MicroSCOPE: Enabling Access Control in Searchable Encryption with the use of Attribute-based Encryption and SGX (Extended Version)

Antonis Michalas[1], Alexandros Bakas[1], Hai-Van Dang[2] and Alexandr Zalitko[1]

[1] Tampere University,
{antonios.michalas,alexandros.bakas,alexandr.zalitko}@tuni.fi
[2] University of Westminster,
H.Dang@westminster.ac.uk

Abstract. Secure cloud storage is considered as one of the most important problems 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). Our construction, MicroSCOPE, combines both ABE and SSE to utilize the advantages that each technique has to offer. We use an SSE scheme to ensure that data stored on the cloud will be protected against both *internal* and *external* attacks. Moreover, through the use of a Ciphertext-Policy ABE (CP-ABE) scheme, our construction allows efficient data sharing between multiple data owners and users. Finally, we enhance our construction with an access control mechanism by utilizing the functionality provided by SGX.

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

1 Introduction

We are in a period where cloud computing has been established as an essential platform for many businesses looking to build innovative services. However, concerns about security and privacy still remain – especially for companies and users moving their data between multiple *public* cloud services. The main reason behind this, is that most implementations assume an honest and therefore fully trusted cloud service provider (CSP) – a significant obstacle towards enabling a secure cloud posture. Having this in mind, researchers try to address the problem of *secure data storage on untrusted clouds* by looking at how modern cryptographic techniques such as Symmetric Searchable Encryption (SSE) and Attribute-Based Encryption (ABE) can be used to protect users' data.

SSE is a promising encryption technique in which users encrypt their data locally using a symmetric key prior sending them to the CSP. The most exciting thing about SSE though, is that it enables users to search directly over the encrypted data without having the need to decrypt them first. Thus, the CSP learns nothing about the content of the data, except for the information leaked during the execution of the scheme (i.e. access and search pattern). However, a drawback of such scheme is that it does not support efficient revocation since sharing an encrypted file implies sharing the underlying symmetric key.

Another promising encryption technique that can address the problem of revocation is ABE. ABE is a public key encryption scheme in which all files are encrypted under a master public key. However, in contrast to traditional public key encryption, in ABE each ciphertext is bound by a policy. A policy can be either a conjunction or a disjunction of attribute variables. Another difference from traditional schemes is that each user has a unique secret key, with attached attributes on it. This way, decryption of a file can work if and only if the attributes of a user's key satisfy the policy bound to the ciphertext. Naturally, if a data owner wishes to revoke access to another user, can do so by revoking certain attributes that correspond to that user. However, most of the schemes that address the problem of revocation are losing efficiency. Furthermore, using public key encryption to encrypt large volumes of data is rather inefficient and according to [GHW11] the cost of the encryption and decryption operations in an ABE scheme grows along with the complexity of the underlying policy.

Our Contribution: We construct MicroSCOPE – a hybrid encryption scheme that utilizes the advantages of both SSE and ABE. In MicroSCOPE, a user encrypts her data locally using the symmetric key of an SSE scheme which is then encrypted under a master public key of an ABE scheme. This is then stored online in an SGX enclave [CD16] and it is bound by a policy. Thus, if another user wishes to access the encrypted database, she needs to be able to recover the corresponding symmetric key (i.e her key attributes need to satisfy the policy of the ciphertext). Moreover, we propose a scope-based mechanism that manages access rights between users. In contrast to other approaches, we do not embed this mechanism in the ABE scheme. This allow us to keep our construction as efficient as possible.

Organization: The rest of the paper is organized as follows: 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 present the cryptographic tools needed for our construction while in Section 4 we provide a brief overview of the components that participate in our scheme. The assumed threat model is presented in Section 5 while Section 6 constitutes the core of this paper as a detailed description of our construction is presented. In Section 7, we prove the security of the protocol against the defined threat model. In Section 8, we provide experimental results and finally, Section 9 concludes the paper.

2 Related Work

In [FBB^{+}17], authors use an enclave-based, tree-based search index to design a searchable encryption scheme. Their scheme, HardIDX, addresses the problem of searching in large volumes of encrypted data by making use of the functionalities offered by SGX enclaves. Their solution places the integrity and confidentiality of the search tree in unprotected memory. Thus, every search operation leaks information, such as the access pattern and the size of the output. A promising idea is presented in [FVBG17], where the authors present an SGX-based functional encryption scheme called IRON. IRON's main functionalities, such as decryption of a file and application of a function on the decrypted file, both occur in the isolated environment offered by SGX. 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, to avoid side-channel attacks, we make sure that sensitive computations do not happen inside the enclaves. A different approach to the problem of data maintenance on untrusted clouds is presentes in [MS18] where authors propose a revocable hybrid encryption scheme enchanced with a key-rotation mechanism to prevent key-scrapping attacks. The scheme makes use of an All-or-Nothing-Transformation (AONT) [Boy99] to prevent revoked users from accessing stored data.

More precisely, Optimal Asymmetric Encryption Padding (OAEP) is used as the AONT due to the fact that reversing OAEP, requires the entire output to be known. As a result, changing random bits of the output renders OAEP's inversion infeasible. However, to decrypt a file, the changed bits need to be stored so that the AONT can be later reversed. Naturally, this implies that with each re-encryption, the size of the ciphertext grows. Thus, decrypting a file that has been re-encrypted multiple times becomes 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. In [LYZL18] a revocable Ciphertext-Policy ABE (CP-ABE) scheme in which a revocation list is embedded into the ciphertexts was presented. Naturally, as the revocation list grows, the ciphertexts would become larger, thus rendering any decryption or file modification operation more expensive. To deal with this, a method based on Hierarchical Identity Based Encryption (HIBE) [BBG05], where users' secret keys expire after a certain period of time was proposed. Hence, the revocation list can only include those keys that were revoked before their expiration date. Similarly, in [BGK08] the authors extended their work on Revocable Identity Based Encryption by constructing a Key-Police ABE (KP-ABE). However, according to their design, revocation relies on frequent key updates for all the different attributes. Naturally, such a solution does not scale well and thus cannot have practical use. This work is an extension of [Mic18, BM19a, Mic19].

3 Cryptographic Primitives

In this section, we present formal definitions for the two main encryption schemes of our construction, CP-ABE and SSE, as described in [BSW07] and [BM19b] respectively.

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

1. $\mathsf{CPABE.Setup}$ 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 $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$.

2. $\mathsf{CPABE.Gen}$ 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 bound both to the corresponding list of attributes and the user. We denote this by $(\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}) \leftarrow \mathsf{Gen}(\mathsf{MSK}, \mathcal{A}, u_i)$.

3. $\mathsf{CPABE.Enc}$ 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 $\mathsf{c_P} \leftarrow \mathsf{Enc}(\mathsf{MPK}, m, P)$.

4. $\mathsf{CPABE.Dec}$ 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 six polynomial algorithms $\mathsf{DSSE} = (\mathsf{KeyGen}, \mathsf{InGen}, \mathsf{AddFile}, \mathsf{Search}, \mathsf{Delete})$ such that:

- $\mathsf{DSSE.KeyGen}$ is 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.

- DSSE.InGen 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.
- DSSE.AddFile is a probabilistic algorithm that takes as input a secret key K and a file f and outputs an add token $\tau_\alpha(f)$ and a ciphertext c_f. The token and the ciphertext are then sent to the storage server, where c_f will be added to the collection of ciphertexts and the index γ will be updated accordingly.
- DSSE.Search Is a deterministic algorithm that takes as input a secret key K and a keyword w and outputs a search token $\tau_s(w)$. The token is then sent to the storage server who will output a sequence of file identifiers $\mathbf{I}_w \subset \mathbf{c}$.
- DSSE.Delete is a deterministic algorithm that takes as input a secret key K and a file identifier $id(f)$ and outputs a delete token $\tau_d(f)$ for f. The token will be sent to the storage server, who will delete c_f and update the index γ accordingly.

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 [DMNP17].

Definition 3. (Dynamic CKA 2-security). Let DSSE = (KeyGen, InGen, AddFile, Search, Delete) 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.

4 Architecture

In this section, we provide an overview of the underlying system model by describing all the different components[1] along with their functionality.

[1]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

Cloud Service Provider (CSP): We consider a cloud computing environment similar to the one described in [PGM17, ?]. CSP is responsible for storing encrypted data. Moreover, it must be SGX-enabled since core entities will be running in the 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. Furthermore, MS is responsible for generating and distributing ABE keys to the registered users. 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 data owners and are needed to recover/decrypt data. Every registered user can directly contact KT and request access to the stored ciphertexts. KT is also SGX-enabled and is running in an enclave called the KT Enclave.

Revocation Authority (REV): REV is responsible for controlling access rights. REV maintains a mapping of each user with her valid scopes. Each time a scope is revoked from a user, REV updates its database. Similar to MS and KT, REV is also SGX-enabled and is running in an enclave called the REV Enclave.

User (u_i): A user interacts with the CSP to manage certain files that has access to according to her assigned scopes (access rights). The set of access rights of u_i is denoted as $\mathcal{SC}_i = \{(j, s_i^j), (z, s_i^k), \ldots (k, s_i^z)\}$ where $j, k, \ldots, z$ represent a collection of files encrypted under the symmetric keys $\mathsf{K_j}, \mathsf{K_k}, \ldots, \mathsf{K_z}$ and s_i^j is a one dimensional bit array of length four that represents the scopes (i.e. view, add, delete, revoke) assigned to u_i for each data collection. For example, if $s_j^i = [1010]$, then u_j has access rights view and delete for data encrypted under the symmetric key $\mathsf{K_i}$.

SGX: Below we provide a brief presentation of the main SGX functionalities needed for our construction. A more detailed description can be found in [FVBG17, CD16]

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 rpt generated by enc_j containing information about the enclave itself and the program running in it. This report is signed with a secret key $\mathsf{sk_{rpt}}$ 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 $\mathsf{sk_{rpt}}$ 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.

5 Threat Model

Our threat model is similar to the one described in [PGM17], which is based on the Dolev-Yao adversarial model [DY83]. We further assume that privileged access rights can be used by a remote adversary $\mathcal{ADV}$ to leak confidential information. $\mathcal{ADV}$, e.g. a

registered.

corrupted system administrator, can obtain remote access to any host maintained by the CSP, but cannot access the volatile memory of guest VMs residing on the compute hosts of the CSP. Furthermore, we assume that $\mathcal{ADV}$ can load programs in the enclaves and observe their output. This assumption significantly strengthens $\mathcal{ADV}$ since we need to ensure that only honest attested programs with correct inputs will run in the enclaves. Finally, we extend the above threat model by defining a set of attacks available to $\mathcal{ADV}$.

Attack 1 (Successful Scope Substitution Attack – SSA). *Let $\mathcal{ADV}$ be an adversary that corrupts a registered user u_m, whose set of valid scopes is given by s^i_m for data encrypted under the symmetric key $\mathsf{K_i}$ $\mathcal{ADV}$ wishes to tamper with u_m's access rights by providing her with more scopes. We say that $\mathcal{ADV}$ successfully launches an SSA attack iff she can change bits from 0 to 1 in s^i_m and produce a new array $s'^i_m \neq s^i_m$ that will be accepted as valid by the corresponding authorities.*

Attack 2 (Successful Revocation of Legitimate User Attack – RLUA). *Let $\mathcal{ADV}$ be an adversary that corrupts a registered user u_m who has access only to data encrypted under a secret key $\mathsf{K_m}$. Additionally, let u_ℓ be a legitimate user that has access to data encrypted under a secret key K_ℓ, $\ell \neq m$. $\mathcal{ADV}$ successfully launches an RLUA attack iff she manages to revoke scopes to u_ℓ for data that is encrypted under K_ℓ.*

Attack 3 (Successful Compromise of Revoked User Attack – CRUA). *Let $\mathcal{R}_{\mathsf{K_i}}$ be the set of all users that their access to the data encrypted under $\mathsf{K_i}$ has been revoked completely (i.e. $s^i_{\mathcal{R}_{\mathsf{K_i}}} = [0000]$). Moreover, let $\mathcal{ADV}$ be an adversary that corrupts a user u_m where $u_m \in \mathcal{R}_{\mathsf{K_i}}$. $\mathcal{ADV}$ successfully launches a CRUA attack iff she manages to extract any valuable information about the content of the files that are encrypted with $\mathsf{K_i}$.*

6 MicroSCOPE (MSCOPE)

In this section, we present MicroScope (MSCOPE) that constitutes the core of this paper's contribution. MSCOPE is built around nine main protocols: Setup, ABEUserKey, Store, KeyTrayStore, KeyShare, Search, Update, Delete, and Revoke.

MSCOPE.Setup: Each entity generates a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ for a CCA2 secure public cryptosystem as well as a signing and a verification key for a EUF-CMA secure signature scheme. Furthermore, MS runs CPABE.Setup and generates a master public/private key pair $(\mathsf{MPK}, \mathsf{MSK})$.

MSCOPE.ABEUserKey : This algorithm is executed by a registered user u_i that wishes to receive a secret CP-ABE key. Since MS is responsible for generating such keys, u_i needs to contact MS and request a key. MS will then execute $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}} \leftarrow \mathsf{CPABE.Gen}(MSK, \mathcal{A}, u_i)$, where $\mathcal{A}$ is the set of attributes that is derived from u_i's registered information. Finally, $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}$ is sent back to u_i over a secure channel.

```
MSCOPE.ABEUserKey
  Input: MSK ; $u_i$; $\mathcal{A}$
  Output: $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}$

    1. $u_i$ attests MS
    2. $u_i$ requests a new CP-ABE key
    3. MS generates key: $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}} \leftarrow \mathsf{CPABE.Gen}(\mathsf{MSK}, \mathcal{A}, u_i)$
    4. MS sends $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}}$ to $u_i$
```

MSCOPE.Store : After a successful registration, we assume that u_i has received a valid credential ($cred_i$) that may use to login to a cloud service offered by the CSP. Additionally, u_i is now able to store data to the cloud storage. During this phase, the communication takes place between the user and the CSP. First, u_i contacts the CSP by sending the following: $m_1 = \big\langle r_1, \mathsf{E}_{\mathsf{pk_{CSP}}}(Auth), StoreReq, H_1 \big\rangle$, where r_1 is a random number generated by u_1, $Auth$ is an authenticator that allows u_i to prove to the CSP that is a legitimate/registered user and H_1 is the following hash $H(r_1||Auth||StoreReq)$. Upon reception, CSP verifies the freshness of the message, the identity of the user and starts processing the store request. To do so, CSP creates the message $m_2 = \big\langle r_2, \sigma_{CSP}(H_2) \big\rangle$, where H_2 is $H(r_2||u_i)$ and $\sigma_{CSP}(H_2)$ is a signature of CSP on H_2. Then, m_2 is sent back to u_i. Upon reception, u_i verifies both the freshness and the integrity of the message. Now, u_i simply generates a symmetric key $\mathsf{K_i}$ by running $\mathsf{DSSE.KeyGen}$. This key will be used to protect the data that will be stored in the cloud. The final step of this phase is the storage of encrypted files by u_i to a storage resource offered by the CSP. User u_i runs $\mathsf{DSSE.InGen}$ – a probabilistic algorithm that takes as input the symmetric secret key $\mathsf{K_i}$ that generated earlier and a collection of files $\mathbf{f_i}$ and outputs a collection of ciphertexts $\mathbf{c_i}$ as well as an encrypted index γ_i. Additionally, u_i generates a unique index $\mathsf{idx_{K_i}}$ for the key $\mathsf{K_i}$. Finally, u_i sends to the CSP: $m_3 = \big\langle r_3, \mathsf{E}_{\mathsf{pk_{CSP}}}(\gamma_i, \mathsf{idx_{K_i}}), \mathbf{c_i}, H_3 \big\rangle$, where $H_3 = H(r_3||\gamma_i||\mathbf{c_i}||\mathsf{idx_{K_i}})$. Upon reception, CSP verifies both the integrity and the freshness of m_3 and stores $\mathbf{c_i}$ along with the encrypted index γ_i, and the index $\mathsf{idx_{K_i}}$.

MSCOPE.Store

Input: User's authenticator Auth, a collention of files $\mathbf{f_i}$
Output: A collection of ciphertexts $\mathbf{c_i}$ along with an encrypted index γ_i are stored on the CSP

1. u_i generates a random number r_1
2. u_i sends $m_1 = \langle r_1, \mathsf{E}_{\mathsf{pk_{CSP}}}(Auth), StoreReq, \sigma_i(H_1) \rangle$ to the CSP
3. CSP verifies the freshness of the message and the identity of the user
4. CSP sends : $m_2 = \langle r_2, \sigma_{CSP}(H_2) \rangle$ to u_i
5. u_i verifies the freshness and the integrity of m_2
6. u_i runs $\mathsf{K_i} \leftarrow \mathsf{DSSE.KeyGen}$
7. u_i generates a unique index idx_{K_i} for the symmetric key $\mathsf{K_i}$
8. u_i runs $(\mathbf{c_i}, \gamma_i) \leftarrow \mathsf{DSSE.InGen}(\mathsf{K_i}, \mathbf{f_i})$
9. u_i sends $m_3 = \langle r_3, \mathsf{E}_{\mathsf{pk_{CSP}}}(\gamma_i, \mathsf{idx}_{K_i}), \mathbf{c_i}, \sigma_i(H_3) \rangle$ to the CSP
10. CSP verifies the integrity and the freshness of m_3
11. CSP stores $\{\mathbf{c_i}, \gamma_i, \mathsf{idx_{K_i}}\}$

MSCOPE.KeyTrayStore : This is a probabilistic algorithm executed by the data owner to store her symmetric key $\mathsf{K_i}$ in KT. The data owner first runs $c_P^{\mathsf{K_i}} \leftarrow \mathsf{CPABE.Enc}(\mathsf{MPK}, \mathsf{K_i}, P)$ to obtain the encryption of $\mathsf{K_i}$, which is associated with a policy P. The generated ciphertext $c_P^{\mathsf{K_i}}$ is then sent to KT who stores it locally. In particular, u_i sends $m_4 = \big\langle \mathsf{r_4}, \mathsf{E}_{\mathsf{pk_{KT}}}(u_i, \mathsf{idx_{K_i}}), c_P^{\mathsf{K_i}}, \sigma_i\big(H\big(r_4||u_i||c_P^{\mathsf{K_i}}||\mathsf{idx_{K_i}}\big)\big) \big\rangle$. Apart from that, u_i also needs to assign scopes (access rights) to the registered users that wishes to share $\mathbf{c_i}$ with. To do so, she sends $m_5 = \Big\langle \mathsf{E}_{\mathsf{pk_{REV}}}(\mathsf{idx_{K_i}}, \{(u_1, s_1^i), (u_2, s_2^i), \dots\}), \sigma_i(H(\mathsf{idx_{K_i}}||\{(u_1, s_1^i), (u_2, s_2^i), \dots\})) \Big\rangle$ to REV, where u_j is the identifier of the users to which u_i assigned scopes. Finally, KT stores the user's identifier, u_i, along with the unique index $\mathsf{idx}_{\mathsf{K_i}}$ of the symmetric key, next to $c_p^{\mathsf{K_i}}$.

MSCOPE.KeyTrayStore

Input: $\mathsf{K_i}$, policy P.
Output: KT stores $c_p^{\mathsf{K_i}}$

1. u_i generates a random number r_4
2. u_i runs: $c_P^{\mathsf{K_i}} \leftarrow \mathsf{CPABE.Enc}(\mathsf{MPK}, \mathsf{K_i}, P)$
3. u_i sends to KT $m_4 = \left\langle E_{pk_{KT}}(r_4, u_i, \mathsf{idx}_{\mathsf{K_i}}), c_P^{\mathsf{K_i}}, \sigma_i\left(H\left(r_4||u_i||c_P^{\mathsf{K_i}}||\mathsf{idx}_{\mathsf{K_i}}\right)\right)\right\rangle$
4. u_i sends $m_5 = \Big\langle \mathsf{E}_{\mathsf{pk_{REV}}}(\mathsf{idx}_{\mathsf{K_i}}, \{(u_1, s_1^i), (u_2, s_2^i), \dots\}),$ $\sigma_i(H(\mathsf{idx}_{\mathsf{K_i}}||\{(u_1, s_1^i), (u_2, s_2^i), \dots\}))\Big\rangle$ to REV
5. KT stores: $\{u_i, c_P^{\mathsf{K_i}}, \mathsf{idx}_{\mathsf{K_i}}\}$
6. REV stores $\langle \mathsf{idx}_{\mathsf{K_i}}, \{(u_1, s_1^i), (u_2, s_2^i), \dots\}\rangle$ into the list of valid scopes L_{VS}.

MSCOPE.KeyShare : **MSCOPE.KeyShare** : We now assume that another registered user u_j, $j \neq i$ wishes to access $\mathbf{c_i}$. The important thing to notice here is that the data sharing will be done without the involvement of u_i. To this end, u_j sends $m_6 = \langle r_6, \mathsf{E}_{\mathsf{pk_{KT}}}(u_j, u_i), \sigma_j(H(r_6||u_j||u_i))\rangle$ to KT. KT will then reply with $m_7 = \langle r_7, \mathsf{E}_{\mathsf{pk_{REV}}}(u_j, \mathsf{idx}_{\mathsf{K_i}}),$ $\sigma_{KT}(H(r_7||u_j||\mathsf{idx}_{\mathsf{K_i}}))\rangle$. This message will then be forwarded to REV who will locate s_j^i and will send $m_8 = \langle r_8, \mathsf{E}_{\mathsf{pk_{KT}}}(s_j^i)), \sigma_{REV}(H(r_8||s_j^i))\rangle$ to KT. At this point, KT retrieves $c_P^{\mathsf{K_i}}$ and sends $m_9 = \langle r_9, \mathsf{E}_{\mathsf{pk_{CSP}}}(u_j, t, s_j^i, \mathsf{idx}_{\mathsf{K_i}}), c_P^{\mathsf{K_i}}, \sigma_{KT}(H(r_9||u_j||t||s_j^i||\mathsf{idx}_{\mathsf{K_i}}||c_P^{\mathsf{K_i}})\rangle$ to u_j. Finally, u_j uses her private CP-ABE key to recover $\mathsf{K_i}$.

MSCOPE.KeyShare

Input: User's id u_j and data owner's id u_i
Output: u_j receives $c_p^{\mathsf{K_i}}$

1. u_j sends $m_6 = \langle r_6, \mathsf{E}_{\mathsf{pk_{KT}}}(u_j, u_i), \sigma_j(H(r_6||u_j||u_i))\rangle$ to KT
2. KT replies with $m_7 = \langle r_7, \mathsf{E}_{\mathsf{pk_{REV}}}(u_j, \mathsf{idx}_{\mathsf{K_i}}), \sigma_{KT}(H(r_7||u_j||\mathsf{idx}_{\mathsf{K_i}}))\rangle$ to the user who forwards the message to REV.
3. REV generates and sends $m_8 = \langle r_8, \mathsf{E}_{\mathsf{pk_{KT}}}(s_j^i)), \sigma_{REV}(H(r_8||s_j^i))\rangle$ to KT.
4. KT sends $m_9 = \langle r_9, \mathsf{E}_{\mathsf{pk_{CSP}}}(u_j, t, s_j^i, \mathsf{idx}_{\mathsf{K_i}}), c_P^{\mathsf{K_i}}, \sigma_{KT}(H(r_9||u_j||t||s_j^i||\mathsf{idx}_{\mathsf{K_i}}||c_P^{\mathsf{K_i}})\rangle$ to the user.

MSCOPE.Search : Now that u_j has gained access to $\mathsf{K_i}$, she can start searching directly over the encrypted data for those files that contain a specific keyword w of her choice. To do so, u_j generates a search token $\tau_s(w)$, for the keyword w and sends $m_{10} = \langle m_8, \tau_s(w), \sigma_j(H(\tau_s(w)))\rangle$ to the CSP. Upon reception, CSP checks if u_j is eligible to search for files by opening m_8, looking at the timestamp provided by KT and verifying that $s_j^i[0] = 1$. If the verifications are correct, CSP runs $\mathsf{DSSE.Search}(\gamma_i, \mathbf{c_i}, \tau_s(w)) \rightarrow \mathbf{I}_w$.

MSCOPE.Search

Input: Keyword w
Output: $\mathbf{I_w}$

1. u_j generates $\tau_s(w)$ for a keyword w.
2. u_j sends $m_{10} = \langle m_8, \tau_s(w), \sigma_j\left(H\left(\tau_s(w)\right)\right)\rangle$ to the CSP
3. CSP opens m_8 to check if the timestamp is valid and if $s_j^i[0] = 1$
4. Assuming that all verifications are successful, CSP identifies the encrypted index γ_i and sequence of ciphertexts $\mathbf{c_i}$ based on $\mathsf{idx}_{\mathsf{K_i}}$
5. CSP runs: $\mathsf{DSSE.Search}(\gamma_i, \mathbf{c_i}, \tau_s(w)) \rightarrow \mathbf{I}_w$.
6. CSP sends to u_j: $\mathbf{I}_w$

MSCOPE.Update : A registered user u_j can also update the database by adding new files, provided that $s_j^i[1] = 1$. To do so, u_j first generates an add token $\tau_\alpha(f)$ and sends the following to the CSP: $m_{11} = \big\langle m_8, \tau_\alpha(f), c_f, \sigma_j\big(H\big(\tau_\alpha(f)||c_f\big)\big)\big\rangle$. Upon reception, CSP checks if u_j is eligible to add a file by opening m_8, looking at the timestamp provided by KT and verifying that $s_j^i[1] = 1$. Assuming that the verifications are successful, CSP executes $\mathsf{DSSE.Add}\left(\gamma_i, \tau_\alpha(f), c_f\right) \rightarrow \left(\gamma_i', \mathbf{c_i'}\right)$ and stores the new ciphertext and the updated index γ_i'.

MSCOPE.Update

Input: A file $\mathbf{f}$
Output: A new ciphertext c_f is stored in the CSP.

1. u_j runs generates $\tau_\alpha(f)$.
2. u_j sends $m_{11} = \big\langle m_8, \tau_\alpha(f), c_f, \sigma_j\big(H\big(\tau_\alpha(f)||c_f\big)\big)\big\rangle$ to the CSP.
3. CSP opens m_8 to check if the timestamp is valid and if $s_j^i[1] = 1$
4. Assuming that all verifications are successful, CSP identifies the encrypted index γ_i and sequence of ciphertexts $\mathbf{c_i}$ based on $\mathsf{idx}_{\mathsf{K_i}}$
5. CSP runs $\mathsf{DSSE.Add}\left(\gamma_i, \tau_\alpha(f), c_f\right) \rightarrow \left(\gamma_i', \mathbf{c_i'}\right)$

MSCOPE.Delete : In our construction, users can also delete files. Let u_j be a registered user who is eligible to delete files (i.e. $s_j^i[2] = 1$). To do so, u_j first generates a delete token $\tau_d(f)$ and then sends $m_{12} = \big\langle m_8, \tau_d(f), \sigma_j(H(\tau_d(f)))\big\rangle$ to the CSP. Just as before, the CSP verifies the integrity and freshness of m_8 and checks whether $s_j^i[2] = 1$ or not. Given that the verifications are successful, the CSP executes $\mathsf{DSSE.Delete}(\gamma_i, \tau_d(f)) \rightarrow \gamma_i'$, which results to the deletion of the requested file and the update of the corresponding index.

MSCOPE.Delete

Input: A file **f**
Output: CSP deletes c_f.

1. u_j generates $\tau_d(f)$.
2. u_j sends $m_{12} = \langle m_8, \tau_d(f), \sigma_j(H(\tau_d(f)))\rangle$ to the CSP.
3. CSP opens m_8 to check if the timestamp is valid and if $s_j^i[2] = 1$
4. Assuming that all verifications are successful, CSP identifies the encrypted index γ_i and sequence of ciphertexts $\mathbf{c_i}$ based on $\mathsf{idx}_{\mathsf{K_i}}$
5. CSP runs $\mathsf{DSSE.Delete}(\gamma_i, \tau_d(f)) \rightarrow \gamma_i'$

MSCOPE.Revoke : Finally, we enhance our construction with a revocation mechanism. Any registered user u_j, for whom $s_j^i[3] = 1$, can access this mechanism to revoke scopes from other registered users. To this end, u_j contacts REV by sending $m_{13} = \langle r_{13}, \mathsf{E}_{\mathsf{pk}_{\mathsf{REV}}}(u_j, u_\ell, n), c_P^{\mathsf{K_i}}, \sigma_j(H(r_{11}||u_j||u_\ell||n||c_P^{\mathsf{K_i}}))\rangle$, where u_ℓ is the id of the user to be revoked and $n \in [0, 3]$ is an index of the one dimensional bit array s_j^i and specifies which scope will be revoked (flipped). After REV recovers both identities, it generates a report $\mathsf{rpt_i}$ containing $m_{14} = \langle r_{14}, E_{\mathsf{pk}_{\mathsf{KT}}}(u_\ell), c_P^{\mathsf{K_i}}, \sigma_{REV}(H(r_{12}||c_P^{\mathsf{K_i}}||u_\ell))\rangle$ which is sent to the KT. Upon reception, KT verifies the report and checks if u_ℓ is the data owner by comparing u_ℓ with the value stored next to $c_P^{\mathsf{K_i}}$. If u_ℓ is not the data owner, KT will proceed by generating a report $\mathsf{rpt_j}$ containing $\mathsf{idx}_{\mathsf{K_i}}$ that will be sent to REV. At this point, REV will retrieve L_{VS} from its local database to check whether u_i is eligible to revoke scopes from users in the specified dataset. Assuming that the verification is successful, REV removes the specified scope by setting $s_\ell^i[n] = 0$.

MSCOPE.Revoke

Input: u_ℓ, n
Output: $s_\ell^i[n] \rightarrow 0$.

1. u_j sends $m_{13} = \langle r_{13}, \mathsf{E}_{\mathsf{pk}_{\mathsf{REV}}}(u_j, u_\ell, n), c_P^{\mathsf{K_i}}, \sigma_j(H(r_{11}||u_j||u_\ell||n||c_P^{\mathsf{K_i}}))\rangle$ to REV.
2. REV attests KT and sends a report $\mathsf{rpt_i}$ containing $m_{14} = \langle r_{14}, E_{\mathsf{pk}_{\mathsf{KT}}}(u_\ell), c_P^{\mathsf{K_i}}, \sigma_{REV}(H(r_{12}||c_P^{\mathsf{K_i}}||u_\ell))\rangle$.
3. KT verifies $\mathsf{rpt_i}$ and compares u_ℓ with the id stored next to $c_p^{\mathsf{K_i}}$ to make sure that u_ℓ is not the data owner.
4. KT attests REV and sends a report $\mathsf{rpt_j}$ containing $\mathsf{idx}_{\mathsf{K_i}}$ to REV.
5. REV verifies $\mathsf{rpt_j}$ and checks if $s_j^i[3] = 1$.
6. REV sets $s_\ell^i[n] = 0$

7 Security Analysis

We prove the security of our construction in three different stages. First, we conduct a simulation-based security analysis. Then, we elaborate on the security of SGX and why MicroSCOPE is *not* susceptible to side-channel attacks. Finally, we conclude the security analysis by proving the resistance of our scheme against the attacks presented in Section 5.

7.1 Simulation-Based Security

We assume the existence of a simulator $\mathcal{S}$. $\mathcal{S}$ can simulate real protocol's algorithms in a way that any polynomial time adversary $\mathcal{ADV}$ cannot 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.

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	Ideal Experiment
1. $\mathbf{EXP}^{real}_{MSCOPE}(1^\lambda)$:	1. $\mathbf{EXP}^{ideal}_{MSCOPE}(1^\lambda)$:
2. $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{MSCOPE.Setup}(1^\lambda)$	2. $(\mathsf{MPK}) \leftarrow \mathcal{S}(1^\lambda)$
3. $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}} \leftarrow \mathcal{ADV}^{\mathsf{MSCOPE.ABEUserKey}(\mathsf{MSK},\mathcal{A})}$	3. $\mathsf{sk}_{\mathcal{A},\mathsf{u_i}} \leftarrow \mathcal{ADV}^{\mathcal{S}(1^\lambda)}$
4. $ct \leftarrow \mathsf{CPABE.Enc}(\mathsf{mpk}, m)$	4. $ct \leftarrow \mathcal{S}(1^\lambda, 1^{\lvert m \rvert})$
5. $(\gamma, c) \leftarrow \mathcal{ADV}^{\mathsf{DSSE.InGen}(\mathsf{K},\mathbf{f})}$	5. $(\gamma, c) \leftarrow \mathcal{ADV}^{\mathcal{S}(\mathcal{L}_{in}(\mathbf{f}))}$
6. $\mathsf{MSCOPE.Search}(\text{“}search\text{”}, m_{search}) \rightarrow \mathbf{I_w}$	6. $\mathcal{S}(\text{“}search\text{”}, m_{search}) \rightarrow \mathbf{I}^*_\mathbf{w}$
7. $\mathsf{MSCOPE.Update}(\text{“}update\text{”}, m_{add}) \rightarrow (\gamma', c')$	7. $\mathcal{S}(\text{“}update\text{”}, m_{add}) \rightarrow (\gamma', c')$
8. $\mathsf{MSCOPE.Delete}(\text{“}delete\text{”}, m_{delete}) \rightarrow (\gamma', c')$	8. $\mathcal{S}(\text{“}delete\text{”}, m_{delete}) \rightarrow (\gamma', c')$
9. Output b	9. Output b'

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

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

At a high-level, we construct a simulator $\mathcal{S}$ to replace the MSCOPE algorithms. $\mathcal{S}$ simulates Key generation and encryption oracles. $\mathcal{S}$ is given the length of the challenge message as well as the leakage functions $\mathcal{L}_i$. In the real experiment, the challenger $\mathcal{C}$ runs $\mathsf{K_i} \leftarrow \mathsf{DSSE.KeyGen}(1^\lambda)$ 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 $\mathsf{MSCOPE.Search}$, $\mathsf{MSCOPE.Update}$ and $\mathsf{MSCOPE.Delete}$ oracles by getting as input the simulated tokens of the SSE security game. In our game, we exclude $\mathsf{MSCOPE.Revoke}$ since rl is not retrievable during the execution of the protocol. Also, 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 MSCOPE 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.

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

- $\mathsf{MSCOPE.ABEUserKey}^*$: 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 $\mathsf{sk}^*_{\mathsf{A},\mathsf{u_i}}$. This key is a random string that has the same length as the output of the real $\mathsf{MSCOPE.ABEUserKey}$. The key will be given to $\mathcal{ADV}$.

- $\mathsf{MSCOPE.KeyShare}^*$: 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 $\mathcal{ADV}$.

- $\mathsf{MSCOPE.Search}^*$: When $\mathcal{ADV}$ wishes to make a search query, $\mathcal{S}$ gets as input the leakage function $\mathcal{L}_s$ and simulates the search token $\tau_s(w)$ and a simulated response.

- $\mathsf{MSCOPE.Update}^*$: When $\mathcal{ADV}$ wishes to make an add query, $\mathcal{S}$ gets as input the leakage function $\mathcal{L}_\alpha$ and simulates the add token $\tau_s(w)$ and ciphertext, to be added to the collection. $\mathcal{S}$ keeps track of file insertions, so that it can create consistent tokens and responses to the search queries.

- $\mathsf{MSCOPE.Delete}^*$: When $\mathcal{ADV}$ wishes to make a delete query, $\mathcal{S}$ gets as input the leakage function $\mathcal{L}_d$ and simulates the delete token $\tau_s(w)$. $\mathcal{S}$ keeps track of file insertions, so that it can create consistent tokens and responses to the search queries.

- $\mathsf{MSCOPE.Revoke}^*$: In contrast to the real experiment, the system does not revoke any user.

In a pre-processing phase, 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.

$\boxed{\textbf{Hybrid 0}}$ MSCOPE runs normally.

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

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

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

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 $\mathsf{MSCOPE.KeyShare}$ with $\mathsf{MSCOPE.KeyShare}^*$.

$\boxed{\textbf{Hybrid 3}}$ Like Hybrid 2, but $\mathsf{MSCOPE.KeyShare}^*$ runs instead of $\mathsf{MSCOPE.KeyShare}$. Also, the algorithm outputs $\perp$ if $\mathcal{ADV}$ sends m_7 but never contacted 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_7, 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. $\square$

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 MSCOPE. For the rest of the proof we assume that $\mathcal{ADV}$ performs search, add and delete queries. The simulator now gets access to all leakage functions $\mathcal{L}$ from the SSE scheme.

$\boxed{\textbf{Hybrid 4}}$ Like Hybrid 3, but when $\mathcal{ADV}$ makes a search query, $\mathcal{S}$ is given the leakage function $\mathcal{L}_s$ and generates $\mathbf{I}^*_\mathbf{w}$ which is then sent to the user. Moreover, $\mathcal{S}$ outputs $\perp$ if $\mathcal{ADV}$ never contacted KT.

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_9, 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_9 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.

□

$\boxed{\textbf{Hybrid 5}}$ Like Hybrid 4, but when $\mathcal{ADV}$ makes an update query, $\mathcal{S}$ is given the leakage function $\mathcal{L}_a$ and tricks $\mathcal{ADV}$ into thinking that she updated the database. Moreover, $\mathcal{S}$ outputs $\perp$ if $\mathcal{ADV}$ never contacted KT.

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. 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_{10} 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. □

$\boxed{\textbf{Hybrid 6}}$ Like Hybrid 5, but when $\mathcal{ADV}$ makes a delete query, $\mathcal{S}$ is given the leakage function $\mathcal{L}_d$ and tricks $\mathcal{ADV}$ into thinking that she deleted a certain file from the database $\mathcal{S}$ outputs $\perp$ if $\mathcal{ADV}$ never contacted KT.

Lemma 4. *Hybrid 6 is indistinguishable from Hybrid 5*

Proof. By assuming the $\mathcal{L}_i-$ security of the DSSE scheme, we know that $\mathcal{ADV}$ will not be able to distinguish between the real delete token and the simulated one. Moreover, similar to the previous Hybrid, if $\mathcal{ADV}$ can generate m_{11} without having contacted KT, then she can also forge KT's signature – which can only happen with negligible probability. Thus, $\mathcal{ADV}$ can only distinguish between Hybrids 5 and 6 with negligible probability. □

$\boxed{\textbf{Hybrid 7}}$ Like Hybrid 6 but instead of MSCOPE.Revoke, $\mathcal{S}$ executes MSCOPE.Revoke*. Moreover, $\mathcal{S}$ outputs $\perp$ if $\mathcal{ADV}$ sends m_{13} to REV.

Lemma 5. *Hybrid 7 is indistinguishable from Hybrid 6.*

Proof. Since the revocation list is not retrievable during the execution of the protocol, $\mathcal{ADV}$ can never tell if she really revoked any scope from a specific user. $\mathcal{ADV}$ could try to bypass KT's authentication by generating and sending rpt directly to REV. However, since $\mathcal{ADV}$ does not posses $\mathsf{sk}_{\mathsf{rpt}}$, she can only do that with negligible probability. Hence, $\mathcal{ADV}$ can only distinguish between Hybrids 6 and 7 with negligible probability. □

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. □

7.2 SGX Security

Recent works [CD16, XCP15, LSG+17, AIKM16] have shown that SGX is vulnerable to software attacks. However, according to [FVBG17], 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.

7.3 Protocol Security

Proposition 1 (SSA Soundness)**.** *Let $\mathcal{ADV}$ be a malicious adversary that corrupts a user u_m with valid scopes s_m^j for data encrypted under a symmetric key $\mathsf{K_j}$. Then $\mathcal{ADV}$ can not successfully perform an SSA attack.*

Proof. Assume that $\mathcal{ADV}$ produces a new set of valid scopes $\mathcal{SC}'_m$ and successfully replaces u_m's valid scopes $\mathcal{SC}_m$. To do so, $\mathcal{ADV}$ needs to successfully flip at least one bit $s_m^j[n]$ to its opposite value $\overline{s_m^j[n]}$, resulting to the new scope array $s_m'^j$. We examine ADV's behavior by analyzing two distinct cases:

α. $\mathbf{0 \leq n \leq 2}$

If $0 \leq n \leq 2$, then $s_m^j[n]$ will correspond to one of the following: $\{\mathsf{view}, \mathsf{add}, \mathsf{delete}\}$. In that case, the attack would commence with $\mathcal{ADV}$ sending $m =< m_8, m_{action} >$, where $m_8 = \langle \mathsf{E}_{\mathsf{pk_{CSP}}}(u_\ell, t, \mathsf{idx}_{\mathsf{K_i}}, s_\ell^i), c_p^{\mathsf{K_i}}, \sigma_{KT}\left(H\left(u_\ell || t || \mathsf{idx}_{\mathsf{K_i}} || c_p^{\mathsf{K_i}} || s_\ell^i\right)\right)\rangle$ and m_{action} is the component of the message which is associated with view, add or delete operations. However, $\mathcal{ADV}$ cannot know $\mathsf{idx}_{\mathsf{K_i}}$ and thus, can never construct this message. As a result $n \notin [0,2]$

β. $\mathbf{n = 3}$

If $n = 3$, then $s_m^j[n]$ is the last bit of the array that corresponds to the scope revoke. $\mathcal{ADV}$ would commence the attack by sending $m_{11} = \langle r_{11}, \mathsf{E}_{\mathsf{pk_{REV}}}\big(u_m, u_\ell, n_\ell, c_P^{\mathsf{K_j}}\big), \sigma_m\big(H\big(r_{11} || u_m || u_\ell || n_\ell || c_P^{\mathsf{K_j}}\big)\big)\rangle$ (n_ℓ is the index of the one dimensional array s_ℓ^j specifying which scope to be revoked for u_ℓ) to REV. REV will check the integrity and the freshness of the message and will also recover the identities of u_m and u_ℓ. At this point, REV will retrieve the list of scopes L_{VS} and will check whether $s_m^j[3] = 1$ or not. Since the adversary can not tamper with L_{VS}, REV will see that $s_m^j[3] = 0$ and the attack will fail.

As a result, the attack will fail $\forall n \in [0,3]$ and this concludes our proof. □

Proposition 2 (RLUA Soundness)**.** *Let $\mathcal{ADV}$ be a malicious adversary that corrupts a user u_m with access rights $\mathcal{SC}_m$. Furthermore, let u_ℓ be a legitimate user, with access rights $\mathcal{SC}_\ell \neq \mathcal{SC}_m$. Moreover, we assume that $\exists\, j : (j, s_\ell^j) \in \mathcal{SC}_l$, and $(j, s_m^j) \notin \mathcal{SC}_m$. Then $\mathcal{ADV}$ cannot successfully perform an RLUA attack.*

Proof. User u_m launches an attack to u_ℓ by sending $m_{11} = \langle r_{11}, \mathsf{E}_{\mathsf{pk_{REV}}}(u_m, u_\ell, n, c_p^{\mathsf{K_j}}), \sigma_m\big(H(r_{11} || u_m || u_\ell || n || c_p^{\mathsf{K_j}})\big)\rangle$ to REV. Upon reception, REV checks the integrity and the freshness of the message. Since this message can be constructed by anyone, REV proceeds by contacting KT to receive $\mathsf{idx}_{\mathsf{K_j}}$. Upon reception of the index, REV retrieves the list L_{VS}

to check whether $\langle \mathsf{idx}_{\mathsf{K_j}}, \{u_m, s_m^j[3] = 1\}\rangle \in L_{VS}$ or not. However, since $(j, s_m^j) \notin \mathcal{SC}_m$, $s_m^j[3] \neq 1$ the verification fails and $\mathcal{ADV}$ cannot successfully launch the attack.

□

Proposition 3 (CRUA Soundness). *Let $\mathcal{ADV}$ be a malicious adversary that corrupts a user u_m whose access to data encrypted under $\mathsf{K_i}$ has been revoked (i.e. $u_m \in \mathcal{R}_{\mathsf{K_i}}$). Then $\mathcal{ADV}$ cannot successfully perform a CRUA attack.*

Proof. $\mathcal{ADV}$ can successfully perform such an attack if and only if the following conditions hold:

α. **$\mathcal{ADV}$ can access the symmetric key $\mathsf{K_i}$.**

Condition α is always true. Since $u_m \in \mathcal{R}_{\mathsf{K_i}}$, u_m used to have access to the data encrypted with $\mathsf{K_i}$.

β. **$\mathcal{ADV}$ can bypass the authentication of the different components of the system model.**

For condition β to hold, the CSP needs to be convinced that $u_m \notin \mathcal{R}_{\mathsf{K_i}}$. To do so, $\mathcal{ADV}$ must generate a valid m_8 message which can be done with the following two ways:

- **Replay of an old message:** A first approach for $\mathcal{ADV}$ would be to try to replay old messages. To this end, she sends the m_8 message she received from KT during the $\mathsf{MSCOPE.KeyShare}$ algorithm to the CSP. Since m_8 used to be valid, $\exists\, n : s_m^i[n] \neq 0$. Moreover, the message contains a valid signature from KT. However, the timestamp contained in this message is not fresh and the verification will fail. An alternative for u_m would be to get a fresh valid m_8 message. To do so, she can forward m_7 to KT. However, m_7 is not fresh since the included timestamp would have expired. As a result, KT will not proceecd with the generation of a new m_8 message.
- **Impersonate a legitimate user:** Another approach for u_m would be to impersonate a registered user u_ℓ, such that $s_\ell^i \neq [0000]$. We assume that u_m obtains message $m_8 = \left\langle \mathsf{E}_{\mathsf{pk_{CSP}}}(u_\ell, t, \mathsf{idx}_{\mathsf{K_i}}, s_\ell^i), c_p^{\mathsf{K_i}}, \sigma_{KT}\left(H\left(u_\ell || t || \mathsf{idx}_{\mathsf{K_i}} || c_p^{\mathsf{K_i}} || s_\ell^i\right)\right)\right\rangle$ and tries to tamper with it. However, without knowing the index of the encryption key $\mathsf{idx}_{\mathsf{K_i}}$, she cannot alter the first part of the message and replaces u_ℓ with u_m. Moreover, since u_ℓ's id is also contained in the second part of the message, swapping the two identities is equivalent to forging KT's signature, which can only happen with negligible probability. Therefore, the attack will again fail.

Hence, only one of the two conditions holds. Therefore, the attack fails. □

8 Experimental Results

Our experiments mainly aimed at analyzing the processing time of MicroSCOPE. For the implementation of the CP-ABE scheme, we used the library provided by Bethencourt et al. [BSW07] while for the SSE scheme we used the Dynamic SSE scheme described in [BM19b] – a dynamic forward private SSE scheme which is a variant of the one presented in [EKPE18]. Furthermore, for the implementation of the parts that run in secure enclaves we used the SGX-OpenSSL library [SGX17].

Further to the above mentioned, since we wanted to evaluate the performance of MicroSCOPE under realistic conditions, we decided to use different machines – depending on the process to be measured. To this end, the setup of the SSE scheme was measured in

Table 1: Size of Datasets and Unique Keywords

No of TXT Files	Dataset Size	Unique Keywords
425	184MB	1,370,023
815	357MB	1,999,520
1,694	670MB	2,688,552
1,883	1GB	7,453,612
2,808	1.7GB	12,124,904

a Microsoft Surface Book laptop with a 2.1GHz Intel Core i7 processor and 16GB RAM running Windows 10 64-bit. The reason for measuring this phase on a laptop is that in a practical scenario, this process would take place on a user's machine. Hence, conducting the experiments on a powerful machine would result in a set of non-realistic measurements. The parts running in an enclave were measured in a powerful desktop PC with an Intel Core i7-8700 at 3.20GHz (6 cores), 32GB of RAM and 512GB SSD running Ubuntu 18.04 Desktop operating system, compiled with 64-bit and Intel SGX Hardware Debug mode build configurations. The reason for running these parts on such a computer is based on the assumption that these processes will be running on the CSP (i.e. a powerful machine with even more resources than the ones used in our testbed).

8.1 Symmetric Searchable Encryption

This part of the experiments was implemented in Python 2.7 using the PyCrypto [PyC13] library. To test the overall performance of the underlying SSE scheme, we used files of different size and structure. More precisely, we selected random data from the Gutenberg dataset [Gut71]. Our experiments on the SSE scheme were focused on two main aspects: *(1)* Indexing and *(2)* Searching for a specific keyword. Additionally, our dictionaries were implemented as tables in a MySQL database.

Dataset: For the needs of our experiments, we created five different datasets with random text files (i.e. e-books in .txt format) from the Gutenberg dataset. The datasets that we selected ranged from text files with a total size of 184MB to a set of text files with a total size of 1.7GB. At this point, it is worth mentioning that using text files (i.e. pure text in comparison to other formats such as PDF, word, excel, etc.) resulted in a very large number of extracted keywords – thus creating a dictionary containing more than 12 million distinct keywords (without counting the stop words). Furthermore, in our implementation we also incorporated a stop words (such as the, a, an, in) removal process. This is a common technique used by search engines where they are programmed to ignore commonly used words both when indexing entries for searching and when retrieving them as the result of a search query. This makes both the searching and indexing more efficient while also reducing the total size of the dictionary. Table 1 shows the five different datasets that we used for our experiments as well as the total number of unique keywords that were extracted from each of the incorporated datasets.

Indexing & Encryption: The indexing phase is considered as the setup phase of the SSE scheme. During this phase the following three steps take place: *(1)* reading plaintext files and generating the dictionary, *(2)* encrypting the files, and *(3)* building the encrypted

Table 2: Keywords and Filenames pairs

Unique Keywords	**(w, id) Pairs**
1,370,023	5,387,216
1,999,520	10,036,252
2,688,552	19,258,625
7,453,612	28,781,567
12,124,904	39,747,904

indexers. In our experiments, we measured the total setup time for each one of the datasets shown in table 1. Each process was run ten times and the average time for the completion of the entire process was measured. Figure 1 illustrates the time needed for indexing and encrypting text files ranging from 184MB to 1.7GB that resulted to a set of more than 12 million unique keywords. As can be seen from figure 1, to index and encrypt text files that contained 1,370,023 distinct keywords the average processing time was 22.48min while for a set of files that resulted in 12,124,904 distinct keywords the average processing time was 203.28min. Based on the fact that this phase is the most demanding one in an SSE scheme the time needed to index and encrypt such a large number of files is considered as acceptable not only based on the size of the datasets but also based on the results of other schemes that do not offer forward privacy [DMNP17] as well as on the fact that we ran our experiments on a commodity laptop and not on a powerful server. Hence, it is evident that the selected SSE scheme can be easily adopted by our protocol.

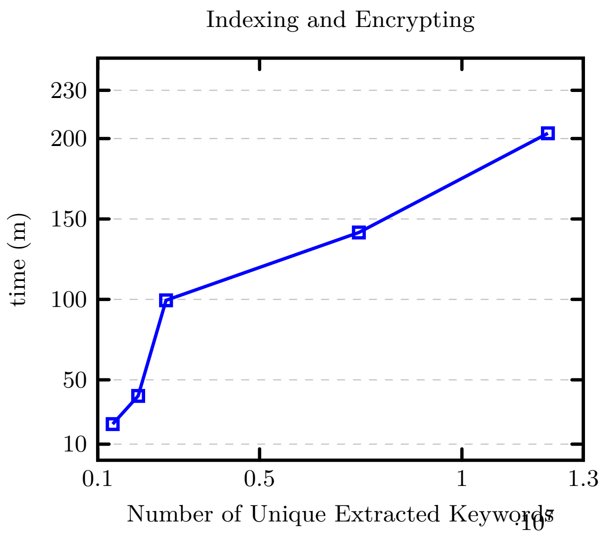

Figure 1: Indexing and Encrypting Files

Apart from the generation of the indexer that contains the unique keywords, the incorporated SSE scheme also creates an indexer that maintains a mapping between a keyword (w) and the filename (id) that w can be found at. The total number of the generated pairs in relation to the size of the underlying datasets is shown in table 2.

Search: In our implementation, the search time is the sum of the time that takes to generate a search token and the time needed to find the corresponding matches at the database. On average the time needed to generate the search token is $9\mu s$ while the actual matching of the files that contain the keyword that is being searched is just a simple `SELECT` and `UPDATE` query to the database. More precisely, searching for a specific keyword over a set of 12,124,904 distinct keywords and 39,747,904 addresses required 3.2sec on average. Even though this time is considered as high for just one search it is important to mention that this process will be running on a CSP with a large pool of resources and computational power. Hence, this time is expected to drop significantly on such a computer where more cores will be also utilized.

8.2 Implementation/Evaluation of MicroSCOPE

In this part of our experiments, we linked the enclaves to an SGX OpenSSL cryptographic library [SGX17], which was used to implement the RSA cryptosystem (i.e. RSA encryption/decryption/ and RSA signature creation/verification) with 4096-bit key sizes, and a set of cryptographic hash functions. All implementations were developed in C using Intel(R) SGX SDK 2.6 for Linux [SGX19].

In an SGX environment there are two main components: *(1)* the trusted component (enclave), and *(2)* the untrusted component (application). The untrusted application makes a call to an enclave by using SGX's `ECall` function which allows the application to enter the enclave. To temporarily exit the enclave and call a function in untrusted space, SGX's function `OCall` is used. For the needs of our experiments, we used both the `ECall` and `OCall` functions of SGX.

Enclave Creation & Key Generation: First, we measured the time needed to launch the four enclaves (MS, CSP, KT, and REV). Each enclave contains a different set of functions that corresponds to different parts of the protocol. We launched each enclave 10,000 times and measured the average completion time. The time to launch the MS enclave, containing functions for the generation of fresh RSA key pairs, was 25.29ms while the time to launch the REV enclave, containing MSCOPE.KeyShare and MSCOPE.Revoke functions, was 27.19ms. Time required to launch the KT enclave, containing MSCOPE.KeyShare, MSCOPE.Revoke and MSCOPE.KeyTrayStore functions, was 28.3ms while for the CSP enclave that contained MSCOPE.Store, MSCOPE.Search, MSCOPE.Update, and MSCOPE.Delete functions, 28.12ms was required. Since the process of launching enclave can run in parallel, the average time required by MicroSCOPE to launch all four enclaves is 28.3ms. Table 3 summarizes the results by presenting the time needed to launch each enclave of MicroSCOPE as well as the specific functions from the protocol that each enclave contains. Additionally, we also measured the time needed to launch an empty enclave. Launching an empty enclave took on average 9.2ms. Even though launching the enclaves of MicroSCOPE needed 28.3ms the difference of 19ms is considered as negligible considering that the setup of MicroSCOPE runs only once.

Each enclave generates an RSA key pair of 4096-bit length. As can be seen in table 3 the average time to generate an RSA key pair of 4096-bit size was 840ms. Again, this is a process that can run in parallel. As a result, the total time required for a complete setup of the enclaves is estimated at 921.71ms (i.e. less than a second) – time that is considered as negligible.

Enclave Attestation: Attestation is the process by which an entity demonstrates that its software runs on the SGX-enabled platform. The Intel SGX platform supports two forms of attestation: Local (Intra-Platform) attestation, and Remote (Inter-Platform) attestation [Int15]. Local attestation involves two or more enclaves running on the same

Table 3: Setup time for main MSCOPE components

Enclave Creation	MSCOPE Functions	Time
RSA Setup	Average time for generating a 4,096 bit long RSA key pair	840ms
EMPTY Enclave	Average time for launching an empty enclave	9.2ms
MS Encalve	Containing MSCOPE's key generation functions	25.29ms
REV Enclave	MSCOPE.KeyShare	27.18ms
KT Enclave	MSCOPE.KeyShare MSCOPE.KeyTrayStore MSCOPE.Revoke	28.3ms
CSP Enclave	MSCOPE.Store MSCOPE.Search MSCOPE.Update MSCOPE.Delete	28.12ms
Local Attestation	KT & REV	1.1ms

platform, whereas the Remote attestation involves an enclave on platform that can be verified by a remote third party. While in theory this verification can be done by any third party, it currently demands to contact the Intel Attestation Server – a process that requires a license from Intel. As a result, in our experiments we only measured the time needed to perform Local Attestation between enclaves. To perform Local attestation, one enclave requires another enclave to prove its identity and integrity by sending a hardware generated report. The report includes a cryptographic proof that the attested enclave exists on the local platform and its configuration can be trusted. The report includes: *(1)* Measurements of the code and data in the enclave; *(2)* A hash of the public key in the ISV certificate presented at the time of enclave initialization; *(3)* User data; *(4)* Other security related state information; and *(5)* A signature block over the above data, which can be verified by the same block that produced the report [Int15]. In our experiments, we measured the time taken for REV enclave to attest with KT enclave. We ran this local attestation 10,000 times and calculated the average time needed to successfully complete the process. The average time needed for a successful local attestation between KT and REV was 1.1ms.

Measuring the Execution Time: In the next phase of our experiments, we measured the total execution time of MicroSCOPE. To do so, we explicitly measured the running time of protocol's core functions by calculating the time time needed to generate, send and verify the messages that are involved in the protocol. This allowed us to evaluate the actual performance and efficiency of MicroSCOPE. We ran each function of MicroSCOPE 100,000 times and then calculated average execution time. Each MSCOPE function consists of an `ECall`, an `OCall` or both. In our experiments, our primary focus was to measure the execution time of all the involved `ECalls` and `OCalls`. Figure 2 summarizes the results of this experiment by showing the average processing time needed for each one of the core MicroSCOPE's functions. As can be seen in Figure 2, MSCOPE.KeyShare and MSCOPE.Revoke are the two most demanding processes. MSCOPE.KeyShare needed

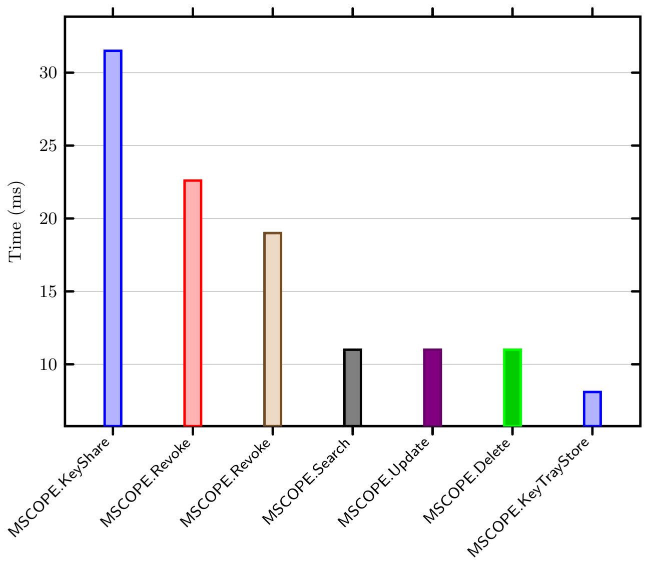

Figure 2: Message Creation and Verification

on average 31.5ms while 22.6ms was the execution time of MSCOPE.Revoke. However, the running time of MSCOPE.Revoke also depends on the length of the revocation list. Currently, we only measured the time to construct, send and verify the required messages. Moreover, the execution time for MSCOPE.Store was measured at 19ms. The corresponding times needed for MSCOPE.Search, MSCOPE.Update and MSCOPE.Delete appeared to be the same – 11ms on average. Finally, MSCOPE.KeyTrayStore appeared to be the lightest function with an average execution time of 8.1ms.

8.3 Ciphertext-Policy Attribute-Based Encryption

In the last part of our experiments, we measured the encryption and decryption time for the underlying CP-ABE scheme. MSCOPE only uses CP-ABE to encrypt a symmetric key and *not* large volumes of data. To this end, we measured the time needed to encrypt and decrypt a symmetric key under policies of different sizes. We used access policies of the type '`Attribute_1 AND Attribute_2 AND ...AND Attribute_n`' as in [GHW11] and [AC17]. Such policies are the most demanding since all attributes are required for the decryption. For the encryption of a file with a policy consisting of the conjunction of 1000 attributes the time needed was 11sec, while the corresponding decryption time was measured at 4.5sec. However, encryption and decryption times of the CP-ABE scheme depends on the size of the policies. As a result, for a realistic scenario in which the policy consists of 200 attributes, the encryption and decryption times were measured at 2.1sec and 0.6sec respectively. Thus, the use of CP-ABE scheme does not put any real computational burden to the performance of MSCOPE.

At this point it is important to mention that in order to present a better and more accurate overview of the MicroSCOPE performance, the results presented in subsections 8.1, 8.2 and 8.3 need to be combined. However, this is a rather demanding procedure that will require us to build a proper client-server setup that will allow us to run experiments in a realistic environment similar to the one offered by existing cloud-based services. At this point, we leave this for future work.

9 Conclusion

In this paper, we proposed MicroSCOPE, 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. Our construction allows data owners to share their data based on certain access rights that can assign to users. Furthermore, we have shown how to rely on SGX to provide an efficient revocation mechanism that is agnostic to the underlying encryption schemes. Finally, we believe that this work can provide the basis for secure data sharing between organizations that use independent cloud platforms. Hence, one of our future goals is to test our construction in a multi-cloud environment.

References

[AC17] Shashank Agrawal and Melissa Chase. Fame: Fast attribute-based message encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 665–682, New York, NY, USA, 2017. ACM.

[AIKM16] Ioannis Askoxylakis, Sotiris Ioannidis, Sokratis Katsikas, and Catherine Meadows. *Computer Security – ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*, volume 9878. 01 2016.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 440–456. Springer, 2005.

[BGK08] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 417–426, New York, NY, USA, 2008. ACM.

[BM19a] Alexandros Bakas and Antonis Michalas. Modern family: A revocable hybrid encryption scheme based on attribute-based encryption, symmetric searchable encryption and sgx. Cryptology ePrint Archive, Report 2019/682, 2019. https://eprint.iacr.org/2019/682.

[BM19b] Alexandros Bakas and Antonis Michalas. Multi-client symmetric searchable encryption with forward privacy. Cryptology ePrint Archive, Report 2019/813, 2019. https://eprint.iacr.org/2019/813.

[Boy99] Victor Boyko. On the security properties of oaep as an all-or-nothing transform. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 503–518, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[CD16] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016.

[DMNP17] Rafael Dowsley, Antonis Michalas, Matthias Nagel, and Nicolae Paladi. A survey on design and implementation of protected searchable data in the cloud. *Computer Science Review*, 2017.

[DY83] Danny Dolev and Andrew C Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2), 1983.

[EKPE18] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(1):5–20, 2018.

[FBB$^+$17] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. Hardidx: Practical and secure index with sgx. In Giovanni Livraga and Sencun Zhu, editors, *Data and Applications Security and Privacy XXXI*, pages 386–408. Springer, 2017.

[FVBG17] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. Iron: Functional encryption using intel sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 765–782. ACM, 2017.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 34–34, Berkeley, CA, USA, 2011. USENIX Association.

[Gut71] Project gutenberg, 1971.

[Int15] Intel's software guard extensions: Enclave writer's guide, 2015.

[LSG$^+$17] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, 2017. USENIX Association.

[LYZL18] Joseph K. Liu, Tsz Hon Yuen, Peng Zhang, and Kaitai Liang. 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.

[Mic18] Antonis Michalas. The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing. Cryptology ePrint Archive, Report 2018/1204, 2018. https://eprint.iacr.org/2018/1204.

[Mic19] Antonis Michalas. 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*, SAC '19. ACM, 2019.

[MS18] Steven Myers and Adam Shull. Practical revocation and key rotation. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, pages 157–178. Springer, 2018.

[PGM17] N. Paladi, C. Gehrmann, and A. Michalas. Providing user security guarantees in public infrastructure clouds. *IEEE Transactions on Cloud Computing*, 5(3):405–419, July 2017.

[PyC13] PyCrypto – the Python cryptography toolkit, 2013.

[SGX17] Sgx-openssl, 2017.

[SGX19] Sgx-sdk, 2019.

[XCP15] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 640–656, Washington, DC, USA, 2015. IEEE Computer Society.