Modern Family: A Revocable Hybrid Encryption Scheme Based on Attribute-Based Encryption, Symmetric Searchable Encryption and SGX (Extended Version)*

Alexandros Bakas and Antonis Michalas

Tampere University,
Tampere, Finland
{alexandros.bakas,antonios.michalas}@tuni.fi

Abstract. Secure cloud storage is considered as one of the most important issues that both businesses and end-users take into account before moving their private data to the cloud. Lately, we have seen some interesting approaches that are based either on the promising concept of Symmetric Searchable Encryption (SSE) or on the well-studied field of Attribute-Based Encryption (ABE). In the first case, researchers are trying to design protocols where users' data will be protected from both *internal* and *external* attacks without paying the necessary attention to the problem of user revocation. In the second case, existing approaches address the problem of revocation. However, the overall efficiency of these systems is compromised since the proposed protocols are solely based on ABE schemes and the size of the produced ciphertexts and the time required to decrypt grows with the complexity of the access formula. In this paper, we propose a hybrid encryption scheme that combines both SSE and ABE by utilizing the advantages of both these techniques. In contrast to many approaches, we design a revocation mechanism that is completely separated from the ABE scheme and solely based on the functionality offered by SGX.

Keywords: Cloud Security · Storage Protection · Access Control · Policies · Attribute-Based Encryption · Symmetric Searchable Encryption · Hybrid Encryption

1 Introduction

Cloud computing plays a significant role in our daily routine. From casual internet users, to big corporations, the cloud has become an integral part of our lives. However, using services that are hosted and controlled by third parties raises several security and privacy concerns. Additionally, it has been observed

* This work was funded by the ASCLEPIOS: Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare Project No. 826093 EU research project.

that the number of attacks that target users' privacy has grown significantly. For example, in [14] it is stated that there has been a 300% increase in Microsoft cloud-based user's account attacks over the past couple of years. However, when considering a cloud-based environment, cyber-attacks performed by remote adversaries is only a part of the problem. More precisely, when we design cloud services we also need to take into consideration cases where the actual cloud service provider (CSP) acts maliciously (e.g. a compromised administrator).

Therefore, organizations and researchers pay particular attention on how to protect users' data while at the same time give certain guarantees to the end users (data owners) about the "isolation" of their private information. Until recently, most approaches were offering only a certain – not acceptable – level of security. More precisely, even though users' data might be stored in an encrypted form while at rest, the encryption key was known to the CSP. As a result, users could not get any guarantees that a malicious CSP will not access their data or that their data will not be shared with third parties (unauthorized access).

To overcome this, both academia and big industrial players have started looking on how to build cloud-based services that will utilize Symmetric Searchable Encryption (SSE) [7, 4, 11] – a promising encryption technique. In such a scheme, users encrypt their files locally and send them encrypted to the CSP. Hence, the CSP who does not have access to the encryption key cannot learn anything about the content of users' data. Furthermore, whenever a user wishes to access her files, she can search directly over the encrypted data for specific keywords. Unfortunately, in an SSE scheme, revocation of a user cannot be implemented efficiently since sharing an encrypted file implies sharing the underlying encryption key. As a result, if a data owner wishes to revoke a user, then all files that are encrypted with the same key must be decrypted and then re-encrypted under a fresh key. Another promising technique that squarely fits cloud-based services is Attribute-Based Encryption (ABE). In ABE schemes, all files are encrypted under a master public key but in contrast to traditional public key encryption, the generated ciphertext is bounded by a policy. Each user has a distinct secret key which is associated with specific attributes (e.g. user's id, age, organization etc.). This way a user's secret key can decrypt a ciphertext if and only if the user's attributes satisfy the policy bounded to the ciphertext. However, using an asymmetric encryption scheme to store data is rather inefficient.

Contribution: Considering both the advantages and the disadvantages of SSE and ABE schemes, we propose a hybrid encryption scheme that combines these two promising techniques in such a way that reduces the problem of multi-user data sharing to that of a single-user. Furthermore, this work extends the protocol presented in [13]. In our construction, data is encrypted locally using SSE just as proposed by typical SSE schemes. However, the symmetric key used for encryption is stored in an SGX enclave [3], encrypted using an ABE scheme, resulting to a ciphertext bounded by a policy specified by the data owner. By doing this, when a user wishes to access files that are stored on the CSP, she first needs to obtain the encrypted key and then decrypt it using her own ABE secret key. Decryption will only work if the user's attributes satisfy the policy

of the ciphertext. This way, ABE works as a sharing mechanism and not as a revocation one, as can been seen in many recent works. Finally, we deal with the problem of revocation by utilizing the SGX functionality. This way we do not add extra computational burden to an already heavy ABE scheme.
Organization: In Section 2, we present important works that have been published and address the problem of secure cloud storage, data sharing and revocation. In Section 3, we define our system model while in Section 4, we present the cryptographic tools needed for the construction of our scheme. In Section 5, we give a formal construction of our scheme which is followed by the security analysis in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

In [15] authors present a revocable hybrid encryption scheme while at the same time a key-rotation mechanism is used to prevent key-scrapping attacks. The core idea of the paper is to use an All-or-Nothing-Transformation (AONT) [2] to prevent revoked users from accessing stored data. In particular, Optimal Asymmetric Encryption Padding (OAEP) was used as the AONT due to the fact that reversing OAEP, requires to know the entire output. Thus, by changing random bits, reversing OAEP becomes infeasible. Hence, to decrypt a file, the changed bits need to be stored so that the AONT can be later reversed. However, this implies that with each re-encryption, the size of the ciphertext grows. Thus, decrypting a file that has been re-encrypted multiple times is an expensive operation. Moreover, to make the scheme more efficient, authors suggest that the AONT could be applied by the server. However, this implies the existence of a fully trusted the server – thus, internal attacks cannot be prevented.

A promising idea is presented in [5], where the authors present a protocol based on functional encryption, with the main functionalities running in isolated environments. The decryption of a file, and the application of a function f on the decrypted file both occur in SGX enclaves. Moreover, all enclaves can attest to each other and exchange data over secure communication channels. In our construction, even though we use the same hardware principles, we build a hybrid encryption scheme by combining SSE and ABE. Additionally, we use SGX to execute vulnerable parts of the protocol and store sensitive information.

In [9] authors present a revocable ciphertext-policy attribute-based encryption scheme. The revocation mechanism is offered by a revocation list that is attached to the resulted ciphertexts. To avoid maintaining long revocation lists, a policy through which users' keys expire after a certain period of time is enforced. As a result, the revocation list only includes keys that have been revoked before the expiration date. Another Hybrid encryption scheme is presented in [6], in which authors propose a scheme based on SSE and ABE. In the proposed scheme, data owners encrypt their files using SSE, but the resulted indexes are encrypted under ABE. This way, users can locally generate search tokens based on their attributes, that are then sent to the cloud. Search only works if the user's attributes satisfy the policy of the encrypted index. However promising,

their scheme is static and as a result can only have very limited applications in real-life scenarios. Moreover, authors do not provide a revocation mechanism – a problem of paramount importance in cloud-based services [11, 18].

In our construction, we overcome these issues by designing an efficient revocation mechanism that is utilizing the SGX functionality and it is separated from the ABE scheme.

3 Architecture

In this section, we introduce the system model by explicitly describing the main entities that participate in our protocol as well as their capabilities.

Cloud Service Provider (CSP): We consider a cloud computing environment similar to the one described in [16], [17]. Moreover, the CSP must support SGX since core entities will be running in a trusted execution environment offered by SGX.

Master Authority (MS): MS is responsible for setting up all the necessary public parameters for the proper run of the involved protocols. MS is responsible for generating and distributing ABE keys to the registered users. Finally, MS is SGX-enabled and is running in an enclave called the Master Enclave.

Key Tray (KT): KT is a key storage that stores ciphertexts of the symmetric keys that have been generated by various users and are needed to decrypt data. Registered users can directly contact KT and request access to the stored ciphertexts. KT is also SGX-enabled and runs in an enclave called the KT Enclave.

Revocation Authority (REV): REV is responsible for maintaining a revocation list (rl) with the unique identifiers of the revoked users. Similar to MS and KT, REV is also SGX-enabled and is running in an enclave called the Revocation Enclave. Finally, for the security of the stored revocation list, it is important to mention that rl is generated by the enclave (i.e. in an isolated environment) and never leaves its perimeter. Therefore, there is no need to encrypt rl.

Moreover, we assume the existence of a registration authority which is responsible for the registration of users. However, registration is out of the scope of this paper and we assume that all users have been already registered. Finally, we assume that all registered users can contact any enclave through a secure channel by using an IND-CCA2 secure public key encryption scheme PKE and an EUF-CMA secure signature scheme sign.

SGX: Below we provide a brief presentation of the main SGX functionalities needed for our construction. A more detailed description can be found in [5, 3]

Isolation: Enclaves are located in a hardware guarded area of memory and they compromise a total memory of 128MB (only 90MB can be used by software). Intel SGX is based on memory isolation built into the processor itself along with strong cryptography. The processor tracks which parts of memory belong to which enclave, and ensures that *only* enclaves can access their own memory.

Attestation: One of the core contributions of SGX is the support for attestation between enclaves of the same (local attestation) and different platforms

(remote attestation). In the case of local attestation, an enclave enc_i can verify another enclave enc_j as well as the program/software running in the latter. This is achieved through a report generated by enc_j containing information about the enclave itself and the program running in it. This report is signed with a secret key $\mathsf{sk}_{\mathsf{rpt}}$ which is the same for all enclaves of the same platform. In remote attestation, enclaves of different platforms can attest each other through a signed quote. This is a report similar to the one used in local attestation. The difference is that instead of using $\mathsf{sk}_{\mathsf{rpt}}$ to sign it, a special private key provided by Intel is used. Thus, verifying these quotes requires contacting Intel's Attestation Server.

Sealing: Every SGX processor comes with a Root Seal Key with which, data is encrypted when stored in untrusted memory. Sealed data can be recovered even after an enclave is destroyed and rebooted on the same platform.

4 Cryptographic Primitives

To provide a concrete and reliable solution, we need to build a protocol through which newly encrypted data will *not* be decryptable by a user if her access has been revoked. Additionally, we want to allow users with certain access rights to be able to search directly over encrypted data. To this end, we will be using a CP-ABE scheme an a SSE scheme. In a CP-ABE scheme every secret key is generated based on a public and a private key as well as on a concrete list of attributes $\mathcal{A}$. Each ciphertext is associated with a policy P such that decryption is only possible if $P(\mathcal{A}) = \mathsf{True}$ – if the attributes on a key satisfy the policy on the ciphertext. From now on we will refer to the space of attributes as $\Omega = \{a_1, \ldots, a_n\}$, while the space of policies will be denoted as $\mathcal{P} = \{P_1, \ldots, P_m\}$.

We now proceed with the definition of the CP-ABE and SSE schemes as described in [1] and [7] respectively.

Definition 1 (Ciphertext-Policy ABE). *A revocable CP-ABE scheme is a tuple of the following five algorithms:*

- $\mathsf{CPABE.Setup}$ *is a probabilistic algorithm that takes as input a security parameter* λ *and outputs a master public key* MPK *and a master secret key* MSK*. We denote this by* $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathsf{CPABE.Gen}$ *is a probabilistic algorithm that takes as input a master secret key, a set of attributes* $\mathcal{A} \in \Omega$ *and the unique identifier of a user and outputs a secret key which is bind both to the corresponding list of attributes and the user. We denote this by* $(\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}) \leftarrow \mathsf{Gen}(\mathsf{MSK}, \mathcal{A}, u_i)$.
- $\mathsf{CPABE.Enc}$ *is a probabilistic algorithm that takes as input a master public key, a message* m *and a policy* $P \in \mathcal{P}$*. After a proper run, the algorithm outputs a ciphertext* c_P *which is associated to the policy* P*. We denote this by* $\mathsf{c_P} \leftarrow \mathsf{Enc}(\mathsf{MPK}, m, P)$.
- $\mathsf{CPABE.Dec}$ *is a deterministic algorithm that takes as input a user's secret key and a ciphertext and outputs the original message* m *iff the set of attributes* $\mathcal{A}$ *that are associated with the underlying secret key satisfies the policy* P *that is associated with* c_p*. We denote this by* $\mathsf{Dec}(\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}, c_P) \rightarrow m$.

Definition 2 (Dynamic Index-based SSE). *A dynamic index-based symmetric searchable encryption scheme is a tuple of nine polynomial algorithms* $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SearchToken}, \mathsf{AddToken}, \mathsf{DeleteToken}, \mathsf{Search}, \mathsf{Add}, \mathsf{Delete}, \mathsf{Dec})$ *:*

- $\mathsf{SSE.Gen}$ *is a probabilistic key-generation algorithm that takes as input a security parameter* λ *and outputs a secret key* K*. It is used by the client to generate her secret-key.*
- $\mathsf{SSE.Enc}$ *is a probabilistic algorithm that takes as input a secret key* K *and a collection of files* $\mathbf{f}$ *and outputs an encrypted index* γ *and a sequence of ciphertexts* $\mathbf{c}$*. It is used by the client to get ciphertexts corresponding to her files as well as an encrypted index which are then sent to the storage server.*
- $\mathsf{SSE.SearchToken}$ *is a (possibly probabilistic) algorithm that takes as input a secret key* K *and a keyword* w *and outputs a search token* $\tau_s(w)$*. It is used by the client in order to create a search token for some specific keyword. The token is then sent to the storage server.*
- $\mathsf{SSE.AddToken}$ *is a (possibly probabilistic) algorithm that takes as input a secret key* K *and a file* f *and outputs an add token* $\tau_a(f)$ *and a ciphertext* c_f*. It is used by the client in order to create an add token for a new file as well as the encryption of the file which are then sent to the storage server.*
- $\mathsf{SSE.DeleteToken}$ *is a (possibly probabilistic) algorithm that takes as input a secret key* K *and a file* f *and outputs a delete token* $\tau_d(f)$*. It is used by the client to create a delete token for some file.*
- $\mathsf{SSE.Search}$ *is a deterministic algorithm that takes as input an encrypted index* γ*, a sequence of ciphertexts* $\mathbf{c}$ *and a search token* $\tau_s(w)$ *and outputs a sequence of file identifiers* $\mathbf{I}_w \subset \mathbf{c}$*. This algorithm is used by the storage server upon receive of a search token in order to perform the search over the encrypted data and determine which ciphertexts correspond to the searched keyword and thus should be sent to the client.*
- $\mathsf{SSE.Add}$ *is a deterministic algorithm that takes as input an encrypted index* γ*, a sequence of ciphertexts* $\mathbf{c}$*, an add token* $\tau_a(f)$ *and a ciphertext* c_f *and outputs a new encrypted index* γ' *and a new sequence of ciphertexts* $\mathbf{c}'$*. This algorithm is used by the storage server (upon reception of an add token) to add a new file to the database (update encrypted index and ciphertext vector).*
- $\mathsf{SSE.Delete}$ *is a deterministic algorithm that takes as input an encrypted index* γ*, a sequence of ciphertexts* $\mathbf{c}$ *and a delete token* $\tau_d(f)$ *and outputs a new encrypted index* γ' *and a new sequence of ciphertexts* $\mathbf{c}'$*. This algorithm is used by the storage server upon receive of a delete token in order to update the encrypted index and the ciphertext vector to delete the data corresponding to the deleted file.*
- $\mathsf{SSE.Dec}$ *is a deterministic algorithm that takes as input a secret key* K *and a ciphertext* c *and outputs a file* f*. It is used by the client to decrypt the ciphertexts that she gets from the storage server.*

The security of an SSE scheme is based on the existence of a simulator that is given as input information leaked during the execution of the protocol. In particular to define the security of SSE we make use of the leakage functions $\mathcal{L}_{in}, \mathcal{L}_s, \mathcal{L}_a, \mathcal{L}_d$ associated to index creation, search, add and delete operations [4].

Definition 3. *(Dynamic CKA 2-security). Let* $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SearchToken}, \mathsf{AddToken}, \mathsf{DeleteToken}, \mathsf{Search}, \mathsf{Add}, \mathsf{Delete}, \mathsf{Dec})$ *be a dynamic index based symmetric searchable encryption scheme and* $\mathcal{L}_{in}, \mathcal{L}_s, \mathcal{L}_a, \mathcal{L}_d$ *be leakage functions associated to index creation, search, add and delete operations. We consider the following experiments between an adversary* $\mathcal{ADV}$ *and a challenger* $\mathcal{C}$*:*

Real$_{\mathcal{ADV}}(\lambda)$

$\mathcal{C}$ runs $\mathsf{Gen}(1^\lambda)$ *to generate a key* K*. $\mathcal{ADV}$ outputs a file* $\mathbf{f}$ *and receives* $(\gamma, c) \leftarrow \mathsf{Enc}(\mathsf{K}, f)$ *from $\mathcal{C}$. $\mathcal{ADV}$ makes a polynomial time of adaptive queries* $q = \{w, f_1, f_2\}$ *and for each q he receives back either a search token for w,* $\tau_s(w)$*, an add token and a ciphertext for* f_1*,* $(\tau_\alpha(f_1), c_1)$ *or a delete token for* f_2*,* $\tau_d(f_2)$*. Finally, $\mathcal{ADV}$ outputs a bit b.*

Ideal$_{\mathcal{ADV},\mathcal{S}}(\lambda)$

$\mathcal{ADV}$ outputs a file $\mathbf{f}$*. $\mathcal{S}$ is given* $\mathcal{L}_{in}$ *and generates* (γ, c) *which is sent back to $\mathcal{ADV}$. $\mathcal{ADV}$ makes a polynomial time of adaptive queries* $q = \{w, f_1, f_2\}$ *and for each q, $\mathcal{S}$ is given either* $\mathcal{L}_s(\mathbf{f}, w), \mathcal{L}_\alpha(\mathbf{f}, f_1)$ *or* $\mathcal{L}_d(\mathbf{f}, f_2)$*. $\mathcal{S}$ then returns a token and, in the case of addition, a ciphertext c. Finally, $\mathcal{ADV}$ outputs a bit b.*

We say that the SSE *scheme is* $\mathcal{L}$*-i secure if for all probabilistic polynomial adversaries* $\mathcal{ADV}$*, there exists a probabilistic simulator* $\mathcal{S}$ *such that:*

$$|Pr[(Real) = 1] - Pr[Ideal) = 1]| \leq negl(\lambda)$$

In the cases of file addition and deletion, the simulator must also generate ciphertexts and update the current indexes. A good example of such a scheme, can be found in [7].

5 Modern Family (MF)

In this section, we present Modern Family (MF) – the core of this paper's contribution. We start by giving an overview of the SGX hardware functionalities used by the communicating parties and we continue with a formal construction.

Hardware: During the execution of the protocol, all parties have access to the secure hardware as defined in [5]. In the beginning, HW.Setup runs to produce the secret key needed to verify reports. Each enclave is then initialized by loading a program Q and producing a handle hdl which is used as an identification for the enclave running Q. This is done by running the HW.Load interface. After the initialization of the enclave, HW.Run is executed with different inputs. For simplicity, we assume that all enclaves run on the same host, so *they only perform local attestations* with each other. To do so, an enclave (enc_i) first runs HW.RunReport which produces a report ($\mathsf{rpt_i}$) that is sent to enc_j. Upon reception, enc_j executes HW.ReportVerify and verifies the validity of $\mathsf{rpt_i}$. In the case where enc_i and enc_j run in different hosts, instead of running HW.RunReport and HW.ReportVerify they run HW.RunQuote and HW.QuoteVerify. A more detailed description of the hardware algorithms used by the enclaves follows:

- **HW.Setup**(1^λ): Takes as input a security parameter λ and produces the secret key $\mathsf{sk}_{\mathsf{rpt}}$ used to MAC the reports.
- **HW.Load**(Q): Takes as input a program Q. An enclave enc_i is created in which Q will be loaded. Moreover a handle $\mathsf{hdl}_{\mathsf{enc}}$ is created that will be used as an identifier for the enclave.
- **HW.Run**($\mathbf{hdl}, in$): Takes as input a handle hdl and some input in. It runs the program in the enclave specified by hdl with in as input.
- **HW.Run&Report**($\mathbf{hdl}, in$): Takes as input a handle hdl and some input in. It will output a report that is verifiable by any other enclave on the same platform. The report contains information about the underlying enclave signed with $\mathsf{sk}_{\mathsf{rpt}}$.
- **HW.ReportVerifiy**($\mathbf{hdl}', \mathsf{rpt}$): Takes as input a handle hdl' and a report rpt. Uses $\mathsf{sk}_{\mathsf{rpt}}$ generated by $\mathsf{HW.Setup}$ to verify the MAC of the report.

5.1 Formal Construction

MF is divided into a *Setup phase* and four main phases; *Initialization, Key Sharing, Editing* and *Revocation*. During the *Setup phase*, all the necessary enclaves are initialized by running the $\mathsf{MF.Setup}$ algorithm. In the rest of the phases, the user is interacting with the enclaves by running one of the following algorithms: $\mathsf{MF.ABEUserKey}$, $\mathsf{MF.Store}$, $\mathsf{MF.KTStore}$, $\mathsf{MF.KeyShare}$, $\mathsf{MF.Search}$, $\mathsf{MF.Update}$, $\mathsf{MF.Delete}$ and $\mathsf{MF.Revoke}$ as described below.

Setup Phase: In this phase $\mathsf{MF.Setup}$ runs. Each entity receives a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ for a CCA2 secure public cryptosystem PKE. In addition to that, the entities running in enclaves generate a signing and a verification key pair. Finally, MS runs $\mathsf{CPABE.Setup}$ to acquire the master public/private key pair $(\mathsf{MPK}, \mathsf{MSK})$. An enclave is initialized as follows:

$\mathsf{MF.Setup}(\text{“}initialize\text{”}, 1^\lambda)$: Each enclave is initialized by generating a public/private and signing/verification key pairs. To do so, the program $\mathbf{Q}^{\mathbf{init}}_{\mathbf{ID}}$ is loaded:

$\mathbf{Q}^{\mathbf{init}}_{\mathbf{ID}}$

- On input $(\text{“}initialize\text{”}, 1^\lambda)$:
  1. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
  2. Output pk.

  Run $\mathsf{hdl} \leftarrow \mathsf{HW.Load}(\mathsf{Q}^{\mathsf{init}}_{\mathsf{ID}})$.

Additionally, during the setup phase, the MS enclave loads a program $\mathbf{Q}^{\mathbf{Setup}}_{\mathbf{MS}}$ that outputs the master public/private key pair $(\mathsf{MPK}, \mathsf{MSK})$:

$\mathbf{Q}^{\mathbf{Setup}}_{\mathbf{MS}}$

- On input $(\text{“}initialize\text{”}, 1^\lambda)$:
  1. Run $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
  2. Output MPK.

  Run $\mathsf{hdl}_{\mathsf{MS}} \leftarrow \mathsf{HW.Load}(\mathsf{Q}^{\mathsf{Setup}}_{\mathsf{MS}})$.

Initialization Phase: As a first step , a user u_i contacts the MS enclave and requests a secret CP-ABE key (Figure 1a). Upon reception, MS authenticates u_i

and checks if the user is eligible for receiving such a key (i.e. is not a compromised user, has not generated such a key in the past, etc.). If so, MS generates a CP-ABE key $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}$, encrypts it under $\mathsf{pk_i}$ and sends it back to u_i. This is done by running the program $\mathbf{Q}_{\mathbf{MS}}^{\mathbf{SKey}}$ in the MS enclave as shown below:

$\mathsf{MF.ABEUserKey}(\text{"}KeyRequest\text{"}, \mathsf{MSK}, u_i, cred_i, \mathcal{A})$: The master enclave program $\mathbf{Q}_{\mathbf{MS}}^{\mathbf{SKey}}$ for generating users' ABE keys is defined as follows:

$\mathbf{Q}_{\mathbf{MS}}^{\mathbf{SKey}}$

- On input ("$KeyRequest$", $\mathsf{MSK}, u_i, cred_i, \mathcal{A}$):
  1. Verify that u_i is registered. If not, output $\perp$.
  2. Use MSK and compute $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}$.
  3. Compute and output $c = \mathsf{PKE.Enc}(\mathsf{pk_i}, \mathsf{sk}_{\mathcal{A},\mathsf{u_i}})$.

  Run $c \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{MS}}, (\text{"KeyRequest"}, \mathsf{MSK}, \mathsf{i}, \mathsf{cred_i}, \mathcal{A}))$.

After u_i successfully received $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}$ she can start using the CSP to store files remotely. To do so, she first sends her credentials $cred_i$ and a store request $StoreReq$ to the CSP. Specifically, u_i sends $m_{req} = \langle r_1, \mathsf{E}_{\mathsf{pk_{CSP}}}(cred_i), StoreReq, H(r_1||cred_i||StoreReq)\rangle$ where r_i is a random number gererated by u_i. The CSP authenticates u_i as a legitimate user and sends back an authorization $Auth$ as $m_{ver} = \langle r_2, (Auth), \sigma_{CSP}(H(r_2||u_i||Auth))\rangle$. At this point, u_i generates a symmetric key $\mathsf{K_i}$ to encrypt her files and sends $m_{store} = \langle r_3, \mathsf{E}_{\mathsf{pk_{CSP}}}(\gamma_i), \mathbf{c_i}, H(r_3||\gamma_i||c_i)\rangle$ to the CSP.

$\mathsf{MF.Store}(\text{"}Store\text{"}, m_{req})$: The CSP enclave program $\mathbf{Q}_{\mathbf{CSP}}^{\mathbf{Store}}$ that is responsible for storing encrypted files is defined as follows:

$\mathbf{Q}_{\mathbf{CSP}}^{\mathbf{Store}}$

- On input ("$StoreReq$", m_{req}):
  1. Open m_{req}; verify the message[a]; if the verification fails, output $\perp$.
  2. Compute $m_{ver} = \langle r_2, (Auth), \sigma_{CSP}(H(r_2||u_i||Auth))\rangle$.
  3. Output m_{ver}.

  Run $m_{ver} \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{CSP}}, (\text{"}StoreReq\text{"}, m_4))$.
- On input ("$store$", m_{store}):
  1. Open m_{store}; verify the message; if the verification fails, output $\perp$.
  2. Store (c_i, γ_i).

  Run $\mathsf{HW.Run}(\mathsf{hdl_{CSP}}, (\text{"}store\text{"}, m_{store}))$.

[a] By this, we mean that the entity receiving the message verifies the freshness and the integrity of the message and it can also authenticate the sender.

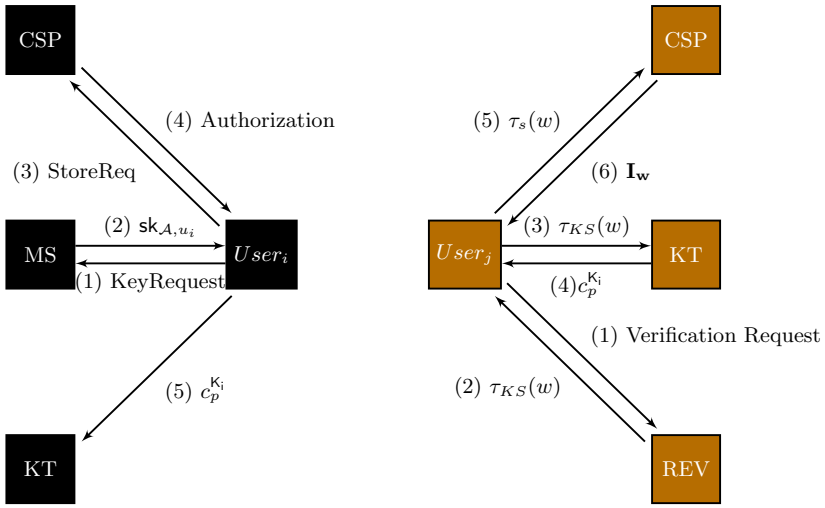
Initialization phase concludes with $\mathsf{MF.KTStore}$ where u_i encrypts $\mathsf{K_i}$ under MPK to get $c_P^{\mathsf{K_i}}$ and sends $m_{keystore} = \left\langle \mathsf{E}_{\mathsf{pk_{KT}}}(r_4), c_P^{\mathsf{K_i}}, \sigma_i\left(H\left(r_4||c_P^{\mathsf{K_i}}\right)\right)\right\rangle$ to KT. Upon reception, KT generates a random number $r_{\mathsf{K_i}}$ that is stored next to $c_P^{\mathsf{K_i}}$.

$\mathsf{MF.KTStore}(\text{"}store\text{"}, m_{keystore})$: The KT enclave program $\mathbf{Q}_{\mathbf{KT}}^{\mathbf{Store}}$ that stores a symmetric key $\mathsf{K_i}$ encrypted with MPK is defined as follows:

$\mathbf{Q}_{\mathbf{KT}}^{\mathbf{Store}}$

- On input $(\text{“}store\text{”}, m_{keystore})$:
  1. Open $m_{keystore}$; verify the message. If the verification fails, output $\perp$.
  2. Generate a random number $r_{\mathsf{K_i}}$.
  3. Compute $c = \mathsf{PKE.Enc}(\mathsf{pk_{u_i}}, \mathsf{r_{K_i}})$.
  4. Store $\left(\mathsf{c_p^{K_i}}, c\right)$.

  Run $\left(\mathsf{c_p^{K_i}}, c\right) \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{“}store\text{”}, m_{keystore}))$.



(a) MF Initialization (b) Key Sharing and Search

Fig. 1: MF Initialization and Key Sharing

Key Sharing Phase: The goal of this phase (Figure 1b) is to share data between legitimate users. We assume that the *Initialization Phase* has been successfully completed (i.e. u_i has stored encrypted files in the CSP). This phase commences with a user u_j wishing to access files stored by u_i. At first, u_j has to be verified by REV as a legitimate user. If the verification is successful, u_j receives a token τ_{KS} that forwards to KT. By doing this, KT is convinced that u_j is legitimate and replies by sending back $c_p^{\mathsf{K_i}}$. The user will be able to decrypt $c_p^{\mathsf{K_i}}$ if and only if her attributes match the policy bounded to $c_p^{\mathsf{K_i}}$. In more detail, the *Key Sharing Phase* begins with u_j executing $\mathsf{MF.KeyShare}$ to prove that is not revoked. To this end, u_j sends $m_{verReq} = \langle r_5, \mathsf{E}_{\mathsf{pk_{REV}}}(u_j), \sigma_j(r_5||u_j)\rangle$ to REV. Upon reception, REV verifies the message and checks whether $u_j \in rl$ or not. Assuming that $u_j \notin rl$ (i.e. she has not been revoked), REV replies with $m_{token} = \langle r_6, \mathsf{E}_{\mathsf{pk_{KT}}}(u_j), \mathsf{E}_{\mathsf{pk_{KT}}}(\tau_{KS}), \sigma_{REV}\left(H(r_6||u_j||\tau_{KS})\right)\rangle$. The user then simply forwards m_{token} to KT who verifies it. After the verification is complete KT sends $m_{key} = \left\langle (\mathsf{E}_{\mathsf{pk_{CSP}}}(u_j, t)), c_p^{\mathsf{K_i}}, \sigma_{KT}\left(H(u_j||t)\right)\right\rangle$ back to u_j, where t is a

timestamp declaring the time that u_j accessed $c_p^{\mathsf{K_i}}$. If u_j already received $\mathsf{K_i}$ in the past, KT will only send back the first and last components of m_{key}.
$\mathsf{MF.KeyShare}(\text{“}share\text{”}, m_4)$: REV and KT enclave programs ($\mathbf{Q_{REV}^{Ver}}$, $\mathbf{Q_{KT}^{Share}}$) that are responsible for sharing $c_p^{\mathsf{K_i}}$ are defined as follows:

$\mathbf{Q_{REV}^{Ver}}$

- On input $(\text{“}share\text{”}, m_{verReq})$:
  1. Open m_5; verify the message; if the verification fails, output $\perp$.
  2. Check if $u_j \in rl$; if so, output $\perp$.
  3. Generate τ_{KS}.
  4. Compute and output m_{token}.

  Run $m_{token} \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{REV}}, (\text{“}share\text{”}, m_{verReq}))$.

$\mathbf{Q_{KT}^{share}}$

- On input $(\text{“}share\text{”}, m_{token})$:
  1. Open m_{token}; verify the message; if the verification fails, output $\perp$.
  2. Decrypt $\mathsf{PKE.Enc}(\mathsf{pk_{KT}}, u_j)$ and $\mathsf{PKE.Enc}(\mathsf{pk_{KT}}, \tau_{KS})$.
  3. Compute and output m_{key}.

  Run $m_{key} \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{“}share\text{”}, m_{ver}))$.

Now that $\mathsf{MF.KeyShare}$ has been completed, u_j can run $\mathsf{MF.Search}$ to access certain files that are stored in the CSP. To do so, she locally runs $\mathsf{SSE.SearchToken}$ to generate $\tau_s(w)$ and then sends $m_{search} = \langle \mathsf{E}_{\mathsf{pk_{CSP}}}(u_j, t, \tau_s(w)), \sigma_i(H(u_j||t||\tau_s(w))), \sigma_{KT}(H(u_j||t)) \rangle$ to the CSP[1]. Upon reception, CSP verifies the message and the timestamp. Assuming that the message is verified, CSP runs $\mathsf{SSE.Search}$ with $\tau_s(w)$ as input. The output $\mathbf{I_w}$ is sent back to u_j.
$\mathsf{MF.Search}(\text{“}search\text{”}, m_{search},)$: The CSP enclave program $\mathbf{Q_{CSP}^{Search}}$ that is responsible for searching over the encrypted data is defined as follows:

$\mathbf{Q_{CSP}^{Search}}$

- On input $(\text{“}search\text{”}, m_{search})$:
  1. Open m_{search}; verify the message; if the verification fails, output $\perp$.
  2. Run $\mathsf{SSE.Search}(\gamma_i, c_i, \tau_s(w)) \rightarrow \mathbf{I_w}$
  3. Output $\mathbf{I}_w$.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{CSP}}, (\text{“}search\text{”}, m_{search})$, which internally runs $\mathsf{SSE.Search} \rightarrow \mathbf{I}_w$.

Editing Phase: In this phase[2], registered users can add files to the database and data owners can also delete files. To do so, u_i executes $\mathsf{MF.Update}$ and $\mathsf{MF.Delete}$. To update the database, u_i first generates an add token by running $(\tau_a(f), c_f) \leftarrow \mathsf{SSE.AddToken}(\mathsf{K_i}, f)$. This token is sent to the CSP via $m_{add} = \langle \mathsf{E}_{\mathsf{pk_{CSP}}}(u_i, t, \tau_\alpha(f), c_i, \gamma_i), \sigma_i\left(H(u_i||t||\tau_{\alpha(f)}||c_i||\gamma_i)\right), \sigma_{KT}(u_i||t) \rangle$. Finally, the CSP verifies the message and its freshness and executes $\mathsf{SSE.Add}(\gamma_i, c_i, \tau_\alpha(f), c_f) \rightarrow (\gamma_i', c_i')$.
$\mathsf{MF.Update}(\text{“}update\text{”}, m_{add})$: The CSP enclave program $\mathbf{Q_{CSP}^{Up}}$ for adding files to the database is defined as follows:

[1] The user simply forwards the components of m_{key} to the CSP along with a search token $\tau_s(w)$.

[2] One could completely ignore the *Editing Phase* and the result would be a static MF.

$\mathbf{Q}^{\mathbf{Up}}_{\mathbf{CSP}}$

- On input ("*update*", m_{add}):
  1. Verify the message. If the verification fails, output $\perp$.
  2. Run $\mathsf{SSE.Add}(\gamma_i, c_i, \tau_\alpha(f), c_f) \rightarrow (\gamma_i', c_i')$.

  Run $\mathsf{HW.Run}(\mathsf{hld}_{\mathsf{CSP}}, (\text{"}update\text{"}, m_{add})$, which internally runs $\mathsf{SSE.Add}(\gamma_i, c_i, \tau_\alpha(f), c_f) \rightarrow (\gamma_i', c_i')$.

Deletion of a file is a more complicated task. This is due to the fact that we only allow the data owner to delete files. To achieve this, u_i needs to to prove her ownership over $\mathsf{K_i}$. This can be done by requesting the random number $r_{\mathsf{K_i}}$ from KT. After u_i receives $r_{\mathsf{K_i}}$, she signs it, runs $\tau_d \leftarrow \mathsf{SSE.DeleteToken}(\mathsf{K_i}, \mathsf{f})$ and replies to KT with: $m_{delete} = \langle \mathsf{E}_{\mathsf{pk_{CSP}}}(u_i, t, \tau_d(f), \gamma_i'), \sigma_i(H(u_i||\tau_d(f)||\gamma_i'||r_{\mathsf{K_i}}\rangle$. KT verifies the message and is convinced that u_i is the owner of $\mathsf{K_i}$. Finally, KT generates a report (rpt) containing the delete token. This is sent to the CSP who proceeds with the deletion of the specified files.

$\mathsf{MF.Delete}(\text{"}request\text{"}, \sigma_i(u_i||t), c_p^{\mathsf{K_i}})$: The enclave programs $\mathbf{Q}^{\mathbf{Del}}_{\mathbf{CSP}}, \mathbf{Q}^{\mathbf{Del}}_{\mathbf{KT}}$ that are responsible for deleting files from the database are defined as follows:

$\mathbf{Q}^{\mathbf{Del}}_{\mathbf{KT}}$

- On input ("*request*", $\sigma_i(u_i||t), c_p^{\mathsf{K_i}}$):
  1. Verify the signature. If the verification fails, output $\perp$.
  2. Get $r_{\mathsf{K_i}}$ and compute $c = \mathsf{PKE.Enc}(\mathsf{pk_{u_i}}, r_{\mathsf{K_i}})$.
  3. Output c.

  Run $c \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{"}request\text{"}, \sigma_i(u_i||t), c_p^{\mathsf{K_i}})$.
- On input ("*delete*", m_{delete}):
  1. Open m_{delete}; verify the message and authenticate u_i as the owner of $\mathsf{K_i}$. If the verification or the authentication fail, output $\perp$.
  2. Generate and output rpt.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{"}delete\text{"}, m_{delete})$ and then $\mathsf{rpt} \leftarrow \mathsf{HW.RunReport}(\mathsf{hdl_{KT}}, (\text{"}delete\text{"}, m_{delete}))$.

$\mathbf{Q}^{\mathbf{Del}}_{\mathbf{CSP}}$

- On input ("*delete*", rpt):
  1. Verify rpt. If the verification fails, output $\perp$.
  2. Run $\mathsf{SSE.Delete}(\gamma_i', c_i', \tau_d(f)) \rightarrow (\gamma_i'', c_i'')$.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{CSP}}, (\text{"}delete\text{"}, \mathsf{rpt})$ who will internally run $\mathsf{HW.ReportVerify}$ $(\mathsf{hdl_{CSP}}, \mathsf{rpt})$ and $\mathsf{SSE.Delete}(\gamma_i, c_i, \tau_d(f)) \rightarrow (\gamma_i'', c_i'')$.

Revocation Phase: The last phase of MF focuses on the revocation of users. More precisely, we consider the scenario where a data owner u_i wishes to revoke access to a user u_j to whom she had granted permission to access certain files in the past. We consider two different scenarios and propose two slightly different solutions for each one. The first scenario (Figure 2a) is best suited for organizations that wish to store large volumes of data online. The problem is the same as in $\mathsf{MF.Delete}$; u_i needs to prove her ownership over $\mathsf{K_i}$. To do so, u_i executes the same steps as in the *Editing phase*. After u_i is authenticated as the owner of $\mathsf{K_i}$, KT sends an acknowledgement to REV containing the identity of u_j. REV then adds u_j to the revocation list rl. To successfully run $\mathsf{MF.Revoke}$, u_i first needs to

prove ownership over $\mathsf{K_i}$ by following the same steps as in MF.Delete. When u_i signs $r_{\mathsf{K_i}}$, she sends $m_{revoke} = \left\langle r_{10}, \mathsf{E}_{\mathsf{pk_{KT}}}\left(u_i, u_j, c_P^{\mathsf{K_i}}\right), \sigma_i\left(H(u_i||u_j||c_P^{\mathsf{K_i}}||r_{\mathsf{K_i}}\right)\right\rangle$ to KT. Now that KT is convinced that u_i is the owner of $\mathsf{K_i}$, it generates rpt containing u_j's identity, which is then sent to REV, who adds u_j to rl.

MF.Revoke$(\text{“}request\text{”}, \sigma_i(u_i||t), c_p^{\mathsf{K_i}})$: The enclave programs $\mathbf{Q_{KT}^{Rev}}, \mathbf{Q_{REV}^{Rev}}$ that are responsible for revoking users are defined as follows:

$\mathbf{Q_{KT}^{Rev}}$

- On input $(\text{“}request\text{”}, \sigma_i(u_i||t), c_p^{\mathsf{K_i}})$:
  1. Verify the signature. If the verification fails, output $\perp$.
  2. Get $r_{\mathsf{K_i}}$ and compute $c = \mathsf{PKE.Enc}(\mathsf{pk_{u_i}}, r_{\mathsf{K_i}})$.
  3. Output c.

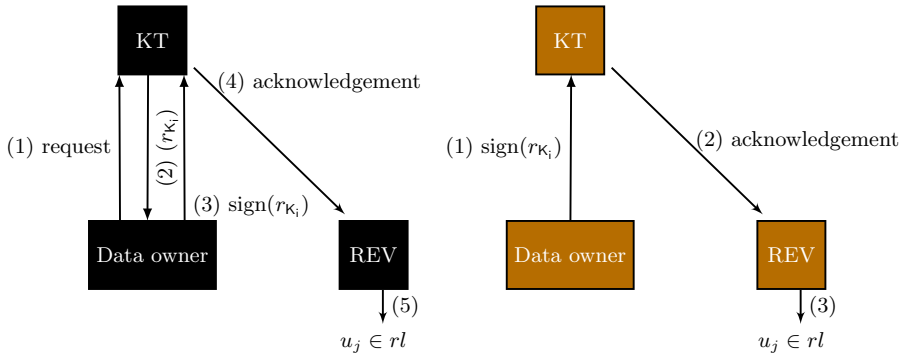
  Run $c \leftarrow \mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{“}request\text{”}, u_i, c_p^{\mathsf{K_i}}))$.
- On input $(\text{“}revoke\text{”}, m_{revoke})$:
  1. Open m_{revoke}; verify the message and authenticate u_i as the owner of $\mathsf{K_i}$. If the verification or the authentication fails, output $\perp$.
  2. Generate $r_{\mathsf{K_i}'}$ and replace it with $r_{\mathsf{K_i}}$.
  3. Generate and output rpt.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{“}revoke\text{”}, m_{revoke})$ and then $\mathsf{rpt} \leftarrow \mathsf{HW.RunReport}(\mathsf{hdl_{KT}}, (\text{“}revoke\text{”}, m_{report}))$.

$\mathbf{Q_{REV}^{Rev}}$

- On input $(\text{“}revoke\text{”}, \mathsf{rpt})$:
  1. Veirfy rpt. If the verification fails, output $\perp$.
  2. Add u_j to the revocation list rl.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{REV}}, (\text{“}revoke\text{”}, \mathsf{rpt})$ who will internally run HW.Report Verify $(\mathsf{hdl_{REV}}, report)$.



(a) Revocation Phase (b) Revocation Phase: Simplified

Fig. 2: MF Revocation

The second scenario (Figure 2b) refers to typical cloud users which mainly use such services for storing files. In this case, we slightly modify the protocol

in order to simplify it. To achieve better efficiency, we assume that when the data owner u_i executes $\mathsf{MF.KTStore}$ the random number $r_{\mathsf{K_i}}$ generated by KT, is now sent directly back to her. This way u_i will not have to request it from the KT every time she wishes to revoke a user. The rest of the protocol remains the same as in the first scenario. In particular:
$\mathsf{MF.Revoke}(\text{“}revoke\text{”}, m_{revoke})$: The enclave programs $\mathbf{Q_{KT}^{Rev^*}}$, $\mathbf{Q_{REV}^{Rev^*}}$ that are responsible for the revocation of users are defined as follows:

$\mathbf{Q_{KT}^{Rev^*}}$

- On input $(\text{“}revoke\text{”}, m_{revoke})$:
  1. Open m_{revoke}; verify the message and authenticate u_i as the owner of $\mathsf{K_i}$. If the verification or the authentication fails, output $\perp$.
  2. Generate $r_{\mathsf{K_i}'}$ and replace it with $r_{\mathsf{K_i}}$.
  3. Generate and output rpt.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{KT}}, (\text{“}revoke\text{”}, m_{revoke})$ and then
  $\mathsf{rpt} \leftarrow \mathsf{HW.Run\&Report}(\mathsf{hdl_{KT}}, (\text{“}revoke\text{”}, m_{revoke}))$.

$\mathbf{Q_{REV}^{Rev^*}}$

- On input $(\text{“}revoke\text{”}, \mathsf{rpt})$:
  1. Veirfy rpt. If the verification fails, output $\perp$.
  2. Add u_j to the revocation list rl.

  Run $\mathsf{HW.Run}(\mathsf{hdl_{REV}}, (\text{“}revoke\text{”}, \mathsf{rpt})$ who will internally run $\mathsf{HW.Report Verify}(\mathsf{hdl_{REV}}, \mathsf{rpt})$.

6 Security Analysis

To prove the security of our construction, we assume the existence of a simulator $\mathcal{S}$. The main purpose of $\mathcal{S}$ is to simulate the algorithms of the real protocol in such a way that any polynomial time adversary $\mathcal{ADV}$ will not be able to distinguish between the real protocol and $\mathcal{S}$. We assume that $\mathcal{S}$ intercepts $\mathcal{ADV}$'s communication with the real protocol and replies with simulated outputs. Before we proceed with the proof, we define the capabilities of $\mathcal{S}$ and $\mathcal{ADV}$.

1. Everything $\mathcal{ADV}$'s observes in the real experiment can be simulated by $\mathcal{S}$.
2. $\mathcal{ADV}$ intercepts all communication between different entities. Since we use an IND-CCA2 public key encryption scheme, if $\mathcal{ADV}$ can distinguish between real and simulated answers, then she can also break the IND-CCA2 security.
3. $\mathcal{ADV}$ can load different programs in the enclaves and record the output. This assumption significantly strengthens $\mathcal{ADV}$ since we need to ensure that only honest attested programs will be executed in the enclaves.

Finally, to prove the security of $\mathsf{MF.Delete}$ we give $\mathcal{ADV}$ the role of data owner. However, $\mathcal{ADV}$ can never perform a direct attack to our construction by impersonating the data owner. The only way for a user to prove that she is the data owner is to retrieve $r_{\mathsf{K_i}}$ from KT, sign it, and send it back. However, this is equivalent with forging the data owner's signature, which given the EUM-CFA security of the signature scheme can only happen with negligible probability.

Definition 4. *(Sim-Security). We consider the following experiments. In the real experiment, all algorithms run as defined in our construction. In the ideal experiment, a simulator $\mathcal{S}$ intercepts $\mathcal{ADV}$'s queries and replies with simulated responses.*

Real Experiment

1. $\mathbf{EXP}^{real}_{MF}(1^\lambda)$:
2. $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{MF.Setup}(1^\lambda)$
3. $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}} \leftarrow \mathcal{ADV}^{\mathsf{MF.ABEUserKey}(\mathsf{MSK},\cdot\mathcal{A})}$
4. $ct \leftarrow \mathsf{CPABE.Enc}(\mathsf{mpk}, m)$
5. $(\gamma, c) \leftarrow \mathcal{ADV}^{\mathsf{SSE.Enc}(\mathsf{K},\mathbf{f})}$
6. $\mathsf{MF.Search}(\text{“}search\text{”}, m_s) \rightarrow \mathbf{I_w}$
7. $\mathsf{MF.Update}(\text{“}update\text{”}, m_{add}) \rightarrow (\gamma', c')$
8. $\mathsf{MF.Delete}(\text{“}delete\text{”}, m_{delete}) \rightarrow (\gamma', c')$
9. *Output b*

Ideal Experiment

1. $\mathbf{EXP}^{ideal}_{MF}(1^\lambda)$:
2. $(\mathsf{MPK}) \leftarrow \mathcal{S}(1^\lambda)$
3. $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}} \leftarrow \mathcal{ADV}^{\mathcal{S}(1^\lambda)}$
4. $ct \leftarrow \mathcal{S}(1^\lambda, 1^{|m|})$
5. $(\gamma, c) \leftarrow \mathcal{ADV}^{\mathcal{S}(\mathcal{L}_{in}(\mathbf{f}))}$
6. $\mathcal{S}(\text{“}search\text{”}, m_s) \rightarrow \mathbf{I_w}$
7. $\mathcal{S}(\text{“}update\text{”}, m_{add}) \rightarrow (\gamma', c')$
8. $\mathcal{S}(\text{“}delete\text{”}, m_{delete}) \rightarrow (\gamma', c')$
9. *Output b′*

We say that MF is sim-secure if for all PPT adversaries $\mathcal{ADV}$:

$$\mathbf{EXP}^{real}_{MF}(1^\lambda) \approx \mathbf{EXP}^{ideal}_{MF}(1^\lambda)$$

At a high-level, we will construct a simulator $\mathcal{S}$ that will replace all the MF algorithms. $\mathcal{S}$ can simulate HW, Key generation and encryption oracles. $\mathcal{S}$ is given the length of the challenge message as well as the leakage functions $\mathcal{L}_{in}$. In the real experiment, the challenger $\mathcal{C}$ runs $\mathsf{K_i} \leftarrow \mathsf{SSE.Gen}(1^\lambda)$ and replies to $\mathcal{ADV}$ in accordance to definition 3. $\mathsf{K_i}$ is not given to $\mathcal{ADV}$, since possession of $\mathsf{K_i}$ implies that $\mathcal{ADV}$ can win the game. $\mathcal{ADV}$ queries $\mathcal{C}$ for an index/ciphertext pair (γ, c) based on a file $\mathbf{f}$. In the real experiment, (γ, c) is generated using $\mathsf{K_i}$. In the ideal one, $\mathcal{S}$ gets as input $\mathcal{L}_{in}(\mathbf{f})$ and outputs a simulated response. $\mathcal{S}$ simulates the $\mathsf{MF.Search}, \mathsf{MF.Update}$ and $\mathsf{MF.Delete}$ oracles by getting as input the simulated tokens of the SSE security game. In our security game, we exclude $\mathsf{MF.Revoke}$ since rl is not retrievable during the execution of the protocol. Also, since rl is stored in plaintext and its values does not depend on sensitive data, side channel attacks on SGX will not reveal any private information. However, for purposes of completeness, we include the revocation oracle in our proof.

Theorem 1. *Assuming that* PKE *is an IND-CCA2 secure public key cryptosystem and* Sign *is an EUF-CMA secure signature scheme then MF is a sim-secure protocol according to Definition 4.*

Proof. We start by defining the algorithms used by the simulator. Then, we will replace them with the real algorithms. Finally, the help of a Hybrid Argument we will prove that the two distributions are indistinguishable.

– $\mathsf{MF.Setup}^*$: Will only generate MPK that will be given to $\mathcal{ADV}$.

- $\mathsf{MF.ABEUserKey}^*$: Will generate a random key to be sent to the adversary. That is, when $\mathcal{ADV}$ makes a key generation query, $\mathcal{S}$ will simulate $\mathsf{CPABE.KeyGen}$ and it will output $\mathsf{sk}^*_{\mathsf{A,u_i}}$. This key is a random string that has the same length as the output of the real $\mathsf{MF.ABEUserKey}^*$. The key will be given to $\mathcal{ADV}$.
- $\mathsf{MF.KeyShare}^*$: In the ideal experiment, after $\mathcal{ADV}$ requests a secret key, $\mathcal{S}$ will encrypt a sequence of bits based on $\mathcal{L}_{in}$, under MPK. The ciphertext will be returned to $\mathcal{ADV}$.
- $\mathsf{MF.Search}^*$: When $\mathcal{ADV}$ generates a search token $\tau_s(w)$, $\mathcal{S}$ gets as input the leakage function $\mathcal{L}_s$ and outputs a simulated response. When $\mathcal{ADV}$ makes a search query, $\mathcal{S}$ will once again generate a simulated $\mathbf{I}^*_{\mathbf{w}}$ which will be sent back to her.
- $\mathsf{MF.Update}^*$: When $\mathcal{ADV}$ generates an add token $\tau_\alpha(f)$, $\mathcal{S}$ gets as input the leakage function $\mathcal{L}_a$ and outputs a simulated response. $\mathcal{S}$ will simulate the add token, the ciphertext to be added to the database and will also update the encrypted index.
- $\mathsf{MF.Delete}^*$: When $\mathcal{S}$ generates a delete token, $\mathcal{S}$ gets as input the leakage function $\mathcal{L}_d$ and outputs a simulated response. Apart from $\tau_d(f)$, $\mathcal{S}$ will also update the encrypted index.
- $\mathsf{MF.Revoke}^*$: In contrast to the real experiment, the system does not revoke any user.

In the pre-processing phase, $\mathcal{S}$ runs $\mathsf{HW.Setup}(1^\lambda)$, just as in the real experiment, in order to acquire $\mathsf{sk_{rpt}}$. Moreover, the challenger $\mathcal{C}$ generates a symmetric key $\mathsf{K_i}$, that will be needed in order to reply to search, add and delete queries. We will now use a hybrid argument to prove that $\mathcal{ADV}$ cannot distinguish between the real and the ideal experiments.

Hybrid 0 MF runs normally.

Hybrid 1 Everything runs like in Hybrid 0, but we replace $\mathsf{MF.Setup}$ with $\mathsf{MF.Setup}^*$.

These algorithms are identical from $\mathcal{ADV}$'s perspective and as a result the hybrids are indistinguishable.

Hybrid 2 Everything runs like in Hybrid 1, but $\mathsf{MF.ABEUserKey}^*$ runs instead of $\mathsf{MF.ABEUserKey}$.

Hybrid 2 is indistinguishable from Hybrid 1 because nothing changes from $\mathcal{ADV}$'s point of view.

After Hybrid 2, we have ensured that $\mathcal{ADV}$ has followed all the required steps in order to ask for $\mathsf{K_i}$. We are now ready to replace $\mathsf{MF.KeyShare}$ with $\mathsf{MF.KeyShare}^*$.

Hybrid 3 Like Hybrid 2, but $\mathsf{MF.KeyShare}^*$ runs instead of $\mathsf{MF.KeyShare}$. Also, the algorithm outputs $\perp$ if $\mathsf{HW.Run}$ is queried with $(\mathsf{hdl_{KT}}, (\text{"}share\text{"}, \ m_{token}))$ but $\mathcal{ADV}$ never contacts REV.

Lemma 1. *Hybrid 3 is indistinguishable from Hybrid 2.*

Proof. By replacing the two algorithms, nothing changes from $\mathcal{ADV}$'s point of view. Moreover if $\mathcal{ADV}$ can generate m_{token}, then she can forge REV's signature. Given the security of the signature scheme, this can only happen with negligible probability. So $\mathcal{ADV}$ can only distinguish between Hybrid 3 and Hybrid 2 with negligible probability. □

At this point, $\mathcal{ADV}$ has received what she thinks is a valid $\mathsf{K_i}$. However, $\mathcal{S}$ sent her an encryption of a random string of the same length as $\mathsf{K_i}$. The last part of the proof concerns the SSE phase of MF. For the rest of the proof we assume that $\mathcal{ADV}$ performs search, add and delete queries in order to get back $\mathbf{I_w}$. We will replace the real algorithms MF.Search, MF.Update and MF.Delete with the simulator. The simulator now gets access to all leakage functions $\mathcal{L}$ from the SSE scheme.

Hybrid 4 Like Hybrid 3, but when HW.Run is queried with $(\mathsf{hdl_{CSP}}, (\text{"}search\text{"}, m_{search}))$, $\mathcal{S}$ is given the leakage function $\mathcal{L}_{\mathcal{S}}$ and generates $\mathbf{I}^*_{\mathbf{w}}$ which is then sent to the user.

Lemma 2. *Hybrid 4 is indistinguishable from Hybrid 3.*

Proof. Assuming the $\mathcal{L}_i-$ security of the SSE scheme, the token sent by $\mathcal{ADV}$ to the CSP, as part of m_{search}, is generated by $\mathcal{S}$ with $\mathcal{L}_s$ as input. As a result when the CSP receives m_{search}, it will generate a sequence of file identifiers $\mathbf{I}^*_{\mathbf{w}}$ that will be send back to $\mathcal{ADV}$. $\mathcal{ADV}$ cannot distinguish between the real and the ideal experiment since she receives a sequence of files corresponding to a search token that was also simulated by $\mathcal{S}$ given $\mathcal{L}_s$ as input. Moreover, if $\mathcal{ADV}$ manages to generate m_{search} without having contacted KT earlier, then she can also forge KT's signature. However, this can only happen with negligible probability, and as a result $\mathcal{ADV}$ can only distinguish between hybrids 4 and 3 with negligible probability. □

Hybrid 5 Like Hybrid 4, but when HW.Run is queried with $(\mathsf{hdl_{CSP}}, (\text{"}update\text{"}, m_{add}))$, $\mathcal{S}$ is given the leakage function $\mathcal{L}_a$ and tricks $\mathcal{ADV}$ into thinking that she updated the database.

Lemma 3. *Hybrid 5 is indistinguishable form Hybrid 4.*

Proof. The proof is similar to the previous one but simpler since $\mathcal{ADV}$ does not expect an output from this algorithm. So, by assuming the $\mathcal{L}_i-$ security of the SSE scheme, we know that $\mathcal{ADV}$ will not be able to distinguish between the real add token and the simulated one. Moreover, similar to the previous Hybrid, if $\mathcal{ADV}$ can generate m_{add} without having contacted KT, then she can also forge KT's signature – which can only happen with negligible probability. Hence, $\mathcal{ADV}$ can only distinguish between hybrids 5 and 4 with negligible probability. □

Hybrid 6 Like Hybrid 5, but when HW.Run is queried with $(\mathsf{hdl_{KT}}, (\text{"}delete\text{"}, m_{del}))$, $\mathcal{S}$ is given the leakage function $\mathcal{L}_d$ and tricks $\mathcal{ADV}$ into thinking that she deleted a certain file from the database. Moreover, $\mathcal{S}$ outputs $\perp$, if ReportVerify is queried with $(\mathsf{hdl_{CSP}}, \mathsf{rpt})$ for a report that was not generated by executing $\mathsf{HW.RunReport}(\mathsf{hdl_{KT}}, (\text{"}delete\text{"}, m_{delete}))$.

Proof. By assuming the $\mathcal{L}_i-$ security of the SSE scheme, we know that $\mathcal{ADV}$ will not be able to distinguish between the real delete token and the simulated one. Moreover, if $\mathcal{ADV}$ can query $\mathsf{HW.ReportVerify}$ with $(\mathsf{hdl}_{\mathsf{CSP}}, \mathsf{rpt})$, for a rpt that was not generated by KT, then $\mathcal{ADV}$ can produce a valid MAC which can only happen with negligible probability since she does not know $\mathsf{sk}_{\mathsf{rpt}}$. Thus, $\mathcal{ADV}$ can only distinguish between Hybrids 5 and 6 with negligible probability. □

Hybrid 7 Like Hybrid 6 but instead of $\mathsf{MF.Revoke}$, $\mathcal{S}$ executes $\mathsf{MF.Revoke}^*$.

The hybrids are indistinguishable since no one can access the content of the revocation list and as a result nothing changes from $\mathcal{ADV}$'s point of view.

With this Hybrid our proof is complete. We managed to replace the expected outputs with simulated responses in a way that $\mathcal{ADV}$ cannot distinguish between the real and the ideal experiment. □

6.1 SGX Security

Recent works [3, 20, 8, 19] have shown that SGX is vulnerable to software attacks. However, according to [5], these attacks can be prevented if the programs running in the enclaves are data-obvious. Thus, leakage can be avoided if the programs do not have memory access patterns or control flow branches that depend on the values of sensitive data. In our construction, no sensitive data (such us decryption keys) are used by the enclaves. KT acts as a storage space for the symmetric keys and does not perform any computation on them. Hence, all the $c_p^{\mathsf{K_i}}$ are data-obvious. Moreover, rl is stored in plaintext and every entry in the list is padded to achieve same length. Finally, we can prevent timing attacks on rl by ensuring that every time REV accesses the list to either send back a token, or add a new id, it goes through the whole list.

7 Conclusion

In this paper, we proposed MF, a hybrid encryption scheme that combines *both* SSE and ABE in a way that the main advantages of each encryption technique are used. The proposed scheme enables clients to search over encrypted data by using an SSE scheme, while the symmetric key required for the decryption is protected via a Ciphertext-Policy Attribute-Based Encryption scheme. Moreover, our construction supports the revocation of users by utilizing the functionality provided by SGX. In contrast to recent works [12], the revocation mechanism has been separated from the actual ABE scheme and is exclusively based on the utilization of trusted SGX enclaves.

Finally, we believe that this work can pave the way for privacy-preserving data sharing between different organizations that are using separate and completely distinct cloud platforms. Hence, one of our main future goals is to test MF in a multi-cloud environment. We hope that this has the potential to solve important problems, such as how a patient that is for example travelling and is in a critical health condition, can share her medical history with (authorized) doctors of a different country (cross-border data sharing) [10].

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy. pp. 321–334. SP '07, IEEE Computer Society, Washington, DC, USA (2007)
2. Boyko, V.: On the security properties of oaep as an all-or-nothing transform. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 503–518. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
3. Costan, V., Devadas, S.: Intel sgx explained. Cryptology ePrint Archive, Report 2016/086 (2016), https://eprint.iacr.org/2016/086
4. Dowsley, R., Michalas, A., Nagel, M., Paladi, N.: A survey on design and implementation of protected searchable data in the cloud. Computer Science Review (2017)
5. Fisch, B., Vinayagamurthy, D., Boneh, D., Gorbunov, S.: Iron: Functional encryption using intel sgx. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 765–782. CCS '17, ACM (2017)
6. Guo, W., Dong, X., Cao, Z., Shen, J.: Efficient attribute-based searchable encryption on cloud storage. Journal of Physics: Conference Series (2018)
7. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. pp. 965–976 (2012)
8. Lee, S., Shih, M., Gera, P., Kim, T., Kim, H., Peinado, M.: Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: 26th USENIX Security Symposium, BC, Canada, August 16-18, 2017. pp. 557–574 (2017)
9. Liu, J.K., Yuen, T.H., Zhang, P., Liang, K.: Time-based direct revocable ciphertext-policy attribute-based encryption with short revocation list. Cryptology ePrint Archive, Report 2018/330 (2018), https://eprint.iacr.org/2018/330
10. Michalas, A., Weingarten, N.: Healthshare: Using attribute-based encryption for secure data sharing between multiple clouds. In: 2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS). pp. 811–815 (June 2017). https://doi.org/10.1109/CBMS.2017.30
11. Michalas, A., Yigzaw, K.Y.: Locless: Do you really care your cloud files are? In: 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC). pp. 618–623 (Dec 2015)
12. Michalas, A.: Sharing in the rain: Secure and efficient data sharing for the cloud. In: Proceedings of the 11th IEEE International Conference for Internet Technology and Secured Transactions (ICITST-2016). pp. 589–595. IEEE (2016)
13. Michalas, A.: The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. pp. 146–155. SAC '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3297280.3297297, http://doi.acm.org/10.1145/3297280.3297297
14. Microsoft: Microsoft Security Intelligence Report (2017)
15. Myers, S., Shull, A.: Practical revocation and key rotation. In: Smart, N.P. (ed.) Topics in Cryptology – CT-RSA 2018. pp. 157–178. Springer, Cham (2018)
16. Paladi, N., Gehrmann, C., Michalas, A.: Providing user security guarantees in public infrastructure clouds. IEEE Transactions on Cloud Computing **5**(3), 405–419 (July 2017). https://doi.org/10.1109/TCC.2016.2525991
17. Paladi, N., Michalas, A., Gehrmann, C.: Domain based storage protection with secure access control for the cloud. In: Proceedings of the 2014 International Workshop on Security in Cloud Computing. ASIACCS '14, ACM, New York, NY, USA (2014)

18. Verginadis, Y., Michalas, A., Gouvas, P., Schiefer, G., Hübsch, G., Paraskakis, I.: Paasword: A holistic data privacy and security by design framework for cloud services. In: Proceedings of the 5th International Conference on Cloud Computing and Services Science. pp. 206–213 (2015). https://doi.org/10.5220/0005489302060213
19. Weichbrodt, N., Kurmus, A., Pietzuch, P.R., Kapitza, R.: Asyncshock: Exploiting synchronisation bugs in intel SGX enclaves. In: Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I. pp. 440–457 (2016)
20. Xu, Y., Cui, W., Peinado, M.: Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland). IEEE (May 2015)