, we invoke again the security of the first MCFE instance and reverse the transition of the first step.

**Theorem 4.1.** The above DMCFE scheme provides sta-IND-sec security under the LWE assumption.

**Acknowledgements**

Part of this work was funded by the French ANR ALAMBIC project (ANR-16-CE39-0006) and by BPI-France in the context of the national project RISQ (P141580). This work was also supported by the European Union PROMETHEUS project (Horizon 2020 Research and Innovation Program, grant 780701).

**References**

A One-or-less MCFE Compiler

In the definition of 1-challenge IND-sec security, (i.e., Definition 2.12), the adversary has to make the challenge encryption queries $\text{CQEncrypt}(j, x_j^{b_j}, x_{j^*}^{b_{j^*}}, t^*)$ for all uncorrupted slots $j \in HS$ as long as a query of the form $\text{CQEncrypt}(i, x_i^{b_i}, x_{i^*}^{b_{i^*}}, t^*)$ is made for some slot $i \in HS$. Hence, it is not allowed to use the partial information that could be revealed by incomplete challenge queries.

It is desirable to strengthen the security definition and prove security even when the adversary is allowed to obtain partial ciphertexts by making challenge queries $\{\text{CQEncrypt}(i, x_i^{b_i}, x_{i^*}^{b_{i^*}}, t^*)\}_{i \in I}$ for a proper subset $I \subset HS$. In this section, we recall the definition of this security requirement and provide a compiler that takes a secure MCFE satisfying 1-challenge IND-sec security and upgrades it so as to achieve security in the sense of the stronger definition. The transformation relies on pseudorandom generators and adaptively secure multi-instance PRFs. A definition of the latter notion is given in this subsection.

**Definition A.1 (1-or-less-challenge IND-sec).** For an MCFE scheme with $\ell$ senders, we consider the following game between an adversary $A$ and a challenger $C$. The game involves a set $HS$ (initialized to $HS := [\ell]$), of honest senders and a set $CS$ (initialized to $CS := \emptyset$), of corrupted senders.

**Initialization:** The challenger $C$ generates $cp$ and runs $(\text{mpk}, \text{msk}, \{ek_i\}_{i=1}^{\ell}) \leftarrow \text{Setup}(cp, 1^\ell)$. Then, it chooses a random bit $b \leftarrow \{0, 1\}$ and gives the master public key $\text{mpk}$ to the adversary $A$.

**Encryption queries:** The adversary can adaptively make encryption queries $\text{Encrypt}(i, x, t)$, to which the challenger replies with $\text{Encrypt}(ek_i, x, t)$. Any further query involving the same pair $(i, t)$ is ignored.

**Challenge queries:** The adversary adaptively makes challenge queries of the form $\text{CQEncrypt}(i, x_i^{b_i}, x_{i^*}^{b_{i^*}}, t^*)$. The challenger replies with $\text{Encrypt}(ek_i, x_i^{b_i}, t^*)$. 

31
Only one tag \( t^* \) can be involved in a challenge query. If \( t^* \) denotes the tag of the first query, the challenger only answers subsequent challenge queries for the same \( t^* \). Moreover, only one query \((i, t^*)\) is allowed for each \( i \in [\ell] \) and subsequent queries involving the same \( i \in [\ell] \) are ignored.

**Functional decryption key queries:** The adversary can adaptively obtain functional decryption keys via queries \( \text{QDKeygen}(f) \). At each query, the challenger returns \( \text{dk}_f \leftarrow \text{DKeygen}(\text{msk}, f) \).

**Corruption queries:** For any user \( i \in \mathcal{HS} \), the adversary can adaptively make queries \( \text{QCorrupt}(i) \), to which the challenger replies with \( \text{ek}_i \) and updates \( \mathcal{HS} \) and \( \mathcal{CS} \) by setting \( \mathcal{CS} := \mathcal{CS} \cup \{ i \} \) and \( \mathcal{HS} := \mathcal{HS} \setminus \{ i \} \).

**Finalize:** The adversary outputs a bit \( b' \in \{0, 1\} \). The adversary \( A \) wins if \( b = b' \), unless one of the situations below occurred.

1. A challenge query \( \text{CQEncrypt}(i, x_i^0, x_i^1, t^*) \) has been made for an index \( i \in \mathcal{CS} \) with \( x_i^0 \neq x_i^1 \).
2. An encryption query \( \text{QEncrypt}(i, x, t^*) \) has been made for the challenge tag \( t^* \) and an index \( i \in [\ell] \).
3. For the challenge tag \( t^* \) and some function \( f \) queried to \( \text{QDKeygen} \), there exists a pair of vectors \((x_i^0, x_i^1)\) such that \( f(x_i^0) \neq f(x_i^1) \), where
   - \( x_i^0 = x_i^1 \) for all \( i \in \mathcal{CS} \);
   - \( \text{CQEncrypt}(i, x_i^0, x_i^1, t^*) \) have been asked for all \( i \in \mathcal{HS} \).

If any of these events occurred, \( A \)'s output is overwritten by \( b \leftarrow U(\{0, 1\}) \).

We say that an MCFE scheme provides \( \leq 1\text{Ch-IND} \) security if, for any efficient adversary \( A \), we have \( \text{Adv}^{1\text{Ch-IND}}(A) := |\text{Pr}[b = b'] - \frac{1}{2}| \in \text{neg}(\lambda) \).

The definition of 1-or-less IND security is identical to the 1Ch-IND definition, except that we removed Condition 3 of the Finalize step in Definition 2.12. In Definition A.1, we insist that the last condition of the Finalize step does not impose any restriction on the functions queried to \( \text{QDKeygen} \) if there exists a single index \( i \in \mathcal{HS} \) for which no challenge query \( \text{CQEncrypt}(i, \cdot, \cdot, t^*) \) was made.

In order to achieve security in the sense of Definition A.1, our compiler uses as a building block a pseudorandom function family that satisfies a definition of *adaptive multi-instance PRF*. This notion is defined as follows.

**Definition A.2 (ad-mi-PRF).** An efficiently computable function \( F : K \times X \rightarrow Y \) is an adaptively secure \( N \)-instance PRF, for some \( N \in \text{poly}(\lambda) \), if no PPT adversary \( A \) has non-negligible advantage in winning the following game.

**Initialization:** The challenger \( \mathcal{C} \) samples \( N \) secret keys \( k_1, k_2, \ldots, k_N \leftarrow K \) and a uniformly random bit \( d \leftarrow U(\{0, 1\}) \).

**Queries.** The adversary \( A \) adaptively interleaves the following kinds queries:

**Evaluation:** Upon receiving a query \( \text{Eval}(i, X) \), where \( i \in [N] \) and \( X \in X \), the challenger returns \( \perp \) if it previously replied to a challenge query for the same pair \((i, X)\). Otherwise, it replies with \( F_{k_i}(X) \in Y \).

**Corruption:** When the adversary makes a query \( \text{Corrupt}(i) \), the challenger returns \( \perp \) if it previously replied to a challenge query for \( i \). Otherwise, the challenger returns \( k_i \in K \).
The adversary’s advantage is $\text{Adv}_{\mathcal{A}}^{\text{ad-mi-PRF}}(\lambda) := \left| \text{Pr}[d' = d] - \frac{1}{2} \right|$. It is possible to show that any PRF that provides single-instance security in the sense of Definition A.2 without a $2$-loss is only linear in the number of slots $\ell$, which would eventually become $\ell^3$ because of the bound (17). In Appendix A.2, we show that a PRF construction described in [56] can be proven secure in the sense of Definition A.2 without a $O(N^2)$ loss in the reduction, so that our security loss is only linear in the number of slots $\ell$. In comparison, the compiler of [1] loses a factor $O(\ell^2)$ but relies on any PRF.

A.1 The Transformation

The compiler is similar to the random-oracle-based transformation of Abdalla et al. [2, Section 4.2] – which is itself inspired by [4] – and its intuition is the following. The ciphertexts of individual slots contain partial MCFE ciphertexts $\text{Encrypt}'(ek'_i, x_i, t)$ which are super-encrypted using a symmetric encryption scheme with secret key $K_{t,i}$. The partial ciphertexts of the compiled scheme also contain shares $k_{ijt}$ of the secret key $K_{i,j}$ in such a way that, once all the partial ciphertexts are gathered for a given tag $t$, the decryptor can recover the secret $K_{t,i} = \bigoplus_{j=1}^\ell k_{ijt}$ and thus the encryption $\text{Encrypt}'(ek'_i, x_i, t)$. If only one such ciphertext is missing, then the secret key $K_{t,i}$ remains computationally hidden because the decryptor does not have all the shares. The symmetric encryption layer thus hides all the information that could leak when the adversary does not make encryption queries for all uncorrupted slots.

Given an MCFE scheme $\text{MCFE}' = (\text{Setup}', \text{Encrypt}', \text{DKeygen}', \text{Decrypt}')$, an adaptively secure $\ell^2$-instances $\text{ad-mi-PRF}$ $F : \mathcal{K} \times \mathcal{T} \rightarrow \{0,1\}^\lambda$ and a PRG $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{[\ell]}$, where $[\ell]$ is the length of the output of the algorithm $\text{Encrypt}'$, we obtain the following compiled scheme $\text{MCFE}$.

**Setup**(cp, $1^\lambda$) : Runs $(\text{mpk}', \text{msk}', \{ek'_i\}_{i=1}^\ell) \leftarrow \text{Setup}'(\text{cp}, 1^\lambda)$. Samples $\ell^2$ secret keys $k_{ij} \leftarrow \mathcal{K}$ for all $i,j \in [\ell]$. Then, set $\text{mpk} := \text{mpk}'$ $\text{msk} := \text{msk}'$ and $ek_i := (ek'_i, \{k_{ij}, k_{ij}'\}_{j=1}^\ell)$

**Encrypt**(ek$_i$, x$_i$, t) : Given $ek_i = (ek'_i, \{k_{ij}, k_{ij}'\}_{j=1}^\ell)$ it computes $k_{ijt} := F_{k_{ij}}(t)$, $k_{ijt} := F_{k_{ij}}(t)$ for all $j \in [\ell]$, $C'_{t,i} := \text{Encrypt}'(ek'_i, x_i, t)$ and $K_{t,i} := \bigoplus_{j=1}^\ell k_{ijt}$. Next, it computes $C_{t,i} := C'_{t,i} \oplus G(K_{t,i})$. It outputs $ct_{t,i} = (C_{t,i}, \{k_{ijt}\}_{j=1}^\ell)$. 

33
**DKeygen**(*msk, f*): Given *msk* and an ℓ-argument function *f* : M^ℓ → R. It outputs a functional decryption key *dk_f := DKeygen*(msk', *f*).

**Decrypt**(dk_f, t, {ct_{t,1}, ct_{t,2}, ... , ct_{t,ℓ}}): Takes as input a functional decryption key dk_f, a tag t ∈ T, and an ℓ-vector of ciphertexts C = (ct_{t,1}, ... , ct_{t,ℓ}). For all *i* ∈ [ℓ] it computes K_{i,t} := ∑_{j=1}^{ℓ} k_{ij,t} and C_{i,t} := C_{t,i} ⊕ G(K_{i,t}) and runs Decrypt*(dk_f, t, {C'_{t,1}, C'_{t,2}, ... , C'_{t,ℓ}}).

The security proof relies on the idea that, if the adversary *A* does not make all the allowed challenge encryptions queries, there must exist an honest slot *j_0* ∈ HS such that CEncrypt((j_0, x_{j_0}^{t*}, x_{j_0}^{t*}, t*) is never asked. This ensures that none of the PRF values {k_{ij,t*} = ̂F_{k_{ij}}(t*)}_{i ∈ HS} is ever exposed to the adversary, which implies that {k_{ij,t*}}_{i ∈ HS} are pseudorandom in the adversary’s view since *F* is a PRF. This implies that secret keys K_{i,t*,i} := ∑_{j=1}^{ℓ} k_{ij,t*} are also pseudorandom for all *i* ∈ HS. Hence, the ciphertexts C_{t*,i} = Encr(*ek_i*, x_{i}, t) ⊕ G(K_{t*,i}) computationally hide the x_{i}’s for all uncorrupted slots *i*. To translate this intuition into a formal security proof, we need a PRF family that provides security in the sense of Definition A.2.

**Theorem A.3.** If MCFE′ is 1Ch-IND secure then the compiled MCFE scheme is ≤ 1Ch-IND secure.

**Proof.** Given an adversary *A* that breaks the MCFE scheme by winning the game of Definition A.1 with non-negligible probability, we give a construction of an adversary *B* that wins with the game of Definition 2.12 with noticeable advantage, thus breaking the MCFE′ scheme. The adversary *B* works as follows.

**Initialization:** The challenger *C* gives *B* the public parameters mpk′ and keeps to itself the secret keys (msk′, (ek′_{i})_{i=1}^{ℓ}). In order to initialize the game with *A*, the reduction *B* samples ℓ^2 secret keys k_{ij} ← K for all *i*, *j* ∈ [ℓ]. Then gives *A* the public parameters mpk := mpk′.

**Encryption queries:** For each encryption query QEncrypt(*i*, *x*, *t* made by *A*, *B* sends to same query to its challenger and obtains C_{t,i} = Encr(*ek_i*, *x*, *t*).

Then, *B* computes {k_{ij,t} := F_{ki_j}(t), k_{ij,t} := F_{ki_j}(t)}_{i=1}^{ℓ}, K_{t,i} := ∑_{j=1}^{ℓ} k_{ij,t}, C_{t,i} := C_{t,i} ⊕ G(K_{t,i}) and gives ct_{t,i} := (C_{t,i}, {k_{ij,t}}_{j=1}^{ℓ}) to *A*.

**Challenge queries:** If *A* makes a query CEncrypt(i, x_i^{t^0}, x_i^{t^1}, t*), *B* relays it to its challenger and obtains C_{t*,i} := Encr(*ek_i*, x_i^{t^0}, t*). Then, *B* computes {k_{ij,t*} := F_{ki_j}(t*), k_{ij,t*} := F_{ki_j}(t*)}_{i=1}^{ℓ}, K_{t*,i} := ∑_{j=1}^{ℓ} k_{ij,t*}, as well as C_{t*,i} := C_{t*,i} ⊕ G(K_{t*,i}). The adversary *A* is given ct_{t*,i} := (C_{t*,i}, {k_{ij,t*}}_{j=1}^{ℓ}).

**Functional decryption key queries:** Each query QDKeygen(*f*) made by *A* is relayed by *B* to its challenger, which replies with dk_f. Then, *B* gives dk_f := dk_f to *A*.

**Corruption queries:** For each corruption query QCorrupt(*i*) that *A* makes, *B* replies with ek_i := (ek_i′, {k_{ij}, k_{ji}}_{j=1}^{ℓ}), where ek_i′ is the key given by its challenger in response to the same corruption query.

**Finalize:** When *A* outputs a result *b*, *B* distinguishes two situations.
- If $A$ makes queries $\text{CQEnc}(i, *, *, t^*)$ for all slots $i \in \mathcal{HS}$ and the conditions of the Finalize step in Definition A.1 are all satisfied (i.e., $A$'s output is not overwritten by a random bit $\beta$), $B$ outputs the same bit $\hat{b} := b'$ as $A$.
- Otherwise, $B$ outputs a random $\hat{b} \leftarrow U(\{0, 1\})$. We call this event $\text{abort}$.

Next, we analyze the probability of $B$ winning the game.

$$
\Pr[\hat{b} = b] = \Pr[\hat{b} = b | \text{abort}] \cdot \Pr[\text{abort}] + \Pr[\hat{b} = b | \text{abort}] \cdot \Pr[\text{abort}]
$$

$$
= \frac{1}{2} \cdot \Pr[\text{abort}] + \Pr[\hat{b} = b | \text{abort}] \cdot \Pr[\text{abort}]
$$

$$
= \Pr[\hat{b} = b] + \left(1 - \frac{1}{2} \cdot \Pr[\hat{b} = b | \text{abort}]\right) \cdot \Pr[\text{abort}]
$$

$$
= \Pr[\hat{b} = b] + \left(1 - \frac{1}{2} \cdot \Pr[\hat{b} = b | \text{abort}]\right) \cdot \Pr[\text{abort}]
$$

where $\text{Abort}$ is the event that $A$ does not make encryption queries for all honest slots in the challenge phase and the conditions in the finalize step of Definition A.1 are all satisfied.

In order to finish the proof, we need to prove the following claim.

**Claim.** If $F$ is a ad-mi-PRF and $G$ a PRG, then $\left| \Pr[\hat{b} = b | \text{abort}] - \frac{1}{2} \right|$ is negligible.

**Proof.** We will prove this claim via a series of games. For each $i \in \{0, 1, 2\}$, we call $W_i$ the event that the challenger outputs $1$ in $\text{Game}_i$.

**Game}_0$: This is the original game in the one-or-less security experiment (cf. Definition A.1) for the compiled scheme. The challenger runs the setup algorithm of MCFE$'$ to obtain $(\text{mpk}, \text{msk}', \{\text{ek}_i\}_{i=1}^\ell)$, samples $k_{ij} \leftarrow \mathcal{K}$, for all $i, j \in [\ell]$ and sets $\text{mpk} := \text{mpk}'$, $\text{msk} := \text{msk}'$ and $\text{ek}_i := (\text{ek}'_i, \{k_{ij}, k_{ji}\}_{j=1}^\ell)$. It gives $\text{mpk}$ to $A$ and handles queries $\text{QCorrupt}(i)$ using $\text{ek}_i := (\text{ek}'_i, \{k_{ij}, k_{ji}\}_{j=1}^\ell)$. To answer query $\text{QEnc}(i, x_i, t)$, it computes $\{k_{ij} := F_{k_{ij}}(t), k_{ji} := F_{k_{ji}}(t)\}_{j=1}^\ell$ and $K_{t,i} := \bigoplus_{j=1}^\ell k_{ij}$. Then, it computes $C_{t,i} := \text{Encrypt}'(\text{ek}'_i, x_i, t) + G(K_{t,i})$. The adversary $A$ is given $C_{t,i} := (C_{t,i}, \{k_{ij}\}_{j=1}^\ell)$. To answer challenge encryption queries $\text{CQEncrypted}(t, x_i^0, x_i^1, t^*)$, it replies with the result of $\text{QEnc}(i, x_i^0, t^*)$. As for functional decryption queries $\text{QDKeygen}(f, \text{msk})$, the challenger replies with $\text{dk}_f \leftarrow \text{DKeygen}(\text{msk}', f)$. When $A$ halts, it outputs a bit $b'$ and the challenger outputs $1$ if and only if $b' = b$, so that $\Pr[W_0] = \Pr[b' = b | \text{Abort}]$.

**Game}_1$: The challenger interacts with $A$ exactly as in $\text{Game}_0$ except that, upon receiving a challenge query $\text{CQEncrypted}(i, x_i^0, x_i^1, t^*)$, it does the following. If $x_i^0 = x_i^1$, it replies as in $\text{Game}_0$. Otherwise, for each slot $i \in [\ell]$ such that $x_i^0 \neq x_i^1$, it samples a uniform value for $K_{t^*, i} \leftarrow U(\{0, 1\}^\lambda)$ which it uses as a seed for the pseudorandom generator $G$ in order to compute $C_{t^*, i}$.

For any index $i$ such that $x_i^0 \neq x_i^1$, corruption queries $\text{Corrupt}(i)$ are disallowed by the conditions of the Finalize step which must be satisfied (recall that we are
conditioning on event Ab). The only information that \( \mathcal{A} \) can gather about \( K_{t,i} \) is thus obtained from \( \text{QEnc}(j, x_j^0, t) \) and \( \text{CQEnc}(j, x_j^0, x_j^1, t^*) \) queries. By making these types of queries, \( \mathcal{A} \) only learns the evaluations \( \{k_{ijt} = F_{k_{ijt}}(t)\}_{j \in [\ell]}, \) for any \( t \in \mathcal{T} \setminus \{t^*\}, \) and \( \{F_{k_{ijt}}(t^*)\}_{j \in [\ell]} \), respectively. Since we are conditioning on Ab, there exists an index \( j_0 \in \mathcal{HS} \) which is never the input of a query of the form \( \text{CQEnc}(j_0, \cdot, \cdot, t^*). \) In order to prove that \( \text{Game}_0 \) and \( \text{Game}_1 \) are computationally indistinguishable, we define \( \text{Game}'_0 \) and \( \text{Game}'_1 \) in the following way.

**Game \(_b\) \((b \in \{0, 1\})\):** This game is identical to \( \text{Game}_b \) with the difference that the challenger initially chooses \( j_0 \leftarrow U([\ell]) \) as a guess for the smallest index \( j_0 \in \mathcal{HS} \) such that no challenge query \( \text{CQEnc}(j_0, \cdot, \cdot, t^*) \) is ever made. The challenger aborts and outputs 0 if the guess eventually turns out to be wrong. For each \( b \in \{0, 1\} \), we have \( \Pr[W_b] = \ell \cdot \Pr[W'_b]. \)

Lemma A.4 shows that \( \text{Game}'_0 \) and \( \text{Game}'_1 \) are computationally indistinguishable if \( F \) is a secure PRF in the multi-instance setting. This implies

\[
|\Pr[W_0] - \Pr[W'_1]| \leq \ell \cdot \text{Adv}^{\text{ad-mi-PRF}}(\lambda).
\]

**Game\(_1\):** This game identical to \( \text{Game}_1 \) except that, at each challenge query \( \text{CQEncrypt}(i, x^*_i, x^1_i, t^*) \) for which \( x^0_i \neq x^1_i \), it also sets \( C_{t,i} \) uniformly random. In this game, we have \( \Pr[W_2] = \Pr[b = b|\text{Ab}] = \frac{1}{2}. \)

To see that \( \text{Game}_1 \) and \( \text{Game}_2 \) are computationally indistinguishable, recall that \( C_{t,i} = C_{t,i} \oplus G(K_{t,i}). \) When the seed \( K_{t,i} \) is uniformly generated the value \( G(K_{t,i}) \) is pseudorandom, thus \( \text{Game}_1 \) and \( \text{Game}_2 \) are computationally indistinguishable. We thus obtain \( |\Pr[W_2] - \Pr[W_1]| \leq \text{Adv}^{\text{PRG}}(\lambda). \)

When combining the above, we obtain

\[
|\Pr[W_0] - \Pr[W_2]| \leq \ell \cdot \text{Adv}^{\text{ad-mi-PRF}}(\lambda) + \text{Adv}^{\text{PRG}}(\lambda),
\]

which proves the result.

\[ \square \]

**Lemma A.4.** If \( F \) is secure in the ad-mi-PRF sense, then \( \text{Game}'_0 \) and \( \text{Game}'_1 \) are computationally indistinguishable. More precisely, there exists a PRF adversary \( \mathcal{B} \) such that \( |\Pr[W_0] - \Pr[W'_1]| \leq \text{Adv}^{\text{ad-mi-PRF}}(\lambda). \)

**Proof.** Suppose that \( \mathcal{A} \) is an adversary that is able to distinguish between the two games. We construct an adversary \( \mathcal{B} \) that breaks the ad-mi-PRF security of \( F \) when the challenger samples \( N = \ell^2 \) secret keys \( k_{ij} \leftarrow \mathcal{K} \) for all \( i, j \in [\ell] \) in the game of Definition A.2. Algorithm \( \mathcal{B} \) proceeds in the following way.

**Initialization:** \( \mathcal{B} \) starts by drawing \( j_0 \leftarrow U([\ell]) \) as a guess that \( j_0 \) is the smallest index such that no challenge query \( \text{CQEncrypt}(j_0, \cdot, \cdot, t^*) \) is made. Then, it runs \( (\text{mpk}', \text{msk}', \{e^i_j\}_{i=1}^\ell) \leftarrow \text{Setup}(1^\lambda) \) and \( \text{mpk} := \text{mpk}' \) given to \( \mathcal{A} \).

**Corruption queries:** In order to answer a corruption query \( \text{Corrupt}(j) \), the reduction \( \mathcal{B} \) aborts if \( j = j_0 \) since its guess for \( j_0 \) was incorrect. Otherwise, it obtains the values \( \{k_{j\tau}, k_{j\tau}\}_{\tau=1}^\ell \) by making corruption queries to its PRF
challenger. Then, it replies with \((\text{ek'}_j, \{k_{ij}, k_{ijt}\})_{j=1}^\ell\). Notice that, if \(j_0\) is a correct guess, we have \(j_0 \in \mathcal{HS}\), which implies that the adversary never learns \(\{k_{ij_0}, k_{ij_0}\}_{i \in \mathcal{HS}}\) from corruption queries.

**Encryption queries:** To answer \(\mathsf{QEncrypt}(i, x, t)\) queries, \(B\) asks its PRF challenger for the evaluations \(\{k_{ijt} := F_{k_{ij}}(t), k_{ij} := F_{k_{ij}}(t)\}_{j=1}^\ell\) and computes \(K_{t,i} := \bigoplus_{j=1}^\ell k_{ijt}\). Then, it computes \(C'_{t,i} = \text{Encrypt}'(\text{ek'}_j, x, t)\) as well as \(C_{t,i} := C'_{t,i} \oplus G(K_{t,i})\). The ciphertext \(c_{t,i} := (C_{t,i}, \{k_{ijt}\}_{j=1}^\ell)\) is given to \(A\).

**Challenge queries:** When receiving a challenge query \(\mathsf{CQEncrypt}(i, x^*_0, x^*_1, t^*)\), \(B\) aborts if \(i = j_0\). Otherwise, it considers the following two cases.

- If \(x^*_0 = x^*_1 = x^*_i\), then \(B\) responds as it would answer the encryption query \(\mathsf{QEncrypt}(i, x^*_i, t^*)\).

- If \(x^*_0 \neq x^*_1\) (which is only possible if \(i \in \mathcal{HS}\)), the reduction \(B\) asks its challenger for the PRF evaluations \(\{k_{ijt^*} := F_{k_{ij}}(t^*)\}_{j \neq j_0}\). It also makes a challenge query for the input \(t^*\) and the secret key \(k_{ij_0}\). The challenger replies with a value \(y_{ij_0t^*}\), which is either the evaluation \(F_{k_{ij_0}}(t^*)\) for the unknown secret key \(k_{ij_0}\) chosen by the PRF challenger or a random value. Then, \(B\) computes \(K'_{t^*,i} := (\bigoplus_{j \neq j_0} k_{ijt^*}) \oplus y_{ij_0t^*}\) and uses \(K'_{t^*,i}\) as a seed for the PRG, by computing \(C'_{t^*,i} := C'_{t^*,i} \oplus G(K'_{t^*,i})\). The adversary \(A\) is given \(c_{t^*,i} := (C'_{t^*,i}, \{k_{ijt^*}\}_{j=1}^\ell)\).

**Guess:** When \(A\) halts, it outputs a bit \(b'\). At this point, \(B\) aborts and outputs 0 if \(j_0\) is not the smallest \(i \in [\ell]\) such that no query \(\mathsf{QEncrypt}(i, \cdot, \cdot, t^*)\) is made. Otherwise, \(B\) outputs 1 (meaning that its challenger always outputs pseudorandom values) if \(b' = b\) and 0 otherwise.

If \(j_0\) is not a correct guess, \(\text{Game}'_0\) and \(\text{Game}'_1\) have the same outcome since the challenger outputs 0 in both games. We thus assume that \(j_0\) is a correct guess, in which case we know that \(j_0 \in \mathcal{HS}\). Moreover, if \(x^*_0 \neq x^*_1\), we necessarily have \(i \in \mathcal{HS}\). Therefore the reduction \(B\) never obtains the secret key \(k_{ij_0}\) by making a \(\mathsf{Corrupt}(\cdot)\) query in the game of Definition A.2. In the same game, we also observe that the value \(F_{k_{ij_0}}(t^*)\) is never asked by \(B\) to its PRF evaluation oracle. Indeed, encryption queries \(\mathsf{QEncrypt}(\cdot, \cdot, t^*)\) are not allowed on the challenge tag \(t^*\) and the constraint \(\mathsf{Ab}\) implies that no challenge query \(\mathsf{QEncrypt}(j_0, \cdot, \cdot, t^*)\) can be made for the index \(j_0\).

We now observe that, if \(B\)’s challenger always returns truly random values \(y_{ij_0t^*}\) at each challenge query, then \(A\)’s view is identical to that of \(\text{Game}'_1\). If \(B\)’s challenger always returns pseudorandom values, \(A\)’s view is as in \(\text{Game}_0\). We thus conclude that \(\mathsf{Adv}_B^{\text{ad-mi-PRF}}(\lambda) \geq |\mathsf{Pr}[W^*_0] - \mathsf{Pr}[W^*_1]|\) \(\Box\)

### A.2 A PRF Construction With Multi-Instance Security in the Adaptive Corruption Setting

We show that the centralized version of the distributed PRF of Libert et al. [56] provides security in the sense of Definition A.2. We first recall the construction.
Let \( \ell \) be a security parameter and let \( \ell \in \Theta(\lambda), L \in \Theta(\lambda) \). The scheme uses public parameters consisting of prime moduli \( p \) and \( q \) such that \( q/p > 2^{L+\lambda} \cdot N \cdot r \), dimensions \( n, m, k \in \text{poly}(\lambda) \) such that \( m \geq 2n \cdot \lceil \log q \rceil \), an integer \( \beta > 0 \), a real \( \alpha > 0 \) and a rounding parameter \( r = m^{2+\alpha} \cdot n \cdot \alpha q \). The PRF family relies on the following lemmas.

- A balanced admissible hash function \( \text{AHF} : \{0,1\}^L \rightarrow \{0,1\}^L \).
- A family \( H_\lambda \) of \( \xi \)-wise independent hash functions \( \pi_i : \mathbb{Z}_p^m \rightarrow \mathbb{Z}_p^k \) for a suitable \( \xi > 0 \) that will be determined later on. Let a random member \( \pi \) of \( H_\lambda \).

Let a Gaussian parameter \( \sigma > 0 \) such that the interval \( [-\beta, \beta] = [-\sigma \sqrt{n}, \sigma \sqrt{n}] \) contains the coordinates of the secret key with overwhelming probability. The PRF family assumes the availability of public parameters

\[
\text{pp} := \left( q, \pi, A_0, \{A_{i,0}, A_{i,1} \in \mathbb{Z}_q^{n \times m} \}_{i=1}^L, \text{AHF}, r, \sigma \right),
\]

where \( A_0 \sim U(\mathbb{Z}_q^{n \times m}) \) and \( A_{i,0}, A_{i,1} \sim U(\mathbb{Z}_q^{n \times m}) \) for each \( i \in [L] \). Importantly, \( \{A_{i,0}, A_{i,1}\}_{i=1}^L \) should be chosen in such a way that \( G^{-1}(A_{i,b}) \in \mathbb{Z}_p^{m \times m} \) is \( \mathbb{Z}_q \)-invertible for all \( i \in [L] \) and \( b \in \{0,1\} \) (see [56, Section 3.1] for details).

**Keygen(pp):** Given \( \text{pp} \), sample a vector \( s \leftarrow D_{\mathbb{Z}^n, \sigma} \) so that \( \|s\|_\infty < \beta = \sigma \sqrt{n} \) with overwhelming probability. The secret key is \( SK := s \in [-\beta, \beta]^n \).

**Eval(pp, SK, X):** Given \( SK = s \in \mathbb{Z}^n \) and an input \( X \in \{0,1\}^L \),

1. Compute \( x = \text{AHF}(X) \in \{0,1\}^L \) and parse it as \( x = x_1 \ldots x_L \).
2. Compute \( A(x) = A_0 \cdot \prod_{i=1}^L G^{-1}(A_{i,x_i}) \) and

\[
z = \left( (A(x))^\top \cdot s \right)_p \in \mathbb{Z}_p^m,
\]

and output \( y = \pi(z) \in \mathbb{Z}_p^k \).

We now prove that this construction is secure in the sense of Definition A.2 under the LWE assumption. The proof of Theorem A.5 is essentially identical to that of [56, Theorem 3.2]. Intuitively, it exploits the fact that the reduction knows the secret key at any time, which makes it easy to consistently answer adaptive corruption queries.

**Theorem A.5.** The above construction provides adaptive multi-instance security under the LWE assumption.

The proof relies on the following lemmas.

**Lemma A.6 (Adapted from [59, Lemma 4.4]).** For any \( n \)-dimensional lattice \( \Lambda, x', c \in \mathbb{R}^n \) and symmetric positive definite \( \Sigma \in \mathbb{R}^{n \times n} \) satisfying \( \sigma_n(\sqrt{\Sigma}) \geq \eta_{2-n}(\Lambda) \), we have

\[
\rho_{\Sigma, c}(\Lambda + x') \in \left[ 1 - 2^{-n}, 1 + 2^{-n} \right] \cdot \det(\Sigma)^{1/2} / \det(\Lambda).
\]
Lemma A.7 ([36, Lemma 5.3]). Let \( m \geq 2n \cdot \log q \) and \( q \geq 2 \) prime and let \( A \leftarrow U(\mathbb{Z}_q^{n \times m}) \). With probability at least 1 − \( 2^{-O(m)} \), we have \( \lambda_{\infty}^\infty (A(A)) \geq q/4 \).

Lemma A.8 ([32, Corollary 3]). Fix any integers \( \bar{n}, m, M \), any real \( \varepsilon < 1 \) and any collection \( \mathcal{X} \) of \( M \) distributions over \( \{0, 1\}^m \) of min-entropy \( \bar{n} \) each. Define

\[
\xi = \bar{n} + \log M, \quad \bar{k} = \bar{n} - \left( 2 \log \frac{1}{\varepsilon} + \log \log M + \log \bar{n} + O(1) \right),
\]

and let \( \mathcal{F} \) be any family of \( \xi \)-wise independent functions from \( \bar{m} \) bits to \( \bar{k} \) bits. With probability at least \( (1 - 1/M) \), a random function \( f \leftarrow U(\mathcal{F}) \) is a good deterministic extractor for the collection \( \mathcal{X} \). Namely, the distribution \( f(X) \) is \( \varepsilon \)-close to \( U(\{0, 1\}^k) \) for any distribution \( X \in \mathcal{X} \).

Lemma A.9. Let \( \mathcal{F} \) be a family of \( \delta \)-sure \( \epsilon \)-extractor for the finite collection \( \mathcal{X} \) of distributions over the input space of \( \mathcal{F} \): namely, \( \mathcal{F} \) satisfies

\[
\Pr_{f \leftarrow \mathcal{F}} [ \forall X \in \mathcal{X} : \Delta(f(X), U) \leq \epsilon ] \geq 1 - \delta,
\]

where \( U \) is the uniform distribution over \( \mathcal{F} \) and \( U \) is the uniform distribution over the range of \( \mathcal{F} \). For any independent distributions \( Z_1, Z_2, \ldots, Z_\ell \in \mathcal{X} \), we have \( \Delta((F, F(Z_1)), F(Z_2), \ldots, F(Z_\ell)), (F, U_1, U_2, \ldots, U_\ell) \leq \ell \cdot \epsilon + \delta \).

Proof. For any function \( f \in \mathcal{F} \), the distributions \( f(Z_1), f(Z_2), \ldots, f(Z_\ell) \) are independent since the initial distributions \( Z_1, Z_2, \ldots, Z_\ell \) are independent. This implies that for, any function \( f \in \mathcal{F} \), we have

\[
\Delta(f(Z_1), f(Z_2), \ldots, f(Z_\ell), (U_1, U_2, \ldots, U_\ell)) \leq \sum_{j=1}^\ell \Delta(f(Z_j), U_j).
\]

Let \( U_{1\ell} := (U_1, U_2, \ldots, U_\ell) \). For any \( f \in \mathcal{F} \), we also define the distribution \( Z_f := (f(Z_1), f(Z_2), \ldots, f(Z_\ell)) \). Since \( \mathcal{F} \) is a \( \delta \)-sure \( \epsilon \)-extractor for the collection \( \mathcal{X} \), it follows that

\[
\Pr_{f \leftarrow \mathcal{F}} [ \Delta(Z_f, U_{1\ell}) \leq \ell \cdot \epsilon ] \geq 1 - \delta.
\]

By using the identity \( \Delta((F, Z_F), (F, U_{1\ell})) = \Exp_{f \leftarrow \mathcal{F}} [\Delta(Z_f, U_{1\ell})] \), we obtain that \( \Delta((F, Z_F), (F, U_{1\ell})) \leq \ell \epsilon \cdot (1 - \delta) + \delta < \ell \cdot \epsilon + \delta \). \( \square \)

Lemma A.10 ([10, Lemma 2.7]). Let \( p, q \) be positive integers such that \( p < q \). Given \( R > 0 \) an integer, the probability that there exists \( \varepsilon \in [-R, R] \) such that \( |y|_p \neq |y + \varepsilon|_p \), when \( y \leftarrow U(\mathbb{Z}_q) \), is smaller than \( \frac{2R}{q} \).

Proof (of Theorem A.5). The proof considers a sequence of hybrid games. In each game, we call \( W_i \) the event that \( b' = b \).
**Game₀:** This is the real experiment of Definition A.2, where the challenger initially samples \( N \) secret keys \( SK_j = s_j \sim \mathcal{D}_{\mathbb{Z}^{n \cdot \sigma}} \) which are revealed to the adversary \( \mathcal{A} \) in case of corruption queries. At each evaluation query \((j, X)\), the challenger replies by returning

\[
y_j = \pi \left( [ (A(x))^\top \cdot s_j ]_p \right) \in \mathbb{Z}_p^k,
\]

where \( A(x) = A_0 \cdot \prod_{i=1}^L G^{-1}(A_{i,x_i}) \). When \( \mathcal{A} \) halts, it outputs \( \hat{d} \in \{0, 1\} \) and the challenger defines \( d' := \hat{d} \). By definition, the adversary’s advantage is \( \text{Adv}(\mathcal{A}) := |\Pr[W_0] - 1/2| \), where \( W_0 \) is event that \( d' = d \).

**Game₁:** This game is like Game₀ with some changes. First, the challenger runs

\( K \leftarrow \text{AdmSmp}(1^\lambda, Q, \delta) \)

\( \) to generate a key \( K \in \{0, 1, \bot\}^L \) for a balanced admissible hash function \( AHF : \{0, 1\}^L \rightarrow \{0, 1\}^L \). When the adversary eventually outputs \( \hat{d} \in \{0, 1\} \), the challenger checks if the conditions

\[
P_K(X^{(1)}) = \cdots = P_K(X^{(Q)}) = 1 \land P_K(X^*) = 0 \quad (19)
\]

are satisfied, where \( X^* \) is the challenge input and \( X^{(1)}, \ldots, X^{(Q)} \) are the evaluation queries. If these conditions do not hold, the challenger ignores \( \mathcal{A}'s \) output \( \hat{d} \in \{0, 1\} \) and replaces it by a random bit \( d'' \leftarrow \{0, 1\} \) to define \( d' = d'' \). If the conditions of (19) are satisfied, the challenger sets \( d' = \hat{d} \).

By Lemma 2.8, we have

\[
|\Pr[W_1] - 1/2| = |\Pr[d' = d] - 1/2| \geq \gamma_{\min} \cdot \text{Adv}(\mathcal{A}) = \frac{1}{2} \cdot (\gamma_{\max} - \gamma_{\min}) = \tau,
\]

where \( \tau(\lambda) \) is a noticeable function.

**Game₂:** In this game, we modify the generation of \( pp \). Initially, the challenger samples a uniformly random matrix \( A \leftarrow U(\mathbb{Z}_q^{n \times m}) \). For each \( i \in [L] \), it samples \( R_{i,0}, R_{i,1} \leftarrow U(\{-1, 1\}^{m \times m}) \) and defines \( \{A_{i,0}, A_{i,1}\}_{i=1}^L \) as follows for all \( i \in [L] \) and \( b \in \{0, 1\} \):

\[
A_{i,b} := \begin{cases} A \cdot R_{i,b} & \text{if } (b \neq K_i) \land (K_i \neq \bot) \\ A \cdot R_{i,b} + G & \text{if } (b = K_i) \lor (K_i = \bot) \end{cases}
\]

It also defines \( A_0 = A \cdot R_0 + G \) for a random \( R_0 \leftarrow U(\{-1, 1\}^{m \times m}) \). Since \( A \sim U(\mathbb{Z}_q^{n \times m}) \), the Leftover Hash Lemma implies that \( \{A_{i,0}, A_{i,1}\}_{i=1}^L \) are statistically independent and uniformly distributed over \( \mathbb{Z}_q^{n \times m} \). The challenger keeps sampling \( R_{ib} \) until \( G^{-1}(A_{ib}) \) is \( \mathbb{Z}_q \)-invertible in such a way that the simulated \( A_{ib} \) are statistically uniform under the constraint that \( G^{-1}(A_{ib}) \) be invertible over \( \mathbb{Z}_q \). The analysis of [56] shows that the probability that \( G^{-1}(A_{ib}) \) is invertible in \( \mathbb{Z}_q \) is roughly \( \left( \exp(-3\sqrt{\exp(1)}) \right)^2 \), so that the public parameters can be simulated in polynomial time. Since the distribution of \( pp \) is statistically unchanged, it follows that \( |\Pr[W_2] - \Pr[W_1]| \leq L \cdot 2^{-\lambda} \).
At each query $X$, we can view $A(x)$ as a GSW encryption

$$A(x) = A \cdot R_x + (\prod_{i=1}^{L} \mu_i) \cdot G,$$

for some small norm $R_x \in \mathbb{Z}^{m \times m}$, where

$$\mu_i := \begin{cases} 
0 & \text{if } (AHF(X)_i \neq K_i) \land (K_i \neq \perp) \\
1 & \text{if } (AHF(X)_i = K_i) \lor (K_i = \perp)
\end{cases}$$

If the conditions (19) are satisfied, at each evaluation query $(j, X^{(i)})$, the admissible hash function ensures that $x^{(i)} = AHF(X^{(i)})$ satisfies

$$A(x^{(i)}) = A \cdot R_{x^{(i)}},$$

for some small norm $R_{x^{(i)}} \in \mathbb{Z}^{m \times m}$. Moreover, the unique challenge input $X^*$ is mapped to an $L$-bit string $x^* = AHF(X^*)$ such that

$$A(x^*) = A \cdot R_{x^*} + G.$$

**Game$_3$:** We modify the distribution of $pp$ and replace the uniform $A \in \mathbb{Z}_{q}^{n \times m}$ by a lossy matrix of the form

$$A^T = \tilde{A}^T \cdot C + E \in \mathbb{Z}_{q}^{m \times n},$$

where $A \leftrightarrow U(\mathbb{Z}_{q}^{n^\prime \times m})$, $C \leftrightarrow U(\mathbb{Z}_{q}^{n^\prime \times n})$ and $E \leftrightarrow D_{\mathbb{Z}_{q}^{m \times n \cdot q}}$, for $n^\prime \ll n$. Under the LWE assumption in dimension $n'$, this change should not affect $A$’s output distribution and we can easily build an LWE distinguisher $B$ shows that $|Pr[W_3] - Pr[W_2]| \leq n \cdot Adv_{\mathbb{Z}_{q}^{n^\prime \times m}, q, \cdot, \cdot}^{LWE}(\lambda)$, where the factor $n$ comes from the use of an LWE assumption with $n$ secrets.

Due to the modification introduced in Game$_3$, if the conditions (19) are satisfied, each evaluation query $(j, X^{(i)})$ obtains a response of the form

$$\left[(A \cdot R_{x^{(i)}})^{\top} \cdot s_j\right]_p = \left[(R_{x^{(i)}}^{\top} \cdot \tilde{A}^{\top} \cdot C + R_{x^{(i)}}^{\top} \cdot E) \cdot s_j\right]_p.$$

Note that the right-hand-side member of (24) uniquely determines $C \cdot s_j$ with high probability: observe that $R_{x^{(i)}}^{\top} \cdot \tilde{A}^{\top}$ is statistically uniform over $\mathbb{Z}_{q}^{n^\prime \times n'}$, so by Lemma A.7, the quantity $[R_{x^{(i)}}^{\top} \cdot \tilde{A}^{\top} \cdot (C \cdot s_j)]_p$ is an injective function of $C \cdot s \mod q$. It comes that evaluation queries $(j, X)$ information-theoretically reveal $C \cdot s_j \mod q$.

**Game$_4$:** We modify the evaluation oracle and introduce a bad event. We define $bad_x$ to be the event that for the input $x \in \{0, 1\}^L$ such that $A(x) = A \cdot R_x$, for some small-norm $R_x \in \mathbb{Z}^{m \times m}$, we have

$$\left[(A \cdot R_x)^{\top} \cdot s_j\right]_p \neq \left[(R_x^{\top} \cdot \tilde{A}^{\top} \cdot C) \cdot s_j\right]_p.$$
for some \( j \in [N] \). We also define \( \text{BAD} \) as the event that \( A \) makes an evaluation query \((j, X)\) such that the event \( \text{bad}_x \) occurs. Note that the challenger can detect \( \text{bad}_x \) since it knows \( A \in \mathbb{Z}_q^{n \times m} \), \( C \in \mathbb{Z}_q^{n \times n} \) and \( E \in \mathbb{Z}^{m \times n} \) satisfying (23). If \( \text{bad}_x \) occurs, the challenger replaces \( A' \)’s output \( \hat{d} \) by a random \( d'' \leftarrow \{0,1\} \) and sets \( d' = d'' \). Otherwise, it sets \( d' = \hat{d} \) as before.

The proof of the following claim is adapted from [56, Appendix D].

**Claim.** If \( q/p > 2^{L+\lambda} \cdot N \cdot r \), where \( r = m^{L+2} \cdot n \cdot \beta \cdot \alpha q \), we have the inequality

\[
|\Pr[W_4] - \Pr[W_3]| \leq \Pr[\text{BAD}] \leq 2^{-\Omega(\lambda)}.
\]

**Proof.** We use the fact that, for each query \( x = x_1 \ldots x_L \), the matrix \( R_x \in \mathbb{Z}^{m \times m} \) is of the form

\[
R_x = R_0 \cdot \prod_{i=1}^{L} G^{-1}(A_{i,x_i})
\]

\[
+ R_{1,x_1} \cdot \prod_{i=2}^{L} G^{-1}(A_{i,x_i})
\]

\[
+ \mu_{1,x_1} \cdot R_{2,x_2} \cdot \prod_{i=3}^{L} G^{-1}(A_{i,x_i})
\]

\[
\vdots
\]

\[
+ \mu_{1,x_1} \mu_{2,x_2} \cdots \mu_{L-2,x_{L-2}} \cdot R_{L-1,x_{L-1}} \cdot G^{-1}(A_{L,x_L})
\]

\[
+ \mu_{1,x_1} \mu_{2,x_2} \cdots \mu_{L-1,x_{L-1}} \cdot R_{L,x_L}
\]

In order to bound \( \Pr[\text{bad}_x] \), we note that, as argued in [10, Proof of Theorem 7.3], \( R_0^T \cdot \tilde{A}^T \) is statistically uniform over \( \mathbb{Z}_q^{m \times n'} \), conditionally on \( R_0^T \cdot E \in \mathbb{Z}^{m \times n} \). Hence, for a fixed choice of \( j \in [N] \) and \( x \in \{0,1\}^L \), the product \( R_x^T \cdot \tilde{A}^T \cdot (C \cdot s_j) \) contains a term

\[
(\prod_{i=1}^{L} G^{-1}(A_{i,x_i}))^T \cdot R_0^T \cdot \tilde{A}^T \cdot (C \cdot s_j),
\]

for which we know that \( G_x^T \cdot R_0^T \cdot \tilde{A}^T \) is statistically uniform over \( \mathbb{Z}_q^{m \times n'} \) since \( G_x^T \) is a \( \mathbb{Z}_q \)-invertible matrix by construction. We know that \( G_x^T \cdot R_0^T \cdot \tilde{A}^T \cdot (C \cdot s_j) \) is statistically close to the uniform distribution over \( \mathbb{Z}_q^m \). We thus have

\[
\Pr[\text{bad}_x] = \Pr \left[ \left( (A \cdot R_x)^T \cdot s_j \right)_p \neq \left( (R_x^T \cdot \tilde{A}^T \cdot C) \cdot s_j \right)_p \right] \leq m \cdot \frac{2rp}{q}. \tag{26}
\]

where \( r = m^{L+2} \cdot n \cdot \beta \cdot \alpha q \). Indeed, we have \( \|R_x^T\|_\infty \leq \frac{m(m^{L+1}-1)}{m-1} \), \( \|s_j\|_\infty \leq \beta \) and

\[
\|E\|_\infty = \max_{i \in [m]} \left( \sum_{j=1}^{n} |e_{i,j}| \right) \leq \sqrt{n} \max_{i \in [m]} \|e_i\| \leq n \cdot \alpha q.
\]
The conditions of Lemma A.10 are satisfied since

\[ \|R^T \cdot E \cdot s\|_\infty \leq \|R^T\|_\infty \cdot \|E\|_\infty \cdot \|s\|_\infty \leq \frac{m(m^L+1) - 1}{m - 1} \cdot n \cdot \alpha q \cdot \beta \leq r, \]

which yields inequality (26).

By taking the union bound over all possible inputs \( x \in \{0,1\}^L \) and all indexes \( j \in [N] \), we can bound the probability \( \Pr[BAD] \leq N \cdot 2^L \cdot m \cdot \frac{2rp}{q} \).

Since \( \text{Game}_3 \) and \( \text{Game}_4 \) are identical if \( BAD \) does not occur, we obtain the inequality

\[ |\Pr[W_4] - \Pr[W_3]| \leq \Pr[BAD] \leq 2^L \cdot N \cdot m \cdot \frac{2rp}{q} = 2m \cdot 2^{-\lambda} \]

which is negligible in \( \lambda \).

Recall that, if \( BAD \) does not occur, we have

\[ \left[(A \cdot R_{x(i)})^T \cdot s_j\right]_p = \left[(R^T \cdot \tilde{A}^T \cdot C) \cdot s_j\right]_p \] \[ \quad \text{(27)} \]

at each query \( (j, X^{(i)}) \). We note that the right-hand-side member of (27) is completely determined by \( R_{x(i)} \cdot \tilde{A}^T \) and the product \( C \cdot s_j \in \mathbb{Z}_q^{n'} \). This means that, for each uncorrupted index \( j \in [N] \), \( \mathcal{A}'s \) evaluation queries \( (j, X) \) always reveal the same \( n' \cdot \log q \) bits of information about \( s_j \in \mathbb{Z}^n \). Consequently, each uncorrupted \( SK_j = s_j \in \mathbb{Z}_n \) retains a lot of entropy after all evaluation queries.

**Game5:** We modify the challenge oracle. If \( d = 1 \), at each challenge query \( (j, X^*) \), the adversary obtains a fresh random \( y^*_j \) \( U(\mathbb{Z}_p^n) \) instead of the real evaluations \( y = \pi((A(x))^T \cdot s_j)_p \). Clearly, we have \( \Pr[W_5] = 1/2 \) since the output distribution of the challenge oracle does not depend on \( d \in \{0,1\} \). In \( \text{Game}_4 \), if \( d = 1 \), each challenge query \( (j, X^*) \) obtains a response

\[ y^*_j = \pi\left(\left(\left(A(x^*)\right)^T \cdot s_j\right)_p\right) = \pi\left(\left(\left(A \cdot R_{x^*} + G\right)^T \cdot s_j\right)_p\right). \]

The same arguments as in [56, Appendix D] show that the right-hand-side member of (28) is statistically close to \( U(\mathbb{Z}_p^n) \). Indeed, \( z^*_j = \left(\left(A \cdot R_{x^*} + G\right)^T \cdot s_j\right)_p \) depends on a term \( G^T \cdot s_j \in \mathbb{Z}_q^m \), which is an injective function of \( s_j \). More precisely, if \( BAD \) does not occur and \( SK_j = s_j \) has not been corrupted, \( \mathcal{A} \) obtains at most \( n' \cdot \log q \) bits of information about \( s_j \).

We also note that \( z^*_j \) can be written

\[ z^*_j = \left(\left(A \cdot R_{x^*} + G\right)^T \cdot s_j\right)_p = \left[R_{x^*}^T \cdot A^T \cdot s_j\right]_p + \left[G^T \cdot s_j\right]_p + e_{s,j}, \quad \text{(29)} \]

for some \( e_{s,j} \in \{0,1\}^m \). Moreover, the proof of the claim implies that

\[ \left[R_{x^*}^T \cdot A^T \cdot s_j\right]_p = \left[R_{x^*}^T \cdot \tilde{A}^T \cdot C \cdot s_j\right]_p \]
since it considers a union bound to show that equality (25) holds for all $x \in \{0,1\}^L$ and $j \in [N]$ with overwhelming probability. This implies $H_\infty (| R_x^\top \cdot A^\top \cdot s_j |_p \mid C \cdot s_j ) = 0$ with high probability. In the right-hand-side member of (29), we also observe that $| G^\top \cdot s_j |_p + e_{s,j,x^*}$ is an injective function of $s_j$. Indeed, we have

$$| G^\top \cdot s_j |_p + e_{s,j,x^*} = (p/q) \cdot G^\top \cdot s_j - t_{s,j,x^*} + e_{s,j,x^*}$$

for some $e_{s,j,x^*} \in (0,1)^m$, so that

$$\left( \frac{q}{p} \right) \cdot (| G^\top \cdot s_j |_p + e_{s,j,x^*}) = G^\top \cdot s_j + e'_{s,j,x^*} \tag{30}$$

for some $e'_{s,j,x^*} \in (-q/p, 2 \cdot q/p)^m$.

When assessing the entropy of $z_j^*$ conditionally on $A$’s view, we thus obtain

$$H_\infty (z_j^* \mid C \cdot s_j) = H_\infty (| R_x^\top \cdot A^\top \cdot s_j |_p + | G^\top \cdot s_j |_p + e_{s,x^*} \mid C \cdot s)$$

$$= H_\infty (| G^\top \cdot s_j |_p + e_{s,x^*} \mid C \cdot s_j) \tag{31}$$

$$= H_\infty (s_j \mid C \cdot s_j) \geq n \cdot \log \sigma - n' \cdot \log q - 1.$$

Here, the second equality follows from the fact that, for any random variables $X, Y, Z$ defined over an additive group, we have $H_\infty (Y + Z \mid X) = H_\infty (Z \mid X)$ if $H_\infty (Y \mid X) = 0$. The last inequality follows from Lemma A.6: the distribution of $s_j \in \mathbb{Z}^n$ conditionally on $C \cdot s_j$ is $s_{j,0} + D_{A^\top(C),\sigma,-s_{j,0}}$, where $s_{j,0} \in \mathbb{Z}^n$ is an arbitrary solution of $C \cdot s_{j,0} = C \cdot s_j$. Then, Lemma A.6 implies that the point with highest probability in $D_{A^\top(C),\sigma,-s_{j,0}}$ occurs with probability $\leq 2 \det (A^\top(C)) / \sigma^n$. Since $\det (A^\top(C)) = q^n$, with probability $1 - 2^{O(\lambda)}$, we have $H_\infty (s_j \mid C \cdot s_j) \geq n \cdot \log \sigma - n' \cdot \log q - 1$.

To extract statistically uniform bits from each uncorrupted $z_j^*$, we need to take into account the fact that $x^*$ may depend on $pp$. Assuming that $P_K (X^*) = 0$, for each uncorrupted $j \in [N]$, the source $z_j^*$ is taken from a distribution determined by $X^*$ within a collection of less than $2^\ell$ distributions (i.e., those for which $P_K (X^*) = 0$), which all have min-entropy $n = n \cdot \log \sigma - n' \cdot \log q - 1$. Moreover, the $\{ z_j^* \}_{j \in [N]}$ are independent since they are obtained by applying the same function to $N$ independent variables. If we call $C \subset [N]$ the subset of corrupted indexes, by applying Lemma A.9 and Lemma A.8 with $\varepsilon = 2^{-\lambda}$ for a collection $X'$ of at most $M = 2^\ell$ distributions, we obtain that the joint distribution of $\{ \pi(z_j^*) \}_{j \in [N] \setminus C}$ is $(\ell \cdot 2^{-\lambda} + 2^{-\ell})$-close to the uniform distribution over $(\mathbb{Z}_p^L)^{|N| \setminus |C|}$.

\[ \square \]

B Relation Between Definitions 2.11 and 2.12

Proposition B.1. For any MCFE scheme, 1Ch-IND security implies IND security.
Proof. Let us consider an efficient MCFE adversary $A$ in the IND security game. We show that $A$ implies an MCFE adversary in the 1Ch-IND game.

Suppose that $A$ makes encryption queries for $Q$ distinct tags during the game. The proof uses a standard hybrid argument over the distinct tags that $A$ queries throughout the attack. Let $H_k$ with $k \in \{0, 1, \ldots, Q\}$ be the game in which the challenger replies to encryption queries $QEncrypt(i, x_0^i, x_1^i, t)$ in the following way.

- If the tag $t$ is one of the first $k$ distinct tags appearing in $A$‘s encryption queries, it replies with $Encrypt(e_k^i, x_0^i, t)$.
- Otherwise, it replies with $Encrypt(e_k^i, x_1^i, t)$.

Note that an IND adversary $A$ in the sense of Definition 2.11 implies a distinguisher between $H_0$ and $H_Q$. We claim that, for any $k \in \{0, \ldots, Q-1\}$, an efficient distinguisher $A_k$ between $H_k$ and $H_{k+1}$ implies the existence of an efficient adversary $B_k$ in 1Ch-IND game and such that $Adv^{1Ch-IND}(B_k) = Adv^{k,k+1}(A_k)$. This adversary $B_k$ proceeds as follows.

Initialization: Having received $mpk$, $B_k$ runs $A_k$ by feeding it with the same public parameters $mpk$.

Encryption queries: For each query $QEncrypt(i, x_0^i, x_1^i, t)$ sent by $A_k$, $B_k$ does the following.

- If $t$ is among the first $k$ distinct tags queried by $A_k$, $B_k$ sends the query $QEncrypt(i, x_0^i, t)$ to its own challenger in the 1Ch-IND game and relays the answer back to $A_k$.
- If $t$ coincides with $(k+1)$-th distinct tag queried by $A_k$, $B_k$ makes the challenge query $QCEncrypt(i, x_0^i, x_1^i, t)$ and transmits the response to $A_k$.
- Otherwise, $B_k$ sends the encryption query $QEncrypt(i, x_1^i, t)$ to its 1Ch-IND challenger and forwards the answer to $A_k$.

Functional decryption queries: For each query $QDKeygen(f)$ made by $A_k$, $B_k$ sends the same query to its own challenger and passes the answer to $A_k$.

Corruption queries: For each query $QCorrupt(i)$ made by $A_k$, $B_k$ makes the same query to its own challenger and forwards the answer to $A_k$.

Finalize: When $A_k$ halts, $B_k$ outputs the same bit $b'$ as $A_k$.

We can conclude that, for any efficient IND adversary $A$, there exists an efficient 1Ch-IND adversary $B$ such that

$$Adv^{IND}(A) \leq Q \cdot Adv^{1Ch-IND}(B).$$

$\Box$

Proposition B.2. 1Ch-IND security is equivalent to 1Ch-IND’ security.

Proof. Recall that 1Ch-IND’ refers to the same security notion as described in Definition 2.12, except that Condition 2 in the Finalize step is replaced by
2': Both $\text{QEncrypt}(i,x,t^*)$ and $\text{CQEncrypt}(i,x_i^0,x_i^1,t^*)$ have been made for an index $i$ and the challenge label $t^*$, such that $x_i^0 \neq x_i^1$.

The non-trivial implication to prove is that $1\text{Ch-IND}$ security implies $1\text{Ch-IND}'$ security. Towards a contradiction, suppose that there exists an adversary $A'$ that breaks the $1\text{Ch-IND}'$ security game. Our adversary $A$ answers both corruption and functional decryption key queries made by $A'$ by relaying them to its challenger and forwarding the responses to $A'$. For a query $\text{QEncrypt}(i,x,t)$ made by $A'$, $A$ checks if a challenge query $\text{CQEncrypt}(i,x_i^0,x_i^1,t)$ was made earlier. If no challenge query has been made so far for the pair $(i,t)$, the reduction sends the same $\text{QEncrypt}(i,x,t)$ query to its challenger and passes the the answer to $A'$. Otherwise, it makes a $\text{Corrupt}(i)$ query to its challenger and thereby obtains the encryption key $ek_i$. To answer any subsequent query for slot $i$, the reduction simply uses $ek_i$.

To answer a challenge query $\text{QEncrypt}(i,x_i^0,x_i^1,t^*)$, the reduction $A$ forwards the query to its challenger and transmits the answer to $A'$ if it did not previously make an encryption $\text{QEncrypt}(i,x,t^*)$ for the pair $(i,t^*)$. If $A$ previously made an encryption query for the pair $(i,t^*)$, it also invokes its challenger and sends it the query $\text{Corrupt}(i)$. Upon receiving $ek_i$, $A$ is able to answer any query concerning the $i$-th slot.

Notice that the adversary $A'$ is only allowed to make both encryption and challenge queries on the same pair $(i,t^*)$ if $x_i^0 = x_i^1$. This implies that the reduction is allowed to make the query $\text{Corrupt}(i)$ to its challenger, which allows it to answer all queries concerning the $i$-th slot. $\square$

C Deferred Proofs for the Scheme in Section 3

C.1 Proof of Lemma 3.1 (Correctness)

Proof. Suppose that $C_{t,i} = G_i^0 \cdot x_i + A(\tau)^T \cdot s_i + e_i \in \mathbb{Z}_q^m$, with each error vector $e_i \leftarrow D_{\mathbb{Z}^m,\alpha q}$ and $\tau = \text{AHF}(t)$. Defining $E = [e_1|\cdots|e_\ell] \in \mathbb{Z}^{m\times \ell}$, we have

$$f_{t,y} = \sum_{i=1}^\ell y_i \cdot C_{t,i} - A(\tau)^T \cdot s_y = G_0^T \cdot X \cdot y + E \cdot y \mod q.$$  

We know that $\Lambda^+(G_0)$ has a public trapdoor $T_0 \in \mathbb{Z}^{m\times m}$ such that $G_0 \cdot T_0 = 0 \mod q$ and $\|T_0\|_\infty \leq \log q$ where $\|T_0\|_\infty := \sup_{x \neq 0} \frac{\|T_0^T \cdot x\|_\infty}{\|x\|_\infty}$. By using this short trapdoor, we are able to recover $X \cdot y \in \mathbb{Z}^{m\times \ell}$ in the following way: observe that $T_0^T \cdot f_{t,y} = T_0^T \cdot E \cdot y \mod q$. By [59, Lemma 4.4], we can bound the Euclidean norm of a Gaussian vector $e \leftarrow D_{\mathbb{Z}^m,\alpha q}$ by $\|e\| \leq \sqrt{\ell} \cdot \alpha q$ with probability exponentially close to 1. It comes that

$$\|E\|_\infty = \max_{i \in [m]} \sum_{j=1}^\ell |e_{i,j}| \leq \sqrt{\ell} \cdot \max_{i \in [m]} \sqrt{\sum_{j=1}^\ell e_{i,j}^2} \leq \ell \cdot \alpha q.$$
Moreover, the hypotheses imply
\[ \|T_0^T \cdot E \cdot y\|_\infty \leq \|T_0^T\|_\infty \cdot \|E\|_\infty \cdot \|y\|_\infty \leq \log q \cdot \ell \cdot \alpha q \cdot Y < q/2. \]

Hence, \( T_0^T \cdot f_i \cdot y \mod q \) actually reveals \( T_0^T \cdot E \cdot y \) over \( \mathbb{Z}^m \). Since \( T_0^T \) is a \( \mathbb{Z} \)-basis, it comes that we can recover \( E \cdot y \in \mathbb{Z}^m \), at which point we also get \( X \cdot y \mod q \).

Since \( \|X \cdot y\|_\infty \leq \ell \cdot X \cdot Y < q/2 \) by hypothesis, the modular product \( X \cdot y \mod q \) is nothing but \( X \cdot y \in \mathbb{Z}^m \).

\[ \Box \]

### C.2 Proof of Lemma 3.3

**Proof.** If the matrices \( A_{i,b} = A \cdot R_{i,b} + \mu_{i,b} \cdot G \in \mathbb{Z}^{n \times m} \), for \( i \in [L] \) and \( b \in \{0, 1\} \), with \( R_{i,b} \leftarrow \{-1, 1\}^{m \times m} \) and \( \mu_{i,b} \in \{0, 1\} \), defined as in equation (20), then \( R_\tau = R_\tau' \cdot G^{-1}(W^T) \), where \( R_\tau' \) is given below:

\[
R_\tau' = R_{L,\tau[L]} \cdot G^{-1} \left( A_{L-1,\tau[L-1]} \cdot G^{-1} \left( A_{L-2,\tau[L-2]} \cdot (\cdots G^{-1} \left( A_{1,\tau[1]} \right) \right) \right) \\
+ \mu_{L,\tau[L]} \cdot R_{L-1,\tau[L-1]} \cdot G^{-1} \left( A_{L-2,\tau[L-2]} \cdot (\cdots G^{-1} \left( A_{1,\tau[1]} \right) \right) \\
+ \mu_{L,\tau[L]} \cdot \mu_{L-1,\tau[L-1]} \cdot R_{L-2,\tau[L-2]} \cdot G^{-1} \left( A_{L-3,\tau[L-3]} \cdot (\cdots G^{-1} \left( A_{1,\tau[1]} \right) \right) \\
\vdots \\
+ \mu_{L,\tau[L]} \mu_{L-1,\tau[L-1]} \cdots \mu_{3,\tau[3]} \cdot R_{2,\tau[2]} \cdot G^{-1}(A_{1,\tau[1]}) \\
+ \mu_{L,\tau[L]} \mu_{L-1,\tau[L-1]} \cdots \mu_{2,\tau[2]} \cdot R_{1,\tau[1]}
\]

\( R_\tau' \) is thus a sum of \( L \) terms of products of at most 3 binary matrices. Since each binary matrix has infinity norm less than \( m \), we obtain the upper bound \( \|R_\tau'\|_\infty \leq L \cdot m^3 \). By [59, Lemma 4.4], we can bound the Euclidean norm of a Gaussian vector \( e \leftarrow D_{\mathbb{Z}^n, \alpha q} \) by \( \|e\| \leq \sqrt{n} \cdot \alpha_q \), with probability exponentially close to 1. We thus obtain

\[
\|E\|_\infty = \max_{i \in [m]} \sum_{j=1}^n |e_{i,j}| \leq \sqrt{n} \cdot \max_{i \in [m]} \sqrt{\sum_{j=1}^n e_{i,j}^2} \leq n \cdot \alpha_q,
\]

so that \( \|R_\tau^T \cdot E \cdot s\|_\infty \leq \|R_\tau^T\|_\infty \cdot \|E\|_\infty \cdot \|s\|_\infty \leq L m^3 \cdot n \alpha_q \cdot \sigma \sqrt{n} \). By Lemma 2.4, the statistical distance to be bounded is at most \( \leq m \cdot \frac{L m^3 \cdot n \alpha_q \cdot \sigma \sqrt{n}}{\alpha_q} = 2^{-\lambda}. \]

\[ \Box \]

### C.3 Proof of Lemma 3.5

**Proof.** By Remark 2.3, each column of \( T \) has norm smaller than

\[
\|t_i\| \leq O(\sqrt{n \cdot (n_0 + n_1) \log q}) \cdot \omega(\sqrt{\log n}) = \omega(n \sqrt{\log n}),
\]

so that \( \|T \cdot \Delta x_i\|_\infty \leq \|T \cdot \Delta x_i\|_\infty \leq 2X_{n_0} \cdot \omega(n \sqrt{\log n}) \). By Lemma 2.4, the considered distance is smaller than \( n \cdot \frac{2X_{n_0} \omega(n \sqrt{\log n})}{\sigma} \leq 2^{-\lambda} \cdot (4X)^{-n_0}. \]

\[ \Box \]
D Proof of Theorem 4.1

Proof. The proof considers a sequence of games. The first game corresponds to the real sta-IND-sec game where the challenger’s bit is \( b = 0 \). The final game is the sta-IND-sec game where \( b = 1 \). We will prove that they are computationally indistinguishable assuming that the scheme of Section 3 provides IND-sec security in the sense of Definition 2.11. For each \( i \), we denote by \( W_i \) the event that the adversary outputs \( \beta = 1 \) in \( \text{Game}_i \).

**Game_0:** This is the real security sta-IND-sec security game, which corresponds to the game of Definition 2.14 with the sets \( CS \) and \( HS \) chosen before the initialization phase. Letting \( \ell_h = |HS| \) and \( |CS| = \ell - \ell_h \), we assume w.l.o.g. that \( A \) chooses to corrupt the last \( \ell - \ell_h \) senders: i.e., \( HS = \{1, \ldots, \ell_h\} \) and \( CS = \{\ell_h + 1, \ldots, \ell\} \). By definition, the adversary outputs \( \beta = 1 \) with probability \( \Pr[W_0] \).

**Game_1:** In this game, we modify the \( \text{QDKeygen}(\cdot, \cdot) \) oracle, which generates partial functional decryption keys. At each partial functional decryption key query \( \text{QDKeygen}(i, f) \) for an index \( i \in HS \) and a function \( f \) described by a vector \( y = (y_1, \ldots, y_\ell) \in [-Y, Y]^\ell \), the challenger \( C \) responds as follows:

- If the pair \( (i, f) \) is such that not all indexes \( j \in HS \setminus \{i\} \) have been the input of a query \( \text{QDKeygen}(j, f) \) for the same function \( f \), \( C \) samples a Gaussian vector \( s'_i \leftarrow D_{Z^n, \sigma} \) and encrypts \( y_i \cdot s'_i \in Z_n \) as
  \[
  dk_{f,i} = G^\top \cdot (y_i \cdot s'_i) + B(\tau_f)^\top \cdot t_i + e_{f,i} \in \bar{Z}_{\bar{m}},
  \]

  where \( e_{f,i} \leftarrow D_{\bar{Z}_{\bar{m}}, \alpha q} \).

- If the pair \( (i, f) \) is such that similar queries \( (j, f) \) have been made earlier for the same function \( f \) and for all indexes \( j \in HS \setminus \{i\} \), \( C \) recalls the vectors \( \{s'_j\}_{j \in HS \setminus \{i\}} \) that were chosen to answer earlier queries. It computes \( s_y = \sum_{i=1}^{\ell} s_i \cdot y_i \in Z_n \) as in the centralized scheme of Section 3. Then, it defines
  \[
  s'_{f,i} = s_y - \sum_{j \in HS \setminus \{i\}} y_j \cdot s'_j - \sum_{i=\ell_h+1}^{\ell} s_i \cdot y_i \quad \text{over } Z,
  \]

  The challenger \( C \) then responds the query \( \text{QDKeygen}(i, f) \) by encrypting \( s'_{f,i} \in Z^n \) and returning
  \[
  dk_{f,i} = G^\top \cdot s'_{f,i} + B(\tau_f)^\top \cdot t_i + e_{f,i} \in \bar{Z}_{\bar{m}},
  \]

  where \( e_{f,i} \leftarrow D_{\bar{Z}_{\bar{m}}, \alpha q} \).

We note that the vectors \( s'_j \) sampled at each query do not have to be consistent across partial key generation queries for distinct functions (they just have to be consistent across queries \( \text{QDKeygen}(\cdot, f) \) involving the same function \( f \)). This will not be necessary since we are just seeking to simulate the
partial key generation oracle $\text{QDKeygen}(\cdot, \cdot)$ in a way that is computationally indistinguishable from the previous game. We also note that the vector $s_{f,i}^t \in \mathbb{Z}^n$ encrypted in (33) may not even be in the lattice $y_t \cdot \mathbb{Z}^n$ (in contrast with a vector encrypted by the real $\text{DKeygenShare}$ algorithm). However, this will not be a problem here. Indeed, our goal is solely to simulate the $\text{QDKeygen}(\cdot, \cdot)$ oracle in such a way that $\text{Game}_1$ remains computationally indistinguishable from $\text{Game}_0$ and, in the next transition from $\text{Game}_1$ to $\text{Game}_2$, allows reducing the security of the scheme in Section 3 to that of our DMCFE scheme.

Lemma D.1 shows that $\text{Game}_1$ is computationally indistinguishable from $\text{Game}_0$ as long as the centralized MCFE scheme of Section 3 is secure in the sense of Definition 2.11. We note that a weaker security definition than Definition 2.11 would be sufficient: as a centralized MCFE adversary, the reduction only needs to: (i) send a single functional secret key query for the vector $(1, 1, \ldots, 1)^\top \in \mathbb{Z}_\ell^n$ to its centralized MCFE challenger; (ii) make all its corruption queries at once after having received the master public key.

$\text{Game}_2$: In this game, we modify the encryption oracle. At each encryption query $\text{QEncrypt}(i, x_{0,i}, x_{1,i}, t)$, the challenger $\mathcal{CH}$ encrypts $x_{1,i} \in [-X, X]^n$ instead of $x_{0,i} \in [-X, X]^n$ (we assume that either $x_{0,i} = x_{1,i}$ or $i \in \mathcal{HS}$ since, otherwise, $\mathcal{A}$’s output is eventually replaced by a random bit). Namely, it sample $e_i \leftarrow D_{\mathbb{Z}_m^{m,\alpha q}}$ and computes

$$C_{t,i} = G_0^\top \cdot x_{1,i} + A(\tau)^\top \cdot s_i + e_i \in \mathbb{Z}_q^m,$$

where

$$A(\tau) = A_{L,\tau[L]} \cdot G^{-1} \left( A_{L-1,\tau[L-1]} \cdot G^{-1} \left( \cdots A_{2,\tau[2]} \cdot G^{-1} (A_{1,\tau[1]}) \right) \right) \cdot G^{-1}(W^\top) \in \mathbb{Z}_q^{n \times m},$$

with $\tau = \text{AHF}(t) \in \{0, 1 \}^L$.

Lemma D.2 relies on the IND-sec security of the scheme in Section 3 to show that $\text{Game}_2$ is computationally indistinguishable from $\text{Game}_2$ under the LWE$_{q,m,n,\alpha_1}$ assumption.

$\text{Game}_3$: In this game, we modify again the $\text{QDKeygen}(\cdot, \cdot)$ oracle and restore it to its original behavior, which is that of the real scheme. This game transition is exactly identical to the one from $\text{Game}_0$ to $\text{Game}_1$, but in the converse direction. Lemma D.3 states that, as long as the MCFE scheme of Section 3 provides IND-sec security in the sense of Definition 2.11, $\text{Game}_3$ and $\text{Game}_2$ are computationally indistinguishable.

In $\text{Game}_3$, it is easy to see that $C$ and $A$ are playing the actual sta-IND-sec game where the challenge bit of $C$ is $b = 1$.

Putting the above altogether, we find that $\text{Game}_0$ and $\text{Game}_3$ are computationally indistinguishable so long as the centralized MCFE scheme of Section 3 provides security in the sense of Definition 2.11. \qed
Lemma D.1. \textbf{Game}_0 \text{ and Game}_1 \text{ are computationally indistinguishable assuming that the MC\,FE scheme of Section 3 provides IND-\,sec security. Under the LWE}_q\,\,\tilde{\alpha},\,\tilde{\alpha},\,\tilde{\alpha}_1 \text{ assumption, we have } |\Pr[\text{Game}_1] - \Pr[\text{Game}_0]| \in \text{negl}(\lambda)|.\]

\textbf{Proof.} Let us assume that an adversary } A \text{ can distinguish Game}_1 \text{ from Game}_0 \text{ with noticeable advantage } \varepsilon. \text{ We construct a IND-\,sec adversary } B \text{ against the centralized MC\,FE scheme with the same advantage } \varepsilon. \text{ Our MC\,FE adversary } B \text{ obtains from its IND-\,sec challenger a master public key}

\[ \text{mpk} := \left( \text{cp}, \ V, \ \{ B_{i,0}, B_{i,1} \in \mathbb{Z}_q^{\tilde{n} \times \tilde{m}} \}_{i=1}^L \right), \]

where the common public parameters

\[ \text{cp} := (\lambda, \ \ell_{\max}, \ 2\ell \cdot \beta \cdot Y, \ 1, \ n, \ \tilde{n}_1, \ \tilde{n}, \ m, \ \tilde{m}, \ \alpha, \ \tilde{\alpha}, \ \ell_t, \ L, \ \tilde{q}, \ \text{AHF}_f) \]

specify the message space } \mathcal{M} = [-2\ell\beta Y, 2\ell\beta Y]^n \text{ and the functional secret key space } [-1, 1]^\ell. \text{ Then, } B \text{ first sends its challenger a unique functional secret key query for the vector } (1, 1, \ldots, 1)^\top \text{ and obtains in return } t = \sum_{i=1}^\ell t_i \in \mathbb{Z}^\tilde{n}. \text{ Next, } B \text{ chooses random matrices } A_{i,b} \leftarrow U(\mathbb{Z}_q^{\tilde{n} \times m}) \text{, for each } i \in [L], \ b \in \{0, 1\}. \text{ It also chooses random matrices } V \leftarrow U(\mathbb{Z}_q^{\tilde{m} \times \tilde{n}}). \text{ For each } i \in [\ell], \text{ it chooses users’ secret keys } s_i \leftarrow D_{\tilde{n}, \tilde{\alpha}} \text{ in the second MC\,FE instance. This allows } B \text{ to create a master public key}

\[ \text{mpk} := \left( \text{cp}, \ V, \ \tilde{V}, \ \{ A_{i,0}, A_{i,1} \in \mathbb{Z}_q^{\tilde{n} \times m} \}_{i=1}^L, \ \{ B_{i,0}, B_{i,1} \in \mathbb{Z}_q^{\tilde{n} \times \tilde{m}} \}_{i=1}^L, \ t \right), \]

where

\[ \text{cp} := (\lambda, \ \ell_{\max}, \ X, \ Y, \ n_0, \ n_1, \ n, \ m, \ \tilde{n}, \ \tilde{m}, \ \alpha, \ \tilde{\alpha}, \ \ell_t, \ \ell_f, \ L, \ q, \ \tilde{q}, \ \text{AHF}_t, \ \text{AHF}_f) \],

is augmented with a description of a second balanced admissible hash function family } \text{AHF}_f : \{0, 1\}^{\ell_f} \rightarrow \{0, 1\}^L. \text{ Before handing } \text{mpk} \text{ to } A, \text{ our centralized MC\,FE adversary } B \text{ waits until } A \text{ chooses the set } \mathcal{CS} \subseteq [\ell] \text{ of players it wants to corrupt. We assume w.l.o.g. that } A \text{ chooses } \mathcal{CS} = \{ \ell_h + 1, \ldots, \ell \}. \text{ At this point, } B \text{ sends corruption queries Q\,Corrupt}(i) \text{ to its own challenger for all } i \in \mathcal{CS}. \text{ The MC\,FE challenger replies by sending } \{ t_i \}_{i \in \mathcal{CS}}, \text{ at which point } B \text{ gives } \{ \text{ske} \leftarrow (s_i, t_i) \}_{i \in \mathcal{CS}} \text{ and mpk to } A. \text{ At any time, } B \text{ can make a query Q\,Keygen}(i, f) \text{ for an index } i \in \mathcal{HS} \text{ and a function } f \text{ described by a vector } y = (y_1, \ldots, y_\ell)^\top \in \mathbb{Z}^\ell. \text{ At each such query, } B \text{ proceeds as follows:}

- If the pair } (i, f) \text{ is such that not all indexes } j \in \mathcal{HS} \setminus \{i\} \text{ have been the input of a query } (j, f) \text{ for the same function } f, \text{ } B \text{ samples } s'_{i} \leftarrow D_{\tilde{n}, \tilde{\alpha}} \text{ and defines } s'_{f,i} = y_i \cdot s'_{i} \in \mathbb{Z}^\tilde{n}. \]
If the pair \((i, f)\) is such that queries \(\mathcal{QDKeygen}(j, f)\) have been made before for the same function \(f\) on all indexes \(j \in \mathcal{H} \setminus \{i\}\), \(\mathcal{C}\) recalls the vectors \(\{s'_j\}_{j \in \mathcal{H} \setminus \{i\}}\) and \(\{s''_j\}_{j \in \mathcal{H} \setminus \{i\}}\) that were chosen to handle earlier queries for \(f\). It then defines \(s_y = \sum_{i=1}^{t} y_i \cdot s_i \in \mathbb{Z}^n\) which is the secret key that would be computed for \(y \in [-Y, Y]^f\) using the real master secret key \(\{s_i\}_{i=1}^t\) in the centralized scheme of Section 3. It finally defines

\[
s'_{f, i} = s_y - \sum_{j \in \mathcal{H} \setminus \{i\}} s'_{f, j} - \sum_{i=t_h+1}^{t} s_i \cdot y_i \quad \text{over } \mathbb{Z}, \quad (34)
\]

Then, \(\mathcal{B}\) sends the query \(\mathcal{QEncrypt}(i, y_i \cdot s_i, s'_{f, i}, t_f)\) to its encryption oracle in the \(\text{IND-sec}\) game. Depending on the value of its secret bit \(b \in \{0, 1\}\), the MCFE challenger replies with either

\[
dk_{f, i} = \mathbf{G}^T \cdot (y_i \cdot s_i) + \mathbf{B}(\tau_f)^T \cdot \mathbf{t}_i + \mathbf{e}_{f, i}
\]

or

\[
dk_{f, i} = \mathbf{G}^T \cdot s'_{f, i} + \mathbf{B}(\tau_f)^T \cdot \mathbf{t}_i + \mathbf{e}_{f, i}, \quad (35)
\]

where \(\mathbf{e}_{f, i} \leftarrow D_{\mathbb{Z}^{q \cdot n}}\) and the response \(dk_{f, i}\) is returned to \(\mathcal{A}\) as a partial functional secret key for the vector \(y\).

At each encryption query \((i, \mathbf{x}_{0, i}, \mathbf{x}_{1, i}, t)\) made by \(\mathcal{A}\), \(\mathcal{B}\) can compute

\[
\mathbf{C}_{t, i} = \mathbf{G}_0^T \cdot \mathbf{x}_{0,i} + \mathbf{A}(\tau)^T \cdot \mathbf{s}_i + \mathbf{e}_i \in \mathbb{Z}_q^m, \quad (36)
\]

itself by faithfully running the encryption algorithm as it knows \(\{s_i\}_{i=1}^t\).

When \(\mathcal{A}\) terminates, \(\mathcal{B}\) checks if \(\mathcal{A}\) made a query \(\mathcal{QDKeygen}(i, f)\) for a function \(f\) such that not all indexes \(j \in \mathcal{H} \setminus \{i\}\) were the input of a query \(\mathcal{QDKeygen}(j, f)\). If so, \(\mathcal{B}\) makes the missing queries for itself. Namely, for each such function \(f\), \(\mathcal{B}\) parses the corresponding vector as \(y = (y_1, \ldots, y_t) \in [-Y, Y]^f\) and defines \(Q_f \subset \mathcal{H} \setminus \{i\}\) to be the set of indexes \(j\) for which no query \(\mathcal{QDKeygen}(j, f)\) was made by \(\mathcal{A}\). For each \(j \in Q_f\), \(\mathcal{B}\) does the following.

- If \(j\) is not the last unqueried index for \(f\), \(\mathcal{B}\) samples a Gaussian \(s'_j \leftarrow D_{\mathbb{Z}^{n \cdot q}}\) and sends the query \(\mathcal{QEncrypt}(j, y_j \cdot s_j, s_j', t_f)\) to its MCFE challenger.
- If \(j\) is the last unqueried index for \(f\), \(\mathcal{B}\) defines

\[
s'_{f, j} = \sum_{i \in \mathcal{H} \setminus \{j\}} y_i \cdot s_i - \sum_{i \in \mathcal{H} \setminus \{j\}} y_i \cdot s'_i \quad (37)
\]

and makes the encryption query \(\mathcal{QEncrypt}(j, y_j \cdot s_j, s'_{f, j}, t_f)\). Since the only functional secret key query made by \(\mathcal{B}\) is for the vector \(\mathbf{1} = (1, 1, \ldots, 1)^T\), \(\mathcal{B}\) only needs to make sure that dummy encryption queries are made for pairs of vectors that sum to the same value over indexes in \(\mathcal{H} \setminus \{i\}\). This is ensured by defining the last term as \((37)\).
In both cases, $B$ just ignores the responses of its challenger as these dummy encryption queries are only made to make sure that $B$’s advantage will not be annihilated by Condition 2 in Definition 2.11.

Finally, $B$ halts and outputs whatever $A$ outputs. We observe that $B$ strictly complies with the rules of the IND-sec game at any time. In particular:

- It never makes a ${\sf QEncrypt}(i, \cdot, \cdot, \cdot)$ query or a corrupted index $i \in CS$.
- By construction, $B$’s encryption queries $Q_{\sf Encrypt}(i, y_i \cdot s_i, s_{f,i}^t, t_f)$ are all made for message pairs $(y_1 \cdot s_1, \ldots, y_\ell \cdot s_\ell)$ and $(s_{f,1}^t, \ldots, s_{f,\ell_h}^t, y_{\ell_h+1} \cdot s_{\ell_h+1}, \ldots, y_\ell \cdot s_\ell)$ such that

$$
\sum_{i=1}^{\ell} y_i \cdot s_i = \sum_{i=1}^{\ell_h} s_{f,i}^t + \sum_{i=\ell_h+1}^{\ell} y_i \cdot s_i \quad \text{ (over $\mathbb{Z}$)}
$$

The lemma’s claim follows from the fact that, if the secret bit of $B$’s MCFE challenger is $b = 0$, $A$’s view is exactly the same as in $\text{Game}_0$. If $b = 1$, the distribution of partial functional secret keys is given by (35), which corresponds to $\text{Game}_1$. \\
\hfill $\square$

**Lemma D.2.** $\text{Game}_2$ and $\text{Game}_1$ are computationally indistinguishable assuming that the MCFE scheme of Section 3 provides security in the IND-sec sense. Under the LWE_{q,m,n_1,\alpha_1} assumption, we have $|\Pr[W_2] - \Pr[W_1]| \in \text{neg}(\lambda)$.

**Proof.** Let $A$ be an adversary that can distinguish between $\text{Game}_1$ and $\text{Game}_2$ with noticeable advantage $\varepsilon$. We build an adversary $B$ against the centralized scheme that wins the IND-sec game (cf. Definition 2.11) with the same advantage.

The reduction $B$ begins by obtaining from its challenger a master public key

$$
\bar{\text{mpk}} := \left( \bar{cp}, V, \{ \bar{A}_{i,0}, \bar{A}_{i,1} \in \mathbb{Z}_q^{n \times m} \}_{i=1}^L \right),
$$

where the common public parameters

$$
\bar{cp} := \left( \lambda, \ell_{\text{max}}, X, Y, n_0, n_1, n, m, \alpha, \sigma, \ell_t, L, q, \text{AHF}_f \right).
$$

The adversary $A$ declares the set $CS$ of corrupted senders, as mandated by the static security game. Then, $B$ makes the relevant queries $Q_{\sf Corrupt}(i)$ for each $i \in CS$ to its challenger and the latter replies with $\{ s_i \in \mathbb{Z}^n \}_{i \in CS}$.

Next, $B$ chooses $V \leftarrow U(\mathbb{Z}_q^{n \times n})$ and $B_{i,b} \leftarrow U(\mathbb{Z}_q^{n \times m})$ for each $i \in [L]$ and $b \in \{0,1\}$. It also samples $t_i \leftarrow D_{\mathbb{Z}_{\lambda\beta}}$ for $i \in [\ell]$ and computes $t = \sum_{i=1}^{\ell} t_i$. This allows $B$ to answer $A$’s static corruption queries by revealing $(s_i, t_i) \in \mathbb{Z}^n \times \mathbb{Z}^\lambda$ for all $i \in CS$. In addition, $B$ can feed $A$ with the public parameters for the decentralized scheme, which consist of $\tilde{\text{mpk}} := (\text{mpk}, \tilde{\text{mpk}}, t)$, where

$$
\tilde{cp} := \left( \lambda, \ell_{\text{max}}, \tilde{X}, \tilde{Y}, n, \tilde{n}_1, \tilde{n}, \tilde{m}, \tilde{\alpha}, \tilde{\sigma}, \ell_f, L, \tilde{q}, \text{AHF}_f \right),
$$

where $\tilde{X} = 2\ell/\beta Y$, and

$$
\tilde{\text{mpk}} := \left( \tilde{cp}, \tilde{V}, \{ \tilde{B}_{i,0}, \tilde{B}_{i,1} \in \mathbb{Z}_q^{\tilde{n} \times \tilde{m}} \}_{i=1}^L \right)
$$

52
For each encryption query $\text{QEncrypt}(i, x_{0,i}, x_{1,i}, t)$ made by $A$ for some honest sender $i \in HS$, $B$ passes the same encryption query to its challenger and relays the response back to $A$. This response is of the form

$$C_{t,i} := G_0^T \cdot x_{b,i} + A(\tau)^T \cdot s_i + e_i,$$

(38)

where $e_i \leftarrow \mathcal{D}_{\mathbb{Z}^{m \cdot \alpha q}}$ and $b \in \{0, 1\}$ is the MCFE challenger’s random bit.

When $A$ makes an functional decryption query $\text{QDKeygen}(i, f_y)$, $B$ parses the corresponding vector as $y = (y_1, \ldots, y_\ell) \in \mathbb{Z}^\ell$ and does the following:

- If there exists $j \in HS \setminus \{i\}$ such that $\text{QDKeygen}(j, f_y)$ has not been asked yet, then $B$ samples $s'_j \leftarrow \mathcal{D}_{\mathbb{Z}^{m \cdot \alpha q}}$ and returns

$$dk_{f,i} := G^T \cdot (y_i \cdot s'_j) + B(\tau_f)^T \cdot t_i + e_{f,i}$$

with $e_{f,i} \leftarrow \mathcal{D}_{\mathbb{Z}^{m \cdot \alpha q}}$.

- If $\text{QDKeygen}(j, f_y)$ queries have been made for all $j \in HS \setminus \{i\}$, $B$ makes the functional decryption query $\text{QDKeygen}(f_y)$ to its centralized MCFE challenger. The challenger replies with $s_y \in \mathbb{Z}^n$. Using the $s'_j \in \mathbb{Z}^n$ that were previously chosen for the function $f$, $B$ computes

$$s'_{f,i} = s_y - \sum_{j \in HS \setminus \{i\}} y_j \cdot s'_j - \sum_{i=\ell_h+1}^{\ell} y_i \cdot s_i \in \mathbb{Z}^n,$$

and returns

$$dk_{f,i} := G^T \cdot s'_{f,i} + B(\tau_f)^T \cdot t_i + e_{f,i}$$

with $e_{f,i} \leftarrow \mathcal{D}_{\mathbb{Z}^{m \cdot \alpha q}}$.

When $A$ halts, $B$ outputs whatever $A$ outputs. We observe that we have $y_j \cdot s'_j \in [-\bar{X}, \bar{X}]^n$, $\bar{X} = 2\bar{Y}\beta$, for all $j \in HS$. We also observe that, if $A$ makes all encryption queries $\text{QEncrypt}(j, x_{0,j}, x_{1,j}, t)$ for all indexes $j \in HS$ for a given tag $t$, so does $B$ for the same tag $t$.

By construction, $B$ only makes a functional decryption query $\text{QDKeygen}(f_y)$ for a function $f_y$ when $A$ makes a partial functional key query $\text{QDKeygen}(i, f_y)$ for the last honest sender’s index $j \in HS$. In the Finalize step of Definition 2.14, Condition 3 requires that, for any function $f_y$ involved in queries $\text{QDKeygen}(i, f_y)$ for all honest senders’ indexes $i \in HS$, the condition $f_y(X_0) = f_y(X_1)$ be satisfied for any pair of messages $X_0 = [x_{0,1}, \ldots, x_{0,\ell}]$ and $X_1 = [x_{1,1}, \ldots, x_{1,\ell}]$ satisfying the two sub-conditions of Condition 3. This ensures that $B$’s functional key queries never break the rules imposed by Condition 3 of the Finalize step of the game described by Definition 2.11.

From (38), it follows that, if secret bit of $B$’s challenger is $b = 0$, $B$ is playing $\text{Game}_1$ with $A$. Similarly, they are playing $\text{Game}_2$ if $b = 1$.

**Lemma D.3.** $\text{Game}_1$ and $\text{Game}_2$ are computationally indistinguishable as long as the MCFE scheme of Section 3 provides IND-sec security. Under the $L\text{WE}_{q,m,n,\alpha_1}$ assumption, we have $|\Pr[W_3] - \Pr[W_2]| \in \text{negl}(\lambda)$. 

---

53
Proof. The proof is almost identical to that of Lemma D.1, the only difference being that, at each encryption query \((i, x_{0,i}, x_{1,i}, t)\) made by \(A\), the reduction \(B\) encrypts \(x_{1,i}\) (instead of \(x_{0,i}\) in (36)). The details are omitted. \(\Box\)