Decentralized Multi-Client Attribute Based Functional Encryption

Yuechen Chen, Linru Zhang and Siu-Ming Yiu
Department of Computer Science, The University of Hong Kong, Pokfulam, HKSAR, China

Keywords: Functional Encryption, Attribute Based Encryption, Inner Product, Duel System Encryption, Decentralized.

Abstract: Functional encryption (FE) allows users to learn only functional values from the encrypted data. However, in existing FE schemes, all legitimate users get the same decryption results. Functional encryption that allows users to get different decryption results based on user attributes/policies has many useful practical applications. For example, a company may only authorize department heads of other sections to query the average sale figures of the sales department from the encrypted sales database.
In this paper, we combine techniques from Attribute Based Encryption(ABE) and Multi Client Function Encryption (MCFE) to propose a primitive that we call "Multi-Client Attribute Based Functional Encryption scheme (MCABFE)" which allows only authorized users to obtain functional values based on the users' attributes/policies, and give the first MCABFE scheme for inner product functionality from simple and well-studied assumptions. Then we extend our new primitive and propose a "Decentralized Multi-Client Attribute Based Functional Encryption scheme (DMCABFE)" in which no trusted party is required in the setup phase and the generation of functional decryption keys, and also give an instantiation for inner product functionality.

1 INTRODUCTION

Functional encryption (FE) (O'Neill, 2010; Boneh et al., 2011; Sahai and Waters, 2005) is a paradigm for encryption which extends the traditional "all-or-nothing" requirement of public-key encryption in a much more flexible way. FE allows users to learn specific functional values of the encrypted data: for any function f from a class F, a functional decryption key dk_f can be computed such that, given any ciphertext CT with underlying plaintext x, using dk_f , a user can efficiently compute $f(x)$, but does not get any additional information about x. This is the most general form of encryption as it encompasses identity-based encryption, attribute-based encryption, broadcast encryption, and predicate encryption.

In some cases, some users want to integrate their data together, and allow authorized user to obtain a functional value of the encrypted integrated data. For example, in a company, employees may not want to disclose their salaries (or sale figures) to even some senior staff, e.g. department heads of other sections. Each employee can store his/her salary (or sale figure) in an encrypted form and further require that some department heads are only authorized to query certain statistics (e.g. the average/highest salary or sale figure) of the employees in another team, but not the exact figure of individual employee. To solve this problem in a practical way, we need an encryption scheme that allows different users (clients) generate encrypted data as partial ciphertext, and can produce decryption keys which only allow authorized users to obtain a functional value of the integrated ciphertext.

1.1 Related Works

At first glance, functional encryption (Ananth et al., 2015; Badrinarayanan et al., 2016; Goldwasser et al., 2013b; Goldwasser et al., 2013a; Gorbunov et al., 2012; Sahai and Seyalioglu, 2010; Waters, 2015) and multi-client functional encryption(MCFE) (Goldwasser et al., 2014; Brakerski et al., 2018) seem to be able to solve our problem as it can provide a decryption key that allows a user to obtain only a functional value of the encrypted data. More precisely, the function can be modeled as a Turing Machine $F(\cdot,\cdot)$. The user who has a secret key sk_k can compute the function $F(k,x)$ on an encryption of x. However, FE schemes that can solve our problem (i.e., the ones that can consider the users' properties in its decryption, e.g. (Boyle et al., 2014; Garg et al., 2016; Bernstein et al., 2013; Waters, 2015; Ananth and Sahai, 2016)) are based on new, but not well-studied assumptions (such as indistinguishable obfuscation or multilinear maps). Attacks were identified for some constructions on indistinguishable obfuscation and multilin-

Chen, Y., Zhang, L. and Yiu, S.
Decentralized Multi-Client Attribute Based Functional Encryption.
DOI: 10.5220/0009824701180129
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 118-129
ISBN: 978-989-758-446-6

ear maps (Apon et al., 2017; Chen et al., 2017; Cheon et al., 2015; Coron et al., 2016). Hence, it is not clear whether these FE schemes are secure.

Recently, a line of work called functional encryption for inner product (IPFE) started by Abdalla et al. (Abdalla et al., 2015) aims at building functional encryption constructions based on standard assumptions such as the decisional Diffie-Hellman assumption. More precisely, a user could store an encrypted vector $\vec{y}$ on an untrusted remote server. The authority can generate a series of secret keys $\{sk_i\}$ corresponding to different vectors $\{\vec{x}_i\}$. These keys can be sent to the server for decrypting the message to get inner product $\langle \vec{x}_i, \vec{y} \rangle$, while ensuring no more leakage of information than the computation result. Abdalla et al. (Abdalla et al., 2015) first proposed a framework to construct IND-secure IPFE scheme with selective security . Continued by (Agrawal et al., 2016), the security of IPFE are improved to fully secure, also from standard assumptions. A generic construction of IPFE is given in (Abdalla et al., 2016), as well as three instantiations from Decisional Diffie-Hellman assumption (DDH), Decisional Composite Residuosity assumption (DCR) and Learning with errors assumption (LWE), respectively. (Baltico et al., 2017) realizes Functional Encryption for quadratic functions, with linear size of ciphertext. In (Chotard et al., 2018), multi-client functional encryption for inner product (MCIPFE) is proposed, in which the single input $\vec{x}$ to the encryption procedure is broken down into an input vector $(x_1, \ldots, x_n)$ where the components are independent. An index i for each client and a (typically time-based) label ℓ are used for every encryption: $(CT_1 = Encrypt(1, x_1, \ell), \ldots, c_n = Encrypt(n, x_n, \ell))$. Anyone having a functional decryption key dk_f , for an n-ary function f and multiple ciphertexts for the same label ℓ, $CT_1 = Encrypt(1, x_1, \ell), \ldots, CT_n = Encrypt(n, x_n, \ell)$, can compute $f(x_1, \ldots, x_n)$ but nothing else about the individual x_i's. They also proposed a decentralized multi-client functional encryption for inner product (DMCIPFE), in which the setup phase and the generation of functional decryption keys are decentralized among the same clients as the ones that generate the ciphertexts. However, these FE schemes reveals the decryption result to all users, thus all users get the same decryption result, which is not enough for allowing only authorized users to obtain functional values based on the users' attributes/policies.

While Attribute-Based Encryption (ABE) offers fine-grained decryption policy such that users can do decryption if their attributes/policies satisfy the access requirement on ciphertexts, users with different attributes/policies will get different decryption values based on their attributes/policies. ABE is first introduced in(Sahai and Waters, 2005). It can be divided into key-policy ABE (KP-ABE) and ciphertext-policy ABE (CP-ABE). In KP-ABE (Sahai and Waters, 2005; Goyal et al., 2006; Ostrovsky et al., 2007; Okamoto and Takashima, 2010), secret key is associated with access policy P, saying that users with what attributes can decrypt the data. And ciphertext is associated with user's attribute set S. The secret key can decrypt the ciphertext if the attribute set S satisfies the policy P. In CP-ABE (Bethencourt et al., 2007; Cheung and Newport, 2007; Waters, 2011; Yamada et al., 2011), ciphertext is associated with policy P, while secret key is associated with user's attribute set S. In (Chase, 2007), the first multi authority attribute based encryption is proposed, which allows the data sender to specify for each authority i a set of attributes monitored by that authority and a number d_i so that the message can be decrypted only by a user who has at least d_i of the given attributes from every authority. The property of ABE inspires us to use ABE to solve our problem in FE.

1.2 Our Contributions

In this paper, we mainly have two contributions. First, we propose a multi-client attribute based IPFE, which allows only authorized users to obtain functional values based on the users' attributes/policies. Then we give a decentralized multi-client attribute based IPFE, which removes the requirement of a trusted authority in the Setup phase.

In order to realize multi-client attribute based functional encryption for inner product, inspired by (Chase, 2007), we use a technique which is similar to KP-ABE to solve our problem. In our scheme, each user is related with a policy. Each data sender can set a data set in its partial ciphertext which restricts the access of the integrated ciphertext. The integrated ciphertext is composed of all the partial ciphertexts. On the other hand, each data sender owns a decryption key which corresponds to the user's own policy, and can decrypt the functional value from the integrated ciphertext, if its policy matches the union of all the required attribute sets.

In the decentralized multi-client attribute based functional encryption for inner product, we remove the trusted authority in multi-client attribute based IPFE, and the clients work together to generate appropriate functional decryption keys. The setup phase and the generation of functional decryption keys are decentralized among the same clients as the ones that generate the ciphertexts.

2 BACKGROUND

2.1 Bilinear Maps

We review some facts related to groups with efficiently computable bilinear maps in (Waters, 2011) and then present the assumptions used in the paper. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$ and e be a bilinear map, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. The bilinear map e has the following properties:

1. Bilinearity: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.

2. Non-degeneracy: $e(g, g) \neq 1$.

We say that $\mathbb{G}$ is a bilinear group if the group operation in $\mathbb{G}$ and the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ are both efficiently computable. Notice that the map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

2.2 Computational Assumptions

Definition 2.1. Decisional Diffie-Hellman Assumption(DDH)

The Decisional Diffie-Hellman Assumption states that, in a prime-order group $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can distinguish between the two distributions $\{(g^a, g^r, g^{ar}) \mid a, r \xleftarrow{\$} \mathbb{Z}_p\}$ and $\{(g^a, g^r, g^s) \mid a, r, s \xleftarrow{\$} \mathbb{Z}_p\}$ with non-negligible advantage.

Definition 2.2. Symmetric eXternal Diffie-Hellman Assumption(SXDH).

The Symmetric eXternal Diffie-Hellman (SXDH) Assumption states that, in a pairing group $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$ the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.3 Linear Secret Sharing Schemes

We review the definition of linear secret sharing scheme (LSSS) in (Waters, 2011) as follows.

Definition 2.3. Linear Secret-Sharing Schemes (LSSS) *A secret-sharing scheme over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if*

1. *The shares for each party form a vector over $\mathbb{Z}_p$.*
2. *There exists a matrix M with ℓ rows and n columns called the share-generating matrix for Π. For all $i = 1, \ldots, \ell$, the i'th row of M, we let the function ρ defining the party labelling row i as $\rho(i)$. When we consider the column vector $v = (s, r_2, \ldots, r_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared, and $r_2, \ldots, r_n \in \mathbb{Z}_p$ are randomly chosen, then Mv is the vector of ℓ shares of the secret s according to Π. The share $(Mv)_i$ belongs to party $\rho(i)$.*

2.4 Key-Policy Attribute Based Encryption

Here We review the definition of KP-ABE defined by Goyal in his paper (Goyal et al., 2006). We let U to represent the attribute universe description, S to represent the attributes set, and $\mathbb{A}$ to represent the access structure. Here in KP-ABE, $\mathbb{A}$ is the input of key generation algorithm while S is used in the encryption algorithm. We let function $f(S, \mathbb{A})$ gives out 1 if and only if $\mathbb{A}$ satisfies S.

Definition 2.4. A KP-ABE scheme is consisting of a series of algorithms, namely Setup, Encrypt, KeyGen and Decrypt, defined as below.

$(PK, MSK) \leftarrow \mathsf{Setup}(\lambda, U)$. The input of setup algorithm includes two parts, the description of the attribute universe denoted by U and the security parameter denoted by λ . The output of this algorithm is the public parameters PK and the master secret key MSK.

$CT \leftarrow \mathsf{Encrypt}(PK, m, S)$. In this algorithm, one will take an attribute set S as part of the input. The input also includes the public parameters, denoted by PK, and a message, denoted by m, to be encrypted. It gives out the ciphertext CT.

$SK \leftarrow \mathsf{KeyGen}(MSK, \mathbb{A})$. In the KeyGen algorithm, the input includes two parts, the master key MSK and an access structure $\mathbb{A}$. It computes and gives out the secret key SK corresponding to the access structure $\mathbb{A}$.

$m/\perp \leftarrow \mathsf{Decrypt}(SK, CT)$. The decrypt algorithm is the final step of this scheme. It will receive a secret key SK for $\mathbb{A}$ and a ciphertext CT which was originally encrypted under S as the input. This algorithm will do the decryption and gives out the message m if and only if $f(S, \mathbb{A}) = 1$. Otherwise, it will output an error symbol $\perp$.

3 DEFINITIONS

3.1 Multi-Client Attribute Based Functional Encryption

Definition 3.1. Multi Client Attribute Based Functional Encryption: A MCABFE on $\mathcal{M}$ over a set of n senders is defined by four algorithms:

– SetUp(1^λ): Takes as input the security parameter λ, and outputs the public parameters mpk, the master secret key msk and the n encryption keys ek_i;

– Encrypt(ek_i, x_i, ℓ, S_i): Takes as input a user encryption key ek_i, a value x_i to encrypt, a label ℓ, and an attribute set S_i, and outputs the ciphertext $CT_{\ell,i}$;

– DKeyGen(msk, f, P_i): Takes as input the master secret key msk, a function $f : \mathcal{M}^n \to \mathcal{R}$ and an access structure P_i, and outputs a functional decryption key dk_f, which corresponds to the access structure P_i ;

– Decrypt(dk_f, ℓ, $CT_{\ell,i}$): Takes as input a functional decryption key dk_f , a label ℓ, and an n-vector ciphertext $CT_{\ell,i}$, and outputs $f(x)$, if $CT_{\ell,i}$ is a valid encryption of $x = (x_i)_i \in M^n$ for the label ℓ and the attribute set S_i satisfies the access structure P_i, or $\perp$ otherwise.

We make the assumption that mpk is included in msk and in all the encryption keys ek_i as well as the functional decryption keys dk_f . The correctness property states that, given (mpk, msk,$(ek_i)_i$) $\leftarrow$ SetUp(λ), for any label ℓ, any function $f : \mathcal{M}^n \to \mathcal{R}$, and any vector $x = (x_i)_i \in \mathcal{M}^n$, if $CT_{\ell,i} \leftarrow$ Encrypt (ek_i, x_i, ℓ, S), for $i \in \{1,\ldots,n\}$, and $dk_f \leftarrow$ DKeyGen(msk, f, P), then Decrypt $(dk_f, \ell, CT_\ell = (CT_{\ell,i})_i) = f(x = (x_i)_i)$.

Definition 3.2. IND-Security Game for MCABFE. Let us consider an MCABFE scheme over a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$ with non-negligible probability:

– Initialization: the challenger $\mathcal{C}$ runs the setup algorithm (mpk, msk, $(ek_i)_i$) $\leftarrow$ SetUp(λ) and chooses a random bit $b \leftarrow \{0,1\}$. It provides mpk to the adversary $\mathcal{A}$;

– Encryption queries QEncrypt (i, x^0, x^1, ℓ, S_i): $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, and receives a set of ciphertexts C_{ℓ,i,S_i} generated by Encrypt (ek_i, x^b, ℓ, S_i). Noted that for the same pair (ℓ, i), further queries will later be ignored;

– Functional decryption key queries QDKeyGen(f, P_i): $\mathcal{A}$ has unlimited and adaptive access to the DKeyGen(msk, f, P_i) algorithm for any input function f and any access structure P_i of its choice . It is given back the functional decryption key dk_f;

– Corruption queries QCorrupt(i): $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the encryption key ek_i of any sender i of its choice;

– Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below.

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to QCorrupt during the whole game), and $\mathcal{HS}$ the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the three cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:

1. some QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)-$ query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^0$;
2. for some label ℓ, an encryption-query QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)$ has been asked for some $i \in \mathcal{HS}$, but encryption-queries QEncrypt $(j, x_j^0, x_j^1, \ell, S_j)$ have not all been asked for all $j \in \mathcal{HS}$;
3. for some label ℓ and for some function f asked to QDKeyGen, there exists a pair of vectors $x^0 = (x_i^0)_i, x^1 = (x_i^1)_i$ such that $f(x^0) \neq f(x^1)$, when

– $x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;

– QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)$-queries have been asked for all $i \in \mathcal{HS}$.

We say this MCABFE is IND-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}}(\mathcal{A}) = |P[\beta = 1|b = 1] - P[\beta = 1|b = 0]|$ is negligible.

3.2 Decentralized Multi-Client Attribute Based Functional Encryption

In MCABFE, an authority owns a master secret key msk to generate the functional decryption keys. We would like to avoid such a powerful authority, and make the scheme totally decentralized among the owners of the data (the senders). We thus define DMCABFE, for Decentralized Multi-Client Attribute Based Functional Encryption. In this context, there are n senders $(\mathcal{S}_i)_i$, for $i = 1,\ldots,n$, who will play the role of both the encrypting players and the functional decryption key generators, for a functional decryptor $\mathcal{FD}$. Of course, the senders do not trust each other and they want to control the functional decryption keys that will be generated. There may be several functional decryptors, but since they could collude and combine all the functional decryption keys, in the description below, and in the security model, we will consider only one functional decryptor $\mathcal{FD}$. As already noticed, we could simply use the definition of MCFE (Goldwasser et al., 2014), where the setup and the functional decryption key algorithms are replaced by MPC protocols among the clients. But this could lead to a quite interactive process. We thus focus on efficient one-round key generation protocols DKeyGen that can be split in a first step DKeyGenShare that generates partial keys and the combining algorithm DKeyComb that combines partial keys into the functional decryption key

Definition 3.3. Decentralized Multi-Client Attribute Based Functional Encryption: A decentralized multi-client Attribute Based functional encryption on $\mathcal{M}$

over a set of n senders $(\mathcal{S}_i)_i$, for $i = 1,\ldots,n$, and a functional decrypter $\mathcal{FD}$ is defined by an setup phase and four algorithms:

– SetUp(λ): This is a protocol between the senders $(\mathcal{S}_i)_i$ that eventually generate their own secret keys sk_i and encryption keys ek_i, as well as the public parameters mpk;

– Encrypt(ek_i, x_i, ℓ, S_i): Takes as input a user encryption key ek_i, a value x_i to encrypt, a label ℓ, and an attribute set S_i, and outputs the ciphertext $CT_{\ell,i}$;

– DKeyGenShare($\mathsf{sk_i}$, ℓ_f, P_i): Takes as input a user's secret key $\mathsf{sk_i}$, a label ℓ_f and an access structure P_i, and outputs a functional decryption key $dk_{f,i}$ for a function $f : \mathcal{M}^n \to \mathcal{R}$ that is described in ℓ_f, which corresponds to the access structure P_i. It also generates some parameters corresponding to the access structure for later decryption;

– DKeyGenComb($dk_{f,i}$, ℓ_f):Takes as input the partial functional decryption keys and eventually outputs the functional decryption key dk_f ;

– Decrypt(dk_f, ℓ, $CT_{\ell,i}$): Takes as input a functional decryption key dk_f , a label ℓ, and an n-vector ciphertext $CT_{\ell,i}$, and outputs $f(x)$, if $CT_{\ell,i}$ is a valid encryption of $x = (x_i)_i \in M^n$ for the label ℓ and the attribute set S_i satisfies the access structure P_i, or $\perp$ otherwise.

We make the assumption that mpk is included in msk and in all the encryption keys ek_i as well as the (partial) functional decryption keys. Similarly, the function f might be included in the (partial) functional decryption keys. The correctness property states that, given (mpk, $(\mathsf{sk_i})_i$,$(ek_i)_i$) $\leftarrow$ $\mathsf{SetUp}(\lambda)$, for any label ℓ, any function $f : \mathcal{M}^n \to \mathcal{R}$, and any vector $x = (x_i)_i \in \mathcal{M}^n$, if $CT_{\ell,i} \leftarrow$ Encrypt (ek_i, x_i, ℓ, S), for $i \in \{1,\ldots,n\}$, and $dk_f \leftarrow$ DKeyGenComb((DKeyGenShare($\mathsf{sk_i}$, ℓ_f, P_i))$_i$, ℓ_f), then Decrypt $(dk_f, \ell, CT_\ell = (CT_{\ell,i})_i) = f(x = (x_i)_i)$.

Definition 3.4. IND- Security Game for DMCABFE. Let us consider an DMCABFE scheme over a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$:

– Initialization: the challenger $\mathcal{C}$ runs the setup algorithm (mpk, msk, $(ek_i)_i$) $\leftarrow$ $\mathsf{SetUp}(\lambda)$ and chooses a random bit $b \leftarrow \{0,1\}$. It provides mpk to the adversary $\mathcal{A}$;

– Encryption queries QEncrypt (i, x^0, x^1, ℓ, S_i): $\mathcal{A}$ has unlimited and adaptive access to a Left-or-Right encryption oracle, and receives a set of ciphertexts C_{ℓ,i,S_i} generated by Encrypt (ek_i, x^b, ℓ, S_i). Noted that for the same pair (ℓ, i), further queries will later be ignored;

– Functional decryption key queries QDKeyGen(f, P_i): $\mathcal{A}$ has unlimited and adaptive access to the DKeyGenShare(msk, f, P_i) algorithm for any input function f and any access structure P_i of its choice . It is given back the functional decryption key dk_f;

– Corruption queries QCorrupt(i): $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the encryption key ek_i of any sender i of its choice;

– Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below.

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to QCorrupt during the whole game), and $\mathcal{HS}$ the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the three cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:

1. Some QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)$- query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^0$;
2. For some label ℓ, an encryption-query QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)$ has been asked for some $i \in \mathcal{HS}$, but encryption-queries QEncrypt $(j, x_j^0, x_j^1, \ell, S_j)$ have not all been asked for all $j \in \mathcal{HS}$;
3. For some label ℓ and for some function f asked to QDKeyGen, there exists a pair of vectors $x^0 = (x_i^0)_i, x^1 = (x_i^1)_i$ such that $f(x^0) \neq f(x^1)$, when

– $x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;

– QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)$-queries have been asked for all $i \in \mathcal{HS}$.

We say this DMCABFE is IND-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IND}}(\mathcal{A}) = |P[\beta = 1|b = 1] - P[\beta = 1|b = 0]|$ is negligible.

4 MULTI-CLIENT ATTRIBUTE-BASED FUNCTIONAL ENCRYPTION FOR INNER PRODUCT

In this section, we mainly propose a scheme of MCABFE for Inner Product. In our scheme, each user is related with a policy. In the Encrypt Phase, each data sender $\mathcal{S}_i$ sets a data set S_i in the partial ciphertext CT_i, and the integrated ciphertext CT is composed of all the partial ciphertexts. Each data sender $\mathcal{S}_i$ owns a decryption key dk_y which corresponds to the user's policy P_i , and can decrypt the functional value from the integrated ciphertext CT if P_i matches the union of all S_i.

4.1 Construction

We use a prime-order group, the pairing group as defined in Section 2.

- SetUp(1^λ): Takes as input the security parameter, and generates pairing group $\mathcal{G} := (\mathbb{G}, p, P, e) \xleftarrow{\$} PGGen(1^\lambda)$, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}^2$. It also generates the encryption keys $s_i \xleftarrow{\$} \mathbb{Z}_p^2$, for $i = 1, \ldots, n$. The public parameters mpk consist of $(\mathcal{G}, \mathcal{H})$, while the encryption keys are $ek_i = s_i$ for $i = 1, \ldots, n$, and the master secret key is $\mathsf{msk} = ((ek_i)_i)$, (in addition to mpk, which is omitted);
- Encrypt(ek_i, x_i, ℓ, S_i): Takes as input the value x_i to encrypt, under the key $ek_i = s_i$ and the label ℓ, and an attribute set S_i. For each attribute $j \in S_i$, it computes $[u_{\ell,j}] := \mathcal{H}(\ell, j) \in \mathbb{G}^2$, chooses random numbers $t_j \in \mathbb{Z}_p, \forall j$ and outputs the ciphertext $CT_{\ell,i,S_i} = (\ [c_{i,j}] = [u_{\ell,j}^\top s_i t_j][x_i]$, where $[u_{\ell,j}^\top s_i t_j]$ and $[x_i] \in \mathbb{G}$, and $[u_{\ell,j}]^{t_j} := \mathcal{H}(\ell, j)^{t_j} \in \mathbb{G}^2) \ \forall j$;
- DKeyGen(msk, y, $P_i = (M, \rho)$): Takes as input $\mathsf{msk} = (s_i)_{i\in[n]}$ and an inner-product function defined by y as $f_y(x) = \langle x, y\rangle$, and an access structure $P_i = (M, \rho)$, where M is an $l \times n$ matrix, and function ρ associates rows of M to attributes. It chooses random numbers r_j and generates a random vector $\vec{r} = (r_{1,j}, r_{2,j}, \ldots, r_{n,j})$. For $j = 1$ to l, it computes $\lambda_j = \vec{r} \cdot M_j$. It chooses random numbers $q_j \in \mathbb{Z}_p$, and generates the functional decryption key $dk_y = ([\vec{y}], [-q_j \sum_i s_i y_i r_j], [\lambda_j q_j]) \in \mathbb{Z}_p^2 \times \mathbb{Z}_p^2, \forall i, j$, which corresponds to an access structure P_i. Then it sends this decryption to the user associated to P_i;
- Decrypt(dk_y, ℓ, $(CT_{\ell,i,S_i})_{i\in[n]}$): Takes as input a user's functional decryption key dk_y, and a label ℓ, and ciphertexts $(CT_{\ell,i,S_i})_{i\in[n]}$. Let attribute set S' be the union set of all the attribute sets S_i over the ciphertexts. Suppose the access structure P_i satisfies S', and let $J \subset \{1, 2, ..., \}$ be defined as $J = \{j : \rho(j) \in S'\}$. Then, let $\omega_j \in \mathbb{Z}_p j \in I$ be a set of constants such that if λ_j are valid shares of any secrets according to M, then $\omega_j \lambda_j = r_j$.
  It computes $[u_{\ell,j}] := \mathcal{H}(\ell, j)$, and computes: $[\alpha]_T = \frac{\sum_i e([c_{i,j}],[y_i])}{\prod_{j\in J}(e([-q_j \cdot \sum_i s_i y_i r_j[,[u_{\ell,j}]^{t_j}) \cdot e([\lambda_j \cdot q_j], g))^{\omega_j}}$
  $= \frac{e(g,[u_{\ell,j}])^{\sum_i s_i y_i t_j} \cdot e(g,g)^{\sum_i x_i \cdot y_i}}{e(g,[u_{\ell,j}])^{\sum_i s_i y_i t_j}} = [\langle x, y\rangle]_T$.
  Finally, it eventually solves the discrete logarithm to extract and return $\alpha = \langle x, y\rangle$.

Note that, as for (Agrawal et al., 2016), the result α must be polynomially bounded to efficiently compute the discrete logarithm in the last decryption step.

4.2 Security Proof

Theorem 4.1. The above MCABFE scheme is IND-secure under the DDH assumption, in the random oracle model. More precisely, we have $\mathsf{Adv}^{\mathsf{IND}}(\mathcal{A}) \leq 2Q \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4Q \times t_{\mathbb{G}}) + \frac{2Q}{p}$, for any adversary $\mathcal{A}$, running within time t, where Q is the number of (direct and indirect —asked by QEncrypt-queries —) queries to $\mathcal{H}$ (modeled as a random oracle), and $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.

We emphasize that 4.1 supports both adaptive encryption queries and adaptive corruptions.

To obtain adaptive security, we use a technique that consists of first proving perfect security in the selective variant of the involved games, then, using a guessing (a.k.a. complexity leveraging) argument, which incurs an exponential security loss, we obtain the same security guarantees in the adaptive games. Since the security in the selective game is perfect (the advantage of any adversary is exactly zero), the exponential security loss is multiplied by a zero term, and the overall adaptive security is preserved.

Proof. We proceed using hybrid games, described in Table 1. Let $\mathsf{RF}, \mathsf{RF}', \mathsf{RF}''$ denote random functions onto $\mathbb{G}^2, \mathbb{Z}_p$, and $\mathbb{Z}_p^*$, respectively. Let $\mathcal{A}$ be a PPT adversary. For any game G_{index}, we denote by $\mathsf{Adv}_{\mathsf{index}} := |Pr[G_{index}(\mathcal{A})|b = 1] - Pr[G_{index}(\mathcal{A})|b = 0]|$, where the probability is taken over the random coins of G_{index} and $\mathcal{A}$. Also, by event $G_{index(\mathcal{A})}$, or just G_{index} when there is no ambiguity, we mean that the Finalize procedure in game G_{index} returns $\beta = 1$ from $\mathcal{A}$'s answer b' when interacting with $\mathcal{A}$.

Game G_0: This is the IND-security game as given in Definition 3.2. Note that the hash function $\mathcal{H}$ is modeled as a random oracle RO onto $\mathbb{G}^2$. This is essentially used to generate $[u_{\ell,j}] = \mathcal{H}(\ell, j)$.

Game G_1: We simulate the answers to any new RO-query by a truly random pair in $\mathbb{G}^2$, on the fly. The simulation remains perfect, and so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: We simulate the answers to any new RO-query by a truly random pair in the span of $[a]$ for $\binom{1}{a}$, with $a \xleftarrow{\$} \mathbb{Z}_p$. This uses the Multi-DDH assumption, which tightly reduces to the DDH assumption using the random-self reducibility: Adv_1 - $\mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4Q \times t_{\mathbb{G}})$, where Q is the number of RO-queries and $t_{\mathbb{G}}$) the time for an exponentiation.

Game G_3: We simulate any QEncrypt query as the encryption of x_i^0 instead of x_i^b and go back for the answers to any new RO query by a random pair in $\mathbb{G}^2$.

While it is clear that in this last game the advantage of any adversary is exactly 0 since b does not appear anywhere, the gap between G_2 and G_3 will be proven using a hybrid technique on the RO-queries. We thus index the following games by q,

where $q = 1,\ldots,Q$. Note that only distinct RO-queries are counted, since a second similar query is answered as the first one. We detail this proof because the technique is important.

$G_{3.1.1}$: This is exactly game G_2. Thus, $\mathsf{Adv}_2 = \mathsf{Adv}_{3.1.1}$.

$G_{3.q.1} \rightsquigarrow G_{3.q.2}$: We first change the distribution of the output of the q-th RO-query, from uniformly random in the span of $[a]$ to uniformly random over $\mathbb{G}^2$, using the DDH assumption. Then, we use the basis $(\binom{1}{a}, \binom{-a}{1})$ of $\mathbb{Z}_p^2$, to write a uniformly random vector over $\mathbb{Z}_p^2$ as $u_1 \cdot a + u_2 \cdot a^\top$, where $u_1, u_2 \xleftarrow{\$} \mathbb{Z}_p$. Finally, we switch to $u_1 \cdot a + u_2 \cdot a^\top$ where $u_1 \xleftarrow{\$} \mathbb{Z}_p$, and $u_2 \xleftarrow{\$} \mathbb{Z}_p^*$, which only changes the adversary view by a statistical distance of $1/p$: $\mathsf{Adv}_{3.q.1}\text{-}\mathsf{Adv}_{3.q.2} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathsf{t}) + 1/\mathsf{p}$. The last step with $u_2 \in \mathbb{Z}_p^*$ will be important to guarantee that $u^\perp_{\ell,j} a^\top \neq 0$

$G_{3.q.2} \rightsquigarrow G_{3.q.3}$: We now change the generation of the ciphertext $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^b]$ by $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^0]$, where $[u_{\ell,j}]$ corresponds to the q-th RO-query. We then prove this does not change the adversary's view.

Note that if the output of the q-th RO-query is not used by QEncrypt-queries, then the games $G_{3.q.2}$ and $G_{3.q.3}$ are identical. But we can show this is true too when there are RO-queries that are really involved in QEncrypt-queries, and show that $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$ in that case too, in two steps. In Step 1, we show that there exists a PPT adversary $\mathcal{B}^*$ such that $\mathsf{Adv}_{3.q.t} = (p^2+1)^n \cdot \mathsf{Adv}^*_{3.q.t}(\mathcal{B}^*)$, for $t = 2, 3$, where the games $G^*_{3.q.2}$ and $G^*_{3.q.3}$ are selective variants of games $G_{3.q.2}$ and $G_{3.q.3}$ respectively (see in Table 2), where QCorrupt queries are asked before the initialization phase. In Step 2, we show that for all PPT adversaries $\mathcal{B}^*$, we have $\mathsf{Adv}^*_{3.q.2}(\mathcal{B}^*) = \mathsf{Adv}^*_{3.q.3}(\mathcal{B}^*)$. This will conclude the two steps.

Step 1. We build a PPT adversary $\mathcal{B}^*$ playing against $G^*_{3.q.t}$ for $t = 2, 3$, such that $\mathsf{Adv}_{3.q.t} = (p^2+1)^n \cdot \mathsf{Adv}^*_{3.q.t}(\mathcal{B}^*)$.

Adversary $\mathcal{B}^*$ first guesses for all $i \in [n], z_i \xleftarrow{\$} \mathbb{Z}_p^2 \cup \{\perp\}$, which it sends to its selective game $G^*_{3.q.t}$. That is, each guess z_i is either a pair of values (x_i^0, x_i^1) queried to QEncrypt, or $\perp$, which means no query to QEncrypt. Then, it simulates $\mathcal{A}$'s view using its own oracles. When $\mathcal{B}^*$ guesses successfully (call E that event), it simulates $\mathcal{A}$'s view exactly as in $G_{3.q.t}$. If the guess was not successful, then $\mathcal{B}^*$ stops the simulation and outputs a random bit β. Since event E happens with probability $(p^2+1)^{-n}$ and is independent of the view of adversary $\mathcal{A}$: $\mathsf{Adv}^*_{3.q.3}(\mathcal{B}^*)$ is equal to

$$|Pr[G^*_{3.q.t}|b=0,E]\cdot Pr[E] + \tfrac{Pr[\neg E]}{2} - Pr[G^*_{3.q.t}|b = 1,E]\cdot Pr[E] - \tfrac{Pr[\neg E]}{2}|$$
$$= Pr[E]\cdot|Pr[G^*_{3.q.t}|b=0,E] - Pr[G^*_{3.q.t}|b=1,E]|$$
$$= (p^2+1)^{-n}\cdot \mathsf{Adv}_{3.q.t}$$

Step 2. We assume the values $(z_i)_{i\in[n]}$ sent by $\mathcal{B}^*$ are consistent, that is, they don't make the game end and return a random bit, and Finalize on b' does not return a random bit independent of b' (call E' this event).

We show that games $G^*_{3.q.2}$ and $G^*_{3.q.3}$ are identically distributed, conditioned on E'. To prove it, we use the fact that the two following distributions are identical, for any choice of γ, $(s_i)_{i\in[n],z_i=(x_i^0,x_i^1)}$ and $\left(s_i + a^\perp \cdot \gamma(x_i^b - x_i^0)\right)_{i\in[n],z_i=(x_i^0,x_i^1)}$ where $a^\perp := \binom{-a}{1} \in \mathbb{Z}_p^2$ and $s_i \xleftarrow{\$} \mathbb{Z}_p^2$, for all $i = 1,\ldots,n$. This is true since the s_i are independent of the z_i (note that this is true because we are in a selective setting, while this would not necessarily be true with adaptive QEncrypt-queries). Thus, we can re-write s_i into $s_i + a^\perp \cdot \gamma(x_i^b - x_i^0)$ without changing the distribution of the game.

We now take a look at where the extra terms $a^\perp \cdot \gamma(x_i^b - x_i^0)$ actually appear in the adversary's view:

– They do not appear in the output of QCorrupt, because we assume event E' holds, which implies that if $z_i \neq \perp$, then i is not queried to QCorrupt or $x_i^1 = x_i^0$.

– They might appear in QDKeyGen($\vec{y}$) as $dk_y = [-q_j(\sum_{i\in[n]} s_i y_i r_j + a^\perp \cdot \gamma \sum_{i:z_i=(x_i^0,x_i^1)} y_i(x_i^b - x_i^0)r_j]$.

But the item $a^\perp \cdot \gamma \sum_{i:z_i=(x_i^0,x_i^1)} y_i(x_i^b - x_i^0)r_j$ equals 0 by the constraints for E' in Definition 3.2: for all $i \in \mathcal{HS}$, $z_i \neq \perp$; if $i \in \mathcal{CS}$ and $z_i \neq \perp$, $x_i^1 = x_i^0$ and $f(x^0) = f(x^1)$, hence $\sum_{i:z_i=(x_i^0,x_i^1)} y_i(x_i^b - x_i^0) = 0$.

– Eventually, they appear in the output of the QEncrypt-queries which use $[u_{\ell,j}]$ computed on the q-th RO-query, since for all others, the vector $[u_{\ell,j}]$ lies in the span of $[a]$, and $a^\top a^\perp = 0$. We thus have $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j + (x_i^b - x_i^0)\gamma u_{\ell,j}^\top a^\perp t_j][x_i^b]$. Since $u_{\ell,j}^\top a^\perp \neq 0$, we can choose $\gamma = -1/u_{\ell,j}^\top a^\perp \bmod p$, and then $[c_{i,j}] = [u_\ell^\top] \cdot s_i + [x_i^0]$, which is the encryption of x_i^0. We stress that γ is independent of the index i, and so this simultaneously converts all the encryptions of x_i^b into encryptions of x_i^0. Finally, reverting these statistically perfect changes, we obtain that $[c_{i,j}]$ is identically distributed to $[u_\ell^\top] \cdot s_i + [x_i^0]$, as in game $G*_{3.q.3}$.

Thus, when event E' happens, the games are identically distributed. When $\neg E$ happens, the games both return $\beta \xleftarrow{\$} \{0,1\}$: $\mathsf{Adv}^*_{3.q.2}(\mathcal{B}^*) = \mathsf{Adv}^*_{3.q.3}(\mathcal{B}^*)$. As a conclusion, we get $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$.

$G_{3.q.3} \rightsquigarrow G_{3.q+1.1}$: This transition is the reverse of $G_{3.q.1} \rightsquigarrow G_{3.q.2}$, namely, we use the DDH assumption to switch back the distribution of $[u_{\ell,j}]$ computed on the q-th RO-query from uniformly random over $\mathbb{G}^2$

Table 1: Games for the proof of Theorem 5.1.

Game G_0, G_1, $G_2, (G_{3.q.1})_{q\in[Q+1]}, (G_{3.q.2}, G_{3.q.3})_{q\in[Q]}$:

$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n], s_i \xleftarrow{\$} \mathbb{Z}_p^2$, $ek_i := s_i$, msk $:= (s_i)_i$, mpk $:= (\mathbb{G}, p, g, e)$.

$a \xleftarrow{\$} \mathbb{Z}_p, a := \binom{1}{a}, a^\perp := \binom{-a}{1}$

Sample a full-domain hash function $\mathcal{H}$ onto $\mathbb{G}^2$, and a bit $b \xleftarrow{\$} \{0,1\}$.
$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot,\cdot),\mathsf{QCorrupt}(\cdot,\cdot),\mathsf{RO}(\cdot)}(mpk)$. Run Finalize on b'.

RO (ℓ, j): // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$

$[u_{\ell,j}] := \mathcal{H}(\ell, j)$, $[u_{\ell,j}] := \mathsf{RF}(\ell, j)$, $[u_{\ell,j}] := [\vec{a} \cdot r_{\ell,j}]$, with $r_{\ell,j} := \mathsf{RF}'(\ell, j)$

On the q'th (fresh) query: $[u_{\ell,j}] := \mathsf{RF}'(\ell, j) \cdot \vec{a} + \mathsf{RF}"(\ell, j) \cdot \vec{a}^\perp$. Return $[u_{\ell,j}]$.

QEncrypt $(i, x_i^0, x_i^1, \ell, S_i)$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$

For each attribute $j \in S_i$, $[u_{\ell,j}] := \mathsf{RO}(\ell, j)$, $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^b]$

If $[u_{\ell,j}]$ is computed on the n'-th RO-query, for $n' < q$: $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^0]$

If $[u_{\ell,j}]$ is computed on the q-th RO-query, $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^0]$. Return $[c_{i,j}]$

QDKeyGen$(y, P_i = (M, \rho))$: // $G_0, G_1, G_2, G_{3.q.1}, G_{3.q.2}, G_{3.q.3}$

Return $[\vec{y}], [-q_j \sum_i s_i y_i r_j], [\lambda_j q_i]$.

QCorrupt (i): Return s_i // $G_0, G_1, G_2, G_{3.q.1}, G_{3.q.2}, G_{3.q.3}$

(conditioned on the fact that $u_{\ell,j}^\top a^\perp \neq 0$) to uniformly random in the span of $[a]$: $\mathsf{Adv}_{3.q.3} - \mathsf{Adv}_{3.q+1.1} \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}} + 1/p$.

As a conclusion, since $G_{3.q+1.1} = G_3$, we have $\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq 2Q(\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}} + 1/p)$. In addition, $\mathsf{Adv}_3 = 0$, which concludes the proof.

5 DECENTRALIZED MULTI-CLIENT ATTRIBUTE BASED FUNCTIONAL ENCRYPTION FOR INNER PRODUCT

In MCABFE for Inner Product, a trusted authority is required and runs the Setup phase. While the authority can generate and hold the master key, it can easily undermine any user's privacy if the authority is malicious or corrupted. Thus we try to build a DMCABFE for Inner Product, in which the trusted third-party is not required, and the generation of functional decryption keys remains an efficient process under the control of the clients themselves. We use a non-interactive generation to generate the functional decryption keys.

5.1 Construction

We use a prime-order group, the pairing group as defined in Section 2.

- SetUp(1^λ, U): Takes as input the security parameter and the attribute universe U, and generates pairing group $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) \xleftarrow{\$} PGGen(1^\lambda)$, and samples two full-domain hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ onto $\mathbb{G}_1^2$ and $\mathbb{G}_2^2$, respectively. Each sender $\mathcal{S}_i$ generates $s_i \xleftarrow{\$} \mathbb{Z}_p^2$, for $i = 1, \ldots, n$. Then, the user interactively generates $T_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ such that $\sum_{i\in[n]} T_i = 0$, and a random numbers r. The public parameters mpk is set as $(\mathcal{PG}, \mathcal{H}_1, \mathcal{H}_2)$, and for $i = 1, \ldots, n$, the encryption keys $ek_i = s_i$, and $sk_i = (s_i, T_i, r)$;
- Encrypt(ek_i, x_i, ℓ, S_i): Takes as input the value x_i to encrypt, under the key $ek_i = s_i$ and the label ℓ, and an attribute set S_i. For each attribute $j \in S_i$, it computes $[u_{\ell,j}]_1 := \mathcal{H}_1(\ell, j) \in \mathbb{G}^2$, chooses random numbers $t_j \in \mathbb{Z}_p, \forall j$ and outputs the ciphertext $CT_{\ell,i} = (\ [c_{i,j}]_1 = [u_{\ell,j}^\top s_i t_j]_1 [x_i]_1$, where $[u_{\ell,j}^\top s_i t_j]_1$ and $[x_i]_1 \in \mathbb{G}_1$,and $[u_{\ell,j}]_1^{t_j} := \mathcal{H}_1(\ell, j)^{t_j} \in \mathbb{G}^2) \ \forall j$;
- DKeyGenShare (sk_i, y, $P_i = (M, \rho)$): Takes as input sk_i and an inner-product function defined by y as $f_y(x) = \langle x, y \rangle$, and an access structure P_i =

Table 2: Games $G^*_{3.q.2}$ and $G^*_{3.q.3}$, with $q \in [Q]$, for the proof of Theorem 5.1.

Game $(G^*_{3.q.2}, G^*_{3.q.3})_{q\in[Q]}$:
$(\mathsf{state}, (z_i \in \mathbb{Z}_p^2 \cup \{\perp\}_{i\in[n]})) \leftarrow \mathcal{A}(1^\lambda, 1^n)$
$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n], s_i \xleftarrow{\$} \mathbb{Z}_p^2$, $ek_i := s_i$, msk $:= (s_i)_i$, mpk $:= (\mathbb{G}, p, g)$.
$a \xleftarrow{\$} \mathbb{Z}_p, a := \binom{1}{a}, a^\perp := \binom{-a}{1}$ $b \xleftarrow{\$} \{0,1\}$,
$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot,\cdot),\mathsf{QCorrupt}(\cdot,\cdot),\mathsf{RO}(\cdot)}(mpk, \mathsf{state})$. Run Finalize on b'
$\mathsf{RO}(\ell, \mathsf{j})$: // $G^*_{3.q.2}$, $G^*_{3.q.3}$
$[u_{\ell,j}] := [a \cdot r_{\ell,j}]$, with $r_{\ell,j} := \mathsf{RF}'(\ell, \mathsf{j})$
On the q'th (fresh) query: $[u_{\ell,j}] := [\mathsf{RF}'(\ell, \mathsf{j}) \cdot \tilde{\mathsf{a}} + \mathsf{RF}''(\ell, \mathsf{j}) \cdot \tilde{\mathsf{a}}^\perp]$. Return $[u_{\ell,j}]$.
$\mathsf{QEncrypt}\ (i, x_i^0, x_i^1, \ell, S_i)$: // $\boxed{G^*_{3.q.2}}$, $G^*_{3.q.3}$
$[u_{\ell,j}] := \mathsf{RO}(\ell, \mathsf{j})$, $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^b]$
If $[u_{\ell,j}]$ is computed on the n' RO-query, for $n' < q$: $[c_{i,j}] := [u_{\ell,j}^\top s_i t_j][x_i^0]$
If $[u_{\ell,j}]$ is computed on the q-th RO-query, then,
· if $(x_i^0, x_i^1) \neq z_i$, the game ends and returns $\beta \xleftarrow{\$} \{0,1\}$.
otherwise, $[c_{i,j}] := [c_{i,j}] := [u_{\ell,j}^\top s_i t_j]$ $\boxed{\cdot[x_i^b]}$ $\cdot[x_i^0]$, $\mathcal{S} := \mathcal{S} \cup \{i\}$. Return $[c_{i,j}]$
$\mathsf{QDKeyGen}(y)$: // $G^*_{3.q.2}$, $G^*_{3.q.3}$
Return $[\vec{y}], [-q_j \sum_i s_i y_i r_j], [\lambda_j q_i]$.
$\mathsf{QCorrupt}\ (i)$: // $G^*_{3.q.2}$, $G^*_{3.q.3}$
If $z_i = (x_i^0, x_i^1)$ with $x_i^0 \neq x_i^1$ the game ends and returns $\beta \xleftarrow{\$} \{0,1\}$, Return s_i.

(M,ρ), where M is an $l \times n$ matrix, and function ρ associates rows of M to attributes. It chooses random number $r_2, \ldots, r_n$ to generate an random vector $\vec{v}_r = (r, r_2, \ldots, r_n)$. For $j = 1$ to l, it computes $\lambda_j = \vec{r} \cdot M_j$. Then it computes $[v_y]_2 := \mathcal{H}_2(y) \in \mathbb{G}_2^2$, $[d_{i,j}]_2 = [s_i y_i r + T_i v_y]_2$, and returns the partial decryption key $[d_{i,j}]_2$.

– DKeyGenComb($[d_{i,j}]_2$): For each attribute $j \in S_i$, it computes: $[d_j]_2 = \sum_{i\in[n]}[d_{i,j}]_2 = \sum_{i\in[n]}[s_i y_i r + T_i v_y]_2 = \sum_{i\in[n]}[s_i y_i r]_2 + [v_y]_2 \sum_{i\in[n]} T_i = \sum_{i\in[n]}[s_i y_i r]_2$.
Then outputs the functional decryption key $dk_y = ([\vec{y}]_2, [d_j]_2 = [\sum_i s_i y_i r_j]) \in \mathbb{Z}_p^2 \times \mathbb{Z}_p^2, \forall i, j$;

– Decrypt(dk_y, ℓ, $(CT_{\ell,i})_{i\in[n]}$): Takes as input a functional decryption key dk_y, and a label ℓ, and ciphertexts. Suppose that S satisfies the access structure and let $J \subset \{1, 2, ..., \}$ be defined as $J = \{j : \rho(j) \in S_i\}$. Then, let $\omega_j \in \mathbb{Z}_p$ $j \in J$ be a set of constants such that if λ_j are valid shares of any secrets according to M, then $\sum_{j\in J} \omega_j \lambda_j = r$.
It computes $[u_{\ell,j}]_2 := \mathcal{H}_2(\ell, j)$, then computes

$$[\alpha]_T = \frac{\sum_i e([c_{i,j}]_1, [y_i]_2)}{\prod_{j\in J}(e([\sum_i s_i y_i r_j]_2, [u_{\ell,j}]_1^{t_j}) \cdot [\lambda_j]_T)^{\omega_j}}$$
$$= \frac{e(g_2, [u_{\ell,j}]_1)^{\sum_i s_i y_i t_j} \cdot e(g_1, g_2)^{\sum_i x_i \cdot y_i}}{e(g_2, [u_{\ell,j}]_1)^{\sum_i s_i y_i t_j}}$$
$$= e(g_1, g_2)^{\sum_i x_i \cdot y_i} = [\langle x, y \rangle]_T.$$

Finally, it eventually solves the discrete logarithm to extract and return $\alpha = \langle x, y \rangle$.

Note that, as for (Agrawal et al., 2016), the result α must be polynomially bounded to efficiently compute the discrete logarithm in the last decryption step.

5.2 Security Proof

Theorem 5.1. The above DMCABFE scheme is sta-IND-secure under the SXDH assumption, in the random oracle model. More precisely, for for any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$\mathsf{Adv}^{\mathsf{IND}}(\mathcal{A}) \leq 2Q_1 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2Q_2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + \frac{2Q_1+2Q_2}{p} + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + 4Q_1 \times t_{\mathbb{G}_1}) + 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2}),$$

where Q_1 and Q_2 are the number of (direct and indirect) queries to $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively (modeled as random oracles). The former being asked by QEncrypt-queries and the latter being asked by QDKeyGen-queries.

We note that 5.1 supports adaptive encryption queries, and static corruptions.

Proof. We proceed using hybrid games, described in Table 3. Let $\mathsf{RF}, \mathsf{RF}'$ denote random functions onto $\mathbb{G}^2$ and $\mathbb{Z}_p$, respectively. Let $\mathcal{A}$ be a PPT adversary. For any game G_{index}, we denote by $\mathsf{Adv}_{\mathsf{index}} := |Pr[G_{index}(\mathcal{A})|b = 1] - Pr[G_{index}(\mathcal{A})|b = 0]|$, where

the probability is taken over the random coins of G_{index} and $\mathcal{A}$. Also, by event $G_{index(\mathcal{A})}$, or just G_{index} when there is no ambiguity, we mean that the Finalize procedure in game G_{index} returns $\beta = 1$ from the adversary's answer b' when interacting with $\mathcal{A}$.

Game G_0: This is the sta-IND-security game as given in Definition 3.4, but with the set $\mathcal{CS}$ of corrupted senders known from the beginning. Note that the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ are modeled as random oracles. The former is used to generate $[u_{\ell,j}]_1 = \mathcal{H}_\infty(\ell) \in \mathbb{G}_1^2$ and the latter $[v_y]_2 = \mathcal{H}_\in(y) \in \mathbb{G}_2^2$.

Game G_1: We replace the hash function $\mathcal{H}_2$ by a random oracle RO_2 that generates random pairs from $\mathbb{G}_2^2$ on the fly. In addition, for any QDKeyGen-query on a corrupted index $i \in \mathcal{CS}$, one generates the partial functional decryption key by itself, without explicitly querying QDKeyGen. Hence, we can assume that $\mathcal{A}$ does not query QCorrupt and QDKeyGen on the same indices $i \in [n]$. The simulation remains perfect, and so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: Now, the outputs of RO_2 are uniformly random in the span of $[b]_2$ for $b := \binom{1}{a'}$, with $a' \xleftarrow{\$} \mathbb{Z}_p$. As in the previous proof, we have $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2})$, where Q_2 is the number of RO_2-queries and $t_{\mathbb{G}_2}$ the time for an exponentiation.

Game G_3: We replace all the partial key decryption answers by $d_{i,j} := [y_i \cdot s_i r + w_i \cdot (b^\perp)^\top v_y + T_i v_y]_2$, for new $w_i \xleftarrow{\$} \mathbb{Z}_p^2$, such that $\sum_i w_i = 0$, for each y. This sum being among the honest clients, we need to know the last queried honest client to set this sum to zero. Hence the requirement to know the set of honest clients, and thus just security against static corruptions. We show below that $\mathsf{Adv}_2 = \mathsf{Adv}_3$.

Game G_4: We switch back the distribution of all the vectors $[v_y]_2$ output by RO_2, from uniformly random in the span of $[b]_2$, to uniformly random over $\mathbb{G}_2^2$, thus back to $\mathcal{H}_2(y)$. This transition is reverse to the two first transitions of this proof: $\mathsf{Adv}_3 - \mathsf{Adv}_4 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(\mathsf{t} + 4\mathsf{Q}_2 \times \mathsf{t}_{\mathbb{G}_2})$.

$G_{3.1.1}$: This is exactly game G_2. Thus, $\mathsf{Adv}_2 = \mathsf{Adv}_{3.1.1}$.

$G_{3.q.1} \rightsquigarrow G_{3.q.2}$: We first change the distribution of the output of the q-th RO-query, from uniformly random in the span of $[b]$ to uniformly random over $\mathbb{G}^2$, using the DDH assumption. Then, we use the basis $(\binom{1}{a}, \binom{-a}{1})$ of $\mathbb{Z}_p^2$, to write a uniformly random vector over $\mathbb{Z}_p^2$ as $v_1 \cdot b + v_2 \cdot b^\top$, where $v_1, v_2 \xleftarrow{\$} \mathbb{Z}_p$. Finally, we switch to $v_1 \cdot b + v_2 \cdot b^\top$ where $v_1 \xleftarrow{\$} \mathbb{Z}_p$, and $v_2 \xleftarrow{\$} \mathbb{Z}_p^*$, which only changes the adversary view by a statistical distance of $1/p$: $\mathsf{Adv}_{3.q.1}$ - $\mathsf{Adv}_{3.q.2} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathsf{t}) + 1/\mathsf{p}$. The last step with $v_2 \in \mathbb{Z}_p^*$ will be important to guarantee that $v_y^\perp b^\top \neq 0$

$G_{3.q.2} \rightsquigarrow G_{3.q.3}$: We now change the simulation of $d_{i,j}$ from $d_{i,j} = [y_i \cdot s_i r + T_i v_y]_2$ to $d_{i,j} = [y_i \cdot s_i + \mathsf{RF_i}(y) + T_i v_y]_2$, with some $\mathsf{RF_i}$ functions onto $\mathbb{Z}_p^2$ such that $\sum_i \mathsf{RF_i}(y) = 0$ for any input y. We prove $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$.

We use the fact that the two following distributions are identical, for any choice of $w_i \xleftarrow{\$} \mathbb{Z}_p^2$, such that $\sum_i w_i = 0$, $(T_i)_{i \in \mathcal{HS}}$ and $(T_i + w_i (b^\perp)^\top)_{i \in \mathcal{HS}}$, where for all $i \in [n]$, $T_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ such that $\sum_i T_i = 0$, and $b^\perp := \binom{-a'}{1}$.

The extra terms $(w_i (b^\perp)^\top)_{i \in \mathcal{HS}}$ only appear in the output of the queries to QDKeyGen which use the vector $[v_y]_2$ computed on the q-th RO_2-query (if there are such queries), because for all other queries, $[v_y]_2$ lies in the span of $[b]_2$, and $b_\top b^\perp = 0$. We thus have $d_{i,j} := [y_i \cdot s_i r + w_i \cdot (b^\perp)^\top v_y + T_i v_y]_2$. Since v_y is such that $v_y^\top b^\perp \neq 0$, $(b^\perp)^\top v_y \neq 0$. In that case, the vectors $w_i \cdot (b^\perp)^\top v_y$ are uniformly random over $\mathbb{Z}_p^2$ such that $\sum_i w_i \cdot (b^\perp)^\top v_y = 0$, which is as in $G_{3.q.3}$, by setting $\mathsf{RF_i}(y) := w_i \cdot (b^\perp)^\top v_y$.

$G_{3.q.3} \rightsquigarrow G_{3.q+1.1}$: This transition is the reverse of$G_{3.q.1} \rightsquigarrow G_{3.q.2}$, namely, we use the DDH assumption to switch back the distribution of $[v_y]_2$ to uniformly random in the span of $[b]_2$: $\mathsf{Adv}_{3.q.3} - \mathsf{Adv3.q} + 1.1 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + 1/p$.

Then one can note that $G_{3.Q_2+1.1} = G_3$, but also that in Game G_4, all the $d_{i,j}$ output by QDKeyGen can be computed only knowing $-q \sum_{i \in [n]} [s_i \cdot y_i r]$ and q, which will reveals the functional decryption key dk_y from our MCABFE in Section 5.1.

Thus, we can reduce to the IND-security of the MCABFE from Section 5.1 (or even sta-IND - security) by designing an adversary $\mathcal{B}$ against the MCABFE from Section 5.1: Adversary $\mathcal{B}$ first samples $T_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ for all $i \in [n]$, such that $\sum_{i \in [n]} T_i = 0$ It sends $\mathcal{CS}$ given by $\mathcal{A}$ (set of static corruptions), then it receives mpk from the MCABFE security game, as well as the secret keys s_i for $i \in \mathcal{CS}$. It forwards mpk as well as (s_i, T_i) for $i \in \mathcal{CS}$ to $\mathcal{A}$. Then, $\mathcal{B}$ answers oracle calls to RO_1, RO_2 and QEncrypt from $\mathcal{A}$ using its own oracles.

To answer QDKeyGen $(i, y, P_i = (M, \rho))$: if i is the last non-corrupted index for y, $\mathcal{B}$ queries its own QDKeyGen oracle on y, to get $dk_y := ([y], [-q_j \sum_i s_i y_i r] \in \mathbb{Z}_p^2)$, computes $[v_y]_2 := \mathcal{H}_2(y)$, and returns $d_{i,j} := [dk_y + \mathsf{RF_i}(y) + T_i v_y]_2$ to $\mathcal{A}$. Otherwise, it computes $[v_y]_2 := \mathcal{H}_2(y)$, and returns $d_{i,j} := [\mathsf{RF_i}(y) + T_i v_y]_2$ to $\mathcal{A}$. The random functions $\mathsf{RF_i}$ are computed on the fly, such that their sum is the zero function.

Note that this last simulation requires to know $\mathcal{CS}$ and $\mathcal{HS}$, hence it is for static corruptions only.

Table 3: Games for the proof of Theorem 5.1.

Game G_0, G_1, $G_2, (G_{3.q.1})_{q\in[Q_{dk}+1]}$, $(G_{3.q.2}, G_{3.q.3})_{q\in[Q_{dk}]}$, G_4:

$\mathcal{PG} \leftarrow \mathsf{PGGen}(1^\lambda)$, $\forall i \in [n] : s_i \xleftarrow{\$} \mathbb{Z}_p^2, T_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$, such that $\sum_{i\in[n]} T_i = 0$, $r \xleftarrow{\$} \mathbb{Z}_p$

$ek_i := s_i$, $sk_i := (s_i, T_i, r)$, $\mathrm{mpk} := (\mathbb{G}, p, g)$. $a' \xleftarrow{\$} \mathbb{Z}_p, b := \binom{1}{a'}$

Sample full-domain hash function $\mathcal{H}_1$ onto $\mathbb{G}_1^2$ and $\mathcal{H}_2$ onto $\mathbb{G}_2^2$. Sample a bit $b \xleftarrow{\$} \{0,1\}$.

$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{QCorrupt}(\cdot,\cdot),\mathsf{RO}_1(\cdot),\mathsf{RO}_2(\cdot)}(mpk)$. Run Finalize on b'

$\mathsf{RO}_1(\ell, \mathsf{j})$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$

Return $\mathcal{H}_1(\ell, j)$.

$\mathsf{RO}_2(\mathsf{y})$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$ G_4

$[v_y]_2 := \mathcal{H}_2(y)$, $[v_y]_2 := \mathsf{RF}(\mathsf{y})$, $[v_y] := [\vec{b} \cdot t_y]$, with $t_y := \mathsf{RF}'(\mathsf{y})$

On the q'th RO_2-query: $[v_y] := \mathsf{RF}(\mathsf{y})$. Return $[v_y]_2$.

$\mathsf{QEncrypt}\ (i, x_i^0, x_i^1, \ell, S_i)$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$ G_4

For each attribute $j \in S_i$, $[u_{\ell,j}]_1 := \mathsf{RO}_1(\ell, \mathsf{j})$, $[c_{i,j}]_1 := [u_\ell^\top \cdot s_i t_j]_1 [x_i^b]_1$. Return $[c_{i,j}]$

$\mathsf{QDKeyGen}(y, i)$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $1G_{3.q.3}$, G_4

Compute $[v_y]_2 := \mathsf{RO_i}\ (y)$, $d_{i,j} := [y_i \cdot s_i r + T_i v_y]_2$, set $\mathcal{S} := \mathcal{S} \cup \{i\}$

If $[v_y]_2$ is computed on the n'-th RO_2-query, for $n' < q$: $d_{i,j} := [y_i \cdot s_i r\ \mathsf{RF_i}\ (y) + T_i v_y]_2$

If $[v_y]_2$ is computed on the q-th RO_2-query: $d_{i,j} := [y_i \cdot s_i r\ \mathsf{RF_i}\ (y) + T_i v_y]_2$

$d_{i,j} := [y_i \cdot s_i r\ \mathsf{RF_i}\ (y) + T_i v_y]_2$. Return $d_{i,j}$.

$\mathsf{QCorrupt}\ (i)$: Return s_i, T_i // $G_0, G_1, G_2, G_{3.q.1}, G_{3.q.2}, G_{3.q.3}, G_4$

From this reduction, one gets $\mathsf{Adv}_4 \leq 2Q_1 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + 4Q_1 \times t_{\mathbb{G}_1} + \frac{2Q_1}{p})$, where Q_1 denotes the number of calls to RO_1, $t_{\mathbb{G}_1}$ denotes the time to compute an exponentiation in $\mathbb{G}_1$. This concludes the proof.

6 CONCLUSION

In view of the growing demand of secure data sharing and computations on sensitive data, improving functionality and fine grained data access become an important question in Functional Encryption. In this paper, we mainly consider the problem of enhancing the Functionality of Multi-Client Functional Encryption such that the decryption can base on the attributes of a user, i.e., users with different attributes will get different decryption results. We define the first Multi-Client Attribute Based Functional Encryption scheme, and provide the first Multi-Client Attribute-Based Inner Product Functional Encryption scheme that is based on simple assumptions. Also we define the first Decentralized Multi-Client Attribute Based Functional Encryption scheme, and provide the first Multi-Client Attribute-Based Inner Product Functional Encryption scheme, which is also based on simple assumptions.

REFERENCES

Abdalla, M., Bourse, F., De Caro, A., and Pointcheval, D. (2015). Simple functional encryption schemes for inner products. In *IACR International Workshop on Public Key Cryptography*, pages 733–751. Springer.

Abdalla, M., Bourse, F., De Caro, A., and Pointcheval, D. (2016). Better security for functional encryption for inner product evaluations. *IACR Cryptology ePrint Archive*, 2016:11.

Agrawal, S., Libert, B., and Stehlé, D. (2016). Fully secure functional encryption for inner products, from standard assumptions. In *Annual International Cryptology Conference*, pages 333–362. Springer.

Ananth, P., Brakerski, Z., Segev, G., and Vaikuntanathan, V. (2015). From selective to adaptive security in functional encryption. In *Annual Cryptology Conference*, pages 657–677. Springer.

Ananth, P. and Sahai, A. (2016). Functional encryption for turing machines. In *Theory of Cryptography Conference*, pages 125–153. Springer.

Apon, D., Döttling, N., Garg, S., and Mukherjee, P. (2017). Cryptanalysis of indistinguishability obfuscations of circuits over ggh13. In *LIPIcs-Leibniz In-*

ternational Proceedings in Informatics, volume 80. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Badrinarayanan, S., Goyal, V., Jain, A., and Sahai, A. (2016). Verifiable functional encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 557–587. Springer.

Baltico, C. E. Z., Catalano, D., Fiore, D., and Gay, R. (2017). Practical functional encryption for quadratic functions with applications to predicate encryption. In *Annual International Cryptology Conference*, pages 67–98. Springer.

Bernstein, D. J., Heninger, N., Lange, T., Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B. (2013). Candidate indistinguishability obfuscation and functional encryption for all circuits.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)*, pages 321–334. IEEE.

Boneh, D., Sahai, A., and Waters, B. (2011). Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer.

Boyle, E., Chung, K.-M., and Pass, R. (2014). On extractability obfuscation. In *Theory of Cryptography Conference*, pages 52–73. Springer.

Brakerski, Z., Komargodski, I., and Segev, G. (2018). Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. *Journal of Cryptology*, 31(2):434–520.

Chase, M. (2007). Multi-authority attribute based encryption. In *Theory of cryptography conference*, pages 515–534. Springer.

Chen, Y., Gentry, C., and Halevi, S. (2017). Cryptanalyses of candidate branching program obfuscators. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 278–307. Springer.

Cheon, J. H., Han, K., Lee, C., Ryu, H., and Stehlé, D. (2015). Cryptanalysis of the multilinear map over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–12. Springer.

Cheung, L. and Newport, C. (2007). Provably secure ciphertext policy abe. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 456–465. ACM.

Chotard, J., Sans, E. D., Gay, R., Phan, D. H., and Pointcheval, D. (2018). Decentralized multi-client functional encryption for inner product. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 703–732. Springer.

Coron, J.-S., Lee, M. S., Lepoint, T., and Tibouchi, M. (2016). Cryptanalysis of ggh15 multilinear maps. In *Annual Cryptology Conference*, pages 607–628. Springer.

Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B. (2016). Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929.

Goldwasser, S., Gordon, S. D., Goyal, V., Jain, A., Katz, J., Liu, F.-H., Sahai, A., Shi, E., and Zhou, H.-S. (2014). Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 578–602. Springer.

Goldwasser, S., Kalai, Y., Popa, R. A., Vaikuntanathan, V., and Zeldovich, N. (2013a). Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564. ACM.

Goldwasser, S., Kalai, Y. T., Popa, R. A., Vaikuntanathan, V., and Zeldovich, N. (2013b). How to run turing machines on encrypted data. In *Annual Cryptology Conference*, pages 536–553. Springer.

Gorbunov, S., Vaikuntanathan, V., and Wee, H. (2012). Functional encryption with bounded collusions via multi-party computation. In *Annual Cryptology Conference*, pages 162–179. Springer.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm.

Okamoto, T. and Takashima, K. (2010). Fully secure functional encryption with general relations from the decisional linear assumption. In *Annual cryptology conference*, pages 191–208. Springer.

O'Neill, A. (2010). Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556.

Ostrovsky, R., Sahai, A., and Waters, B. (2007). Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM.

Sahai, A. and Seyalioglu, H. (2010). Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472. ACM.

Sahai, A. and Waters, B. (2005). Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer.

Waters, B. (2011). Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography*, pages 53–70. Springer.

Waters, B. (2015). A punctured programming approach to adaptively secure functional encryption. In *Annual Cryptology Conference*, pages 678–697. Springer.

Yamada, S., Attrapadung, N., Hanaoka, G., and Kunihiro, N. (2011). Generic constructions for chosen-ciphertext secure attribute based encryption. In *International Workshop on Public Key Cryptography*, pages 71–89. Springer.