Project title: Enforceable Security in the Cloud to Uphold Data Ownership
Project acronym: ESCUDO-CLOUD
Funding scheme: H2020-ICT-2014
Topic: ICT-07-2014
Project duration: January 2015 – December 2017

D2.6
Final Report on Data Protection and Key Management Solutions

Editors: Mathias Björkqvist (IBM)
Christian Cachin (IBM)
Björn Tackmann (IBM)
Reviewers: Neeraj Suri (TUD)
Sabine Delaitre (WT)

Abstract

This deliverable provides an integrated view of the contributions and findings that were produced by Tasks 1–4 of WP2, addressing the solutions for data protection and key management. WP2 is responsible for the technical work on the design and implementation of protection solutions for the management of outsourced data. Such techniques allow a data owner to ensure confidentiality, integrity, and availability of her data, and perform fine-grained data access. These solutions form the basis of the tools applied in the Use Cases.

Type	Identifier	Dissemination	Date
Deliverable	D2.6	Public	2017.10.31

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644579. This work was supported in part by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract No 150087. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission or the Swiss Government.

ESCUDO-CLOUD Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	British Telecom	BT	United Kingdom
3.	EMC Corporation	EMC	Ireland
4.	IBM Research GmbH	IBM	Switzerland
5.	SAP SE	SAP	Germany
6.	Technische Universität Darmstadt	TUD	Germany
7.	Università degli Studi di Bergamo	UNIBG	Italy
8.	Wellness Telecom	WT	Spain

Disclaimer: The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The below referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Versions

Version	Date	Description
0.1	2017.10.05	Initial Release
0.2	2017.10.26	Second Release
1.0	2017.10.31	Final Release

List of Contributors

This document contains contributions from different ESCUDO-CLOUD partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	IBM
Chapter 1: Protection of data at rest	Enrico Bacis (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Marco Rosa (UNIBG), Ali Sajjad (BT), Pierangela Samarati (UNIMI)
Chapter 2: Key-management solutions	Mathias Björkqvist (IBM), Christian Cachin (IBM), Björn Tackmann (IBM)
Chapter 3: Efficient and private data access	Enrico Bacis (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Marco Rosa (UNIBG), Pierangela Samarati (UNIMI)
Chapter 4: Requirements-based threat analysis	Ahmed Taha (TUD), Patrick Metzler (TUD), Salman Mazoor (TUD), Neeraj Suri (TUD)
Chapter 5: Conclusion	All Partners

Contents

List of Figures

Executive Summary

The overarching goal of ESCUDO-CLOUD is to protect outsourced data, and to empower data owners to take control of their data. ESCUDO-CLOUD considers data security from two perspectives. On one hand, Cloud providers employ security mechanisms for protecting data in storage, processing, and communication, and devote resources to ensure security for customer data. On the other hand, a data owner who relies on the Cloud loses control over the data and their processing, and needs mechanisms to protect herself against the exposure of her data to threats. As the goal of ESCUDO-CLOUD is to empower data owners, it provides effective and deployable solutions allowing data owners to maintain control over their data when relying on Cloud providers for data storage, processing, and management.

This deliverable provides an integrated view of the contributions and findings that were produced by Tasks T1–T4 of WP2, addressing the solutions for data protection and key management. WP2 covers the technical work on the design and implementation of protection solutions for the management of outsourced data. Such techniques allow a data owner to ensure confidentiality, integrity, and availability of her data, and perform fine-grained data access. These solutions form the basis of the tools applied in the Use Cases.

T2.1 Protection of data at rest (M1–M24). This task developed the high-level requirements and an architecture for client-side encryption of data in a Cloud object storage service, as provided by OpenStack Swift. An implementation of this has been developed in the form of the EncSwift tool. This tool provided protection of data at rest with the use of client-side encryption. Catalogs were used to manage encryption keys. Furthermore, T2.1 provided an in-depth look at the key-management options for the client proposed architectures, and introduced the use of Barbican as key storage. The final released version of EncSwift makes use of Barbican as key storage and lets the user decide what over-encryption scenario to enforce in order to provide efficient policy update. EncSwift has also been integrated with other key managers, as described in this document.

T2.2 Key-management solutions (M4–M34). Within Task T2.2, the server-side encryption and key-management functions of the OpenStack Swift Cloud storage system have been addressed. In particular, ESCUDO-CLOUD has contributed to the data-at-rest encryption and key management features included in OpenStack Swift distribution, implementing server-side encryption and integration with Cloud-based key managers. Furthermore, the design and a prototype implementation of key rotation and secure deletion in the OpenStack Swift Cloud object storage system were developed. As another contribution, a scalable key management for distributed Cloud storage was developed and a prototype has been integrated with IBM Spectrum Scale. It demonstrated linear scalability even under load from concurrent key updates.

T2.3 Efficient and private data access (M4–M34). This task aims to develop solutions for effectively and efficiently accessing data stored at an external Cloud provider, while maintaining confidentiality of access operations. The developed solutions are based on dynamically allocated data structures, that is, on continuous data re-allocation at each access. T2.3 specifically analyzed and extended the shuffle index approach and proposed an alternative data structure, based on a dynamically allocated binary tree, that does not rely on client-side storage. Task T2.3 also studies integrity in Cloud scenarios. The task developed a novel encryption mode (mixing) that provides strong mutual inter-dependency in the encrypted representation of each resource. It can be used by the data owner to verify the integrity of outsourced data.

T2.4 Requirements-based threat analysis (M4–M34). This task utilizes the ESCUDO-CLOUD Use Cases (UCs) requirements to conduct a Requirements Based Threat Analysis (RBTA) by analyzing the threats whose impact results in a violation of the data ownership principles as related to the ESCUDO-CLOUD UCs. T2.4 first captures the specific data owner requirements on functionality, performance and security. Then, it assesses the threat violations by their type, impact and also the likelihood of their occurrence. Furthermore, T2.4 provides an input for estimating the risks based on such requirement analysis of the data owner. The manual RBTA process was able to establish the viability of identifying threats based on a UC-level requirements analysis. However, the manual process is both time intensive and makes no assurance on completeness of the analysis. Thus, building upon the basic viability of the RBTA result, the task developed a systematic analytical technique that enumerates the set of UC requirements and then determines all possible direct/indirect dependencies across them to conduct a generalized threat analysis from their requirements. The initial UC-specific RBTA approach was also extended to apply to generic UCs via the development of a generalized Cloud threat model.

The described solutions are summarized within this deliverable by their corresponding tasks. Several transfers from research to exploitation have already been realized. Particularly, Use Case 1 led by IBM as well as Use Case 3 led by BT benefit from the work in WP2.

1. Protection of data at rest

T2.1 deals with protection of data at rest. The goal is to focus on the support for confidentiality and integrity in a scenario where only basic upload and download operations are available over objects and where encryption is applied at the client side. The assumption on the application of encryption at the client side is one of the crucial aspects that characterizes the work in ESCUDO-CLOUD and the results of the work in WorkPackage WP2 demonstrate the applicability of this assumption to several scenarios. Most of the work in Task T2.1 has considered the OpenStack Swift system, which is one of the most successful platforms for object storage in the Cloud. The specific innovations are described in the next section. A significant contribution of the work in this task has been the development of the EncSwift tool, which is described in Section 1.2.

1.1 ESCUDO-CLOUD Innovation

Several advancements over the state-of-the-art have been achieved by carrying out Task T2.1.

- Extension of OpenStack Swift with over-encryption: we investigated the current status of OpenStack Swift, and identified several scenarios for EncSwift application, i.e., *trusted client*, *light trusted client*, *partially trusted CSP* and *Multi-CSP*. For each scenario, we analyzed the high-level requirements and proposed an architecture. We also introduced a policy evolution approach based on Over-Encryption [DFJ$^+$07], and identified three different approaches (i.e., *on-the-fly*, *on-resource* and *end-to-end*) that can be adopted in order to enforce it. These results are reported in Deliverable D2.1 and Deliverable D2.2.

- Design of the EncSwift tool: the considerations presented above were the basis for the realization of the first version of the EncSwift tool. This tool was based on one of the architectures we proposed, and provided protection of data at rest with the use of client-side encryption. This required the introduction of the concept of *catalog* in order to manage keys. A catalog is a json file stored in a meta-container in Swift. Every user is associated with a catalog that stores all the keys owned by the user (in an encrypted form). We used the catalogs in order to manage encryption keys. This work is also reported in Deliverable D2.2.

- Integration of EncSwift with Barbican: we analyzed in depth the proposed client architectures, and introduced the use of Barbican as key storage. We implemented all the approaches for providing secure policy update, and made an experimental evaluation on them. We also released a final version of the EncSwift tool that makes use of Barbican as key storage and lets the user decide what Over-Encryption scenario to enforce in order to provide efficient policy update. This work is reported in Deliverable D2.5.

- Integration of EncSwift with BT KMS: a further innovation of T2.1 is introduced in this Deliverable D2.6, where we illustrate the integration of the EncSwift tool with a new key manager provided by BT.

The results obtained by this task have been published in [BdVF$^+$16, BDF$^+$16a, BRS17].

1.2 EncSwift and Key Management: An Integrated Approach in an Industrial Setting

A clear trend of users and organizations for storing data is moving them to the Cloud. This provides great advantages, mostly in terms of cost and availability, but also introduces major concerns with regard to confidentiality. This becomes pivotal when considering the Cloud provider as honest-but-curious, i.e., it behaves as expected but it cannot be trusted for accessing the content of data it stores. An efficient approach for dealing with an honest-but-curious provider is the use of encryption for protecting data before outsourcing them to an external Cloud provider [BDF$^+$16a, WLOB09]. In this way, data are protected both against privacy interfering providers and adversaries who may exploit vulnerabilities of the provider in order to gain access to the physical representation on disks. Moreover, the use of encryption offers the opportunity to effectively enforce access control by encrypting data with different keys, each of them shared with users according to an authorization policy [DFJ$^+$10b], [DFJ$^+$10a]. EncSwift [BDF$^+$16a] is a tool for protecting data confidentiality in OpenStack Swift already introduced in [PBD$^+$16] and [FR17], that uses different encryption keys to protect resources, according to the policies ruling access on them, and providing in this way fine grained (i.e., file level) access control.

In a dynamic system with frequent policy updates, the use of encryption brings complexity since keys must be shared and revoked in an efficient and trusted way. In fact, as though granting access to a resource only requires to share its encryption key, access revocation is a more relevant problem, because the revoked user could have stored local copies of the keys and, if the access control enforced by the server is faulty, she potentially could still access the plain content of the resource. Nevertheless, forcing the owner of the resource to download and re-encrypt objects with a fresh key would add a unbearable overhead to the performance of the system. In this case, Over-Encryption [DFJ$^+$07] offers a solution, adopted also in the EncSwift tool, that can manage access revocation in an efficient way, applying a server-side encryption layer such that this new encryption key, generated by the server, cannot be known to revoked users. Over-Encryption would provide a clear benefit both in terms of performance and security, but requires an efficient key management service able to manage a potential large number of encryption keys.

The purpose of this new contribution for T2.1, analyzed in [BRS17], is to introduce a new efficient key management service, BT KMS, and investigate its integration with EncSwift, in a scenario with frequent policy updates, and thus with the generation and administration of a large number of encryption keys.

1.2.1 EncSwift

This section introduces the EncSwift approach, and briefly describes the OpenStack organization and its infrastructure, EncSwift principles for data confidentiality protection, and how the Over-Encryption approach can be introduced into EncSwift for managing policy updates.

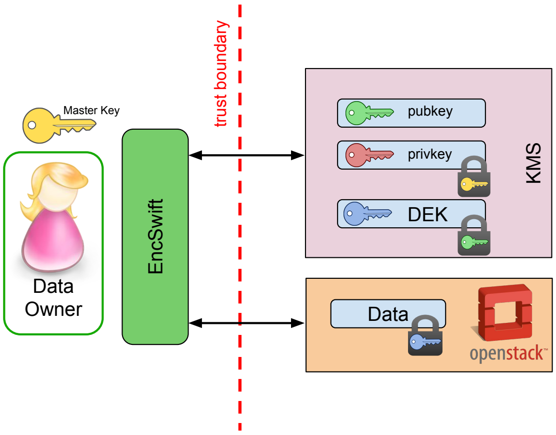


Figure 1.1: EncSwift architecture

OpenStack Swift

In this document, we assume that the user wants to outsource her data to a Cloud provider based on OpenStack [Ope]. OpenStack, originally developed by Nasa and Rackspace has become today the *standard-de-facto* for private Cloud. It offers a combination of open source tools for managing the core Cloud-computing services of compute, networking, storage, identity, and image services, and more other projects can be bundled together for customizing the platform based on need. In this document we focus on Swift, the object storage of OpenStack. Swift is designed to efficiently, safely, and cheaply store files, that can be accessed through a *tenant/container/object* structure. *Objects* correspond to files and are organized in *containers*, i.e., folders, whereas at the highest level of this hierarchy are the *tenants*, sets of containers that are usually assigned to an organization. Swift also defines two levels of Access Control Lists (ACLs): *tenant* and *container*. A tenant-level ACL defines users that can perform administrative operations on the tenant. A container-level ACL defines $read|write|listing$ permissions on the container. There is no specific object-level ACL: the container-level ACL also applies to the contained objects.

With regard to its internal structure, Swift it based on *WSGI* (Web Server Gateway Interface). This permits to define a *pipeline*, i.e., a chain of modules (the *middlewares*) that can enrich requests before they reach the main web server component (the Proxy Node), and that can modify the responses coming from it.

Protection of data confidentiality

EncSwift is a solution presented in [BDF$^+$16a] for enforcing data confidentiality and efficient access control in OpenStack Swift. EncSwift behaves as a transparent proxy, encrypting objects, with a symmetric key (DEK, *Data Encryption Key*) before uploading them to the Cloud provider, and decrypting them at download time. Each DEK is associated with a container, such that all the objects stored in the same container are encrypted with the same DEK, that must be shared along all the users that can access it. Every user is also associated with a signature key pair (for signing messages), and a RSA key pair, that can be used to asymmetrically encrypt the DEKs she owns, in order that she can store them in an encrypted form, i.e., as KEKs (*Key Encryption Keys*) Owing to the fact that users may want to access their data from several devices, a secret known only to the user is needed (MK, *Master Key*). All the other keys (both the RSA key pair and the KEKs) are stored in a key management service. The Master Key is therefore used to encrypt the

RSA and signature keys of the user. OpenStack already offers a secret storage, Barbican. Barbican stores secrets, that are equivalent to objects in Swift, and that can be organized in containers, just like Swift. Each Swift container then has a corresponding container in Barbican, storing both users' RSA and signature key pairs, and KEKs. Note that keys are stored in encrypted form in Barbican since we do not trust the Cloud provider for accessing the sensitive content of outsourced objects. Indeed, storing keys in plaintext in a Barbican container would put the confidentiality of outsourced data at risk. Even if keys are stored in encrypted form at the Cloud provider, in [BDF$^+$16a] the authors decided to use Barbican for their storage, in contrast to Swift, because it enforces additional security measures against outside attacks. The EncSwift architecture is shown in Figure 1.1.

Policy Update

Due to the adoption of policy-based encryption, a modification to the ACL of a container spreads to the encryption policy of that container. In fact, a policy update would require to change the encryption key for that container, but this is unfeasible since a user would be required to download the whole content of a container, re-encrypt it and re-upload everything. Over-Encryption [DFJ$^+$07] is the solution that EncSwift adopts in order to prevent this excessively onerous operation. The integration of Over-Encryption requires the addition of a layer of encryption. Indeed, a first layer of encryption (BEL, *Base Encryption Layer*) is applied at client side in order to provide data confidentiality against an honest-but-curious provider, enforcing the initial access control policy. A second layer of encryption (SEL, *Surface Encryption Layer*) is then applied server side in order to enforce policy updates. Policy updates can be of two kinds: grant and revoke. In case of a grant access to a container, the data owner has to share the encryption key used for protecting resources at the BEL level, i.e., the BEL DEK, with the new user, creating a new KEK and sharing it through the Key Management Service (KMS). In case of revoke, the objects in the container must be protected at the SEL level with a new DEK, generated by the server. In [BDF$^+$16a] three implementations for the Over-Encryption are illustrated. In this document we focus on the *on-the-fly* mode. This means that Over-Encryption is enforced every time a user downloads an object from that container, by generating a new SEL DEK to encrypt the resource on-the-fly. The Over-Encryption can be easily integrated in EncSwift by inserting a new custom middleware in the Swift pipeline, thus it requires a modification of the server. The SEL DEK permits to enforce a fine grained access control mechanism at container level. It is generated by the custom middleware at the first access to the container after a policy update, and stored as a KEK for every authorized user in the KMS. Figure 1.2 shows the architecture of a policy update.

The number of keys that Barbican must manage increases exponentially with the number of policy updates. This is the rationale behind the investigation on integrating a more efficient KMS in EncSwift.

1.2.2 Industrial Key Management for EncSwift

This section illustrates the contributions of this line of work. In Section 1.2.2 we describe a new industrial key management service, BT KMS, together with its architecture. In Section 1.2.3 we analyze the integration between this industrial key manager and EncSwift for protecting data confidentiality and enforcing access control.

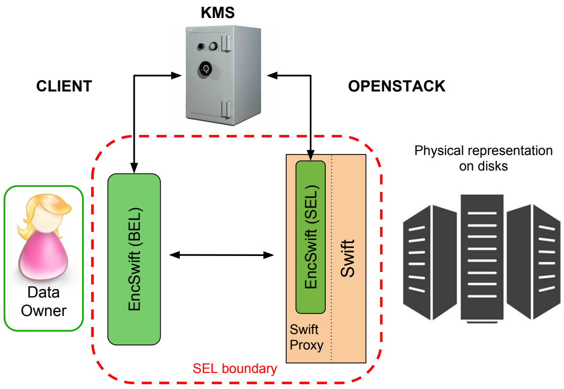


Figure 1.2: Over-Encryption management in EncSwift

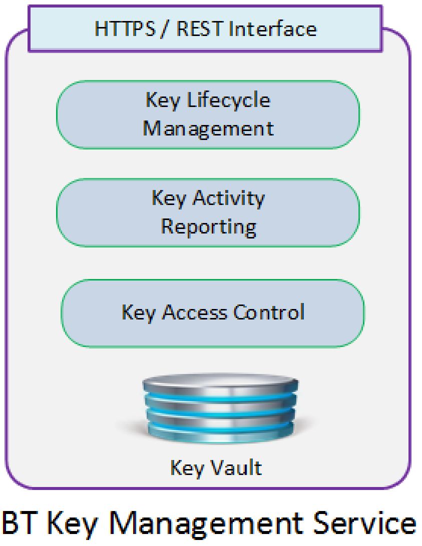


Figure 1.3: Architecture of the BT KMS

BT Key Management Service

The main purpose of the BT Key Management Service (KMS) is to allow users to securely manage their encryption keys and certificates, either in a Cloud based or in an on-premise environment. Furthermore, in a Cloud based setting, the BT KMS can be provisioned and managed via the BT Service Store [DDD$^+$16], which enables its users to create isolated and compartmentalized key management domains. This allows the users to manage their keys for use in multiple Cloud platforms in a secure and multi-tenant environment. The main components of the BT KMS are shown in Figure 1.3 and are described in more detail below.

- *HTTPS/REST Interface*: it offers an intuitive web-based management console for enterprise-scale administration, as well as Single Sign-On capabilities that can be seamlessly integrated with LDAP based user identity management services. It also supports management through the use of a full-featured REST based API.
- *Key Lifecycle Management*: it provides complete lifecycle management of the keys, including key creation, storage, import, export, rotation, revocation and deletion. Thus users have complete control over every aspect of a key during its existence. It also supports multiple

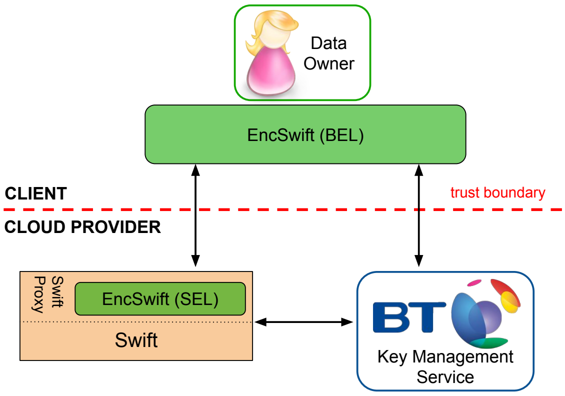


Figure 1.4: Architecture of EncSwift and BT KMS integration

API standards like PKCS#11, Microsoft Extensible Key Management (EKM) and OASIS KMIP, which helps with the automation of various operational tasks.

- *Key Activity Reporting*: it implements the ability to audit and report on all activities relating to keys, so that the users are able to generate comprehensive and granular audit logs of encryption key and certificate management activities. This is an essential requirement for a lot of compliance standards, and BT KMS complies with the FIPS 140-2 Level 1 standard.
- *Key Access Control*: lastly, it is tightly integrated with an access control capability that governs the rules and policies for releasing the relevant data encryption keys to the authorized users and processes. It also enforces the storage and retrieval to and from the key vault, which provides high availability storage and backup of the keys.

The key core requirements for users of Cloud based services are transparent control and ownership of their data in the Cloud eco-system. This can be realized by utilizing the BT KMS, as it enforces the user-based management of the cryptographic keys, so that only the users are able to authorize policy-based release of keys to trusted applications. Even the Cloud service provider on which the BT KMS is deployed has no view or control of the users' keys and other security credentials. This approach also keeps the keys separate from the data that they are protecting.

Each instance of the BT KMS contains a public/private key pair (RSA key pair), which is used to protect the Key Encrypting Keys (KEK) and other sensitive security objects stored within the Key Vault. The RSA key pair can be changed periodically without any impact to the keys and credentials being protected by it. A KEK is a random AES-256 key which is used to protect the Data Encryption Keys (DEK) while they are at-rest in the Key Vault or in transit from the BT KMS to the client application. The DEKs are AES-256 keys that are used for the actual encryption of the objects, and are always encrypted at-rest and in transit. This minimizes the exposure of the DEK, as well as the need for its frequent rotation.

1.2.3 Integration between BT KMS and EncSwift

We decided to analyze possible challenges and enhancements to the EncSwift solution in case of adoption of the BT KMS in place of Barbican. Obviously, a research tool such as EncSwift would benefit from the integration of a full real industrial key management service. Indeed, BT KMS is designed to bring good performance and to be scalable in a real industrial scenario. Moreover, thanks to its flexibility and the wide number of communication protocols it supports, BT

KMS would fully fit into the EncSwift architecture. However, the integration requires some implementation effort in order to fix the differences in the key usage. Both BT KMS and EncSwift requires to store KEKs. Yet, in EncSwift KEKs are produced as symmetric DEKs asymmetrically encrypted with the RSA keys of the users. On the contrary BT KMS supports encryption of DEKs with a symmetric KEKs, that are then encrypted with the RSA keys of the users. An integration of the two architectures requires to choose one of the key treatment alternatives.

In our analysis, we chose to adopt the EncSwift key structure, and to deprecate the use of BT KMS encrypting keys, i.e., DEKs are asymmetrically encrypted with the RSA keys of the users. An architectural overview of the integration is shown in Figure 1.4. EncSwift still relies on an OpenStack Swift object storage, but all the encryption keys are stored at the BT KMS instead of Barbican. Moreover, the EncSwift SEL middleware implemented to enforce Over-Encryption can communicate with BT KMS in order to store and retrieve the SEL DEK.

1.3 Summary

This chapter described how EncSwift can be integrated with BT KMS. BT KMS is a solution for key management services developed by BT and already adopted in real industrial settings. Thanks to BT KMS flexibility and well structured architecture, we investigated the realization of a system combining BT KMS and the EncSwift tool [BDF+16a], which implements a solution for protecting data stored in OpenStack Swift. EncSwift guarantees data confidentiality and integrity, also naturally regulating - via encryption - access to them. With frequent policy updates, there is a trend to generate a large number of encryption keys. In [PBD+16] and [FR17], EncSwift stored them in Barbican, that is, OpenStack secret storage. With the investigation shown in this chapter, we have demonstrated the open and composable structure of the tools developed in the project.

2. Key-management solutions

Key-management systems are the single most critical part of any cryptographic security infrastructure. All uses of cryptography for protecting data rely on a proper implementation of key management. Key management deals with the complete lifecycle of cryptographic keys, with operations for creating, importing, storing, reading, updating, exporting, and deleting them, and with distributing keys before they are used in cryptographic functions. Key-management systems address different types of cryptographic objects, such as keys and secrets. These objects typically include symmetric keys, private keys, public keys, and certificates. An important aspect is to manage the attributes of keys that govern their usage and their relation to other keys. In a complex storage system, multiple processes acting on behalf of one entity may access the key-management system concurrently and perform operations on the maintained keys.

Task 2.2 (Key-management solutions) of ESCUDO-CLOUD addresses the management of cryptographic keys and other secret material used by the cryptographic protection systems in Cloud platforms. Particular focus is placed on the interplay between the requirements of the service provider and its users, on the feasibility of key management offered as a Cloud service, and on supporting secure deletion as a feature.

The work in Task 2.2 is related to Task 2.1 (Protection of data at rest), which is described in the previous chapter. Both tasks address protection of data-at-rest in a Cloud storage system, particularly in OpenStack Swift object storage. The tasks differ in the assumed security model: Task 2.1 always considers the Cloud storage provider as untrusted, keeps the keys with the clients, and operates outside the Cloud premise. In other words, the encryption in Task 2.1 acts as a proxy between the client and the Cloud. In contrast, the key management and enterprise-integration work reported in Task 2.2 here considers the encryption *within* a Cloud platform. It still permits to reduce the trust placed in the Cloud provider because the keys are not always available together with the data, and may never be stored in the clear by the Cloud provider. See the discussion in Deliverable D2.1 (Section 1) for background.

2.1 ESCUDO-CLOUD Innovation

The key contributions developed in the context of ESCUDO-CLOUD are as follows:

- Development of data-at-rest encryption and key-management features for OpenStack Swift, which are included in the distribution and target server-side encryption and integration with Cloud-based key managers.

- Design and implementation of key rotation and secure deletion in the OpenStack Swift Cloud object storage system. This feature has been released as open source.

- Scalable key management for distributed Cloud storage. A prototype was integrated with IBM Spectrum Scale and linear scalability was achieved even under load from concurrent key updates.
- New provable-security definitions and encryption schemes that allow for key rotation with stronger security guarantees. The discussed schemes achieve better security compared to those implemented above, but are less efficient to implement.

This work has been documented in public designs and forthcoming publications [BCES17, LT17].

2.2 Key management and at-rest encryption in contributions to OpenStack Swift

In close collaboration with developers of OpenStack Swift, consisting of a team at HP, HP Enterprise, IBM (mainly IBM Corp. and IBM Research – Zurich), and SwiftStack, the basic encryption functionality for OpenStack Swift has been implemented, tested, and released. The design is documented publicly:

```
https://specs.openstack.org/openstack/swift-specs/specs/in_progress/at_rest_encryption.html
```

This code has been included in the main OpenStack Swift distribution in July 2016 and has been available since. A blog post by John Dickinson, Director of Technology at SwiftStack and Project Technical Lead for OpenStack Object Storage describes the release:

```
https://www.swiftstack.com/blog/2016/07/12/at-rest-encryption-in-openstack-swift/
```

The design has also been described in ESCUDO-CLOUD Work Document W2.4. (M18).

Furthermore, code to integrate the data at-rest encryption inside the OpenStack Swift object storage system and the OpenStack Barbican key manager has been contributed and included in Swift. The native object encryption feature within Swift that has been released in 2016 assumed static keys and the encryption keys needed be provisioned in static files. With this integration to the key manager in OpenStack, standard key-lifecycle management functions have become available and may use a standardized API.

2.3 Key rotation and secure deletion for OpenStack Swift

In data storage, key rotation is considered good practice as it hedges against the impact of cryptographic keys being compromised over time. For instance, the Payment Card Industry Data Security Standard (PCI DSS) [PCI16], which specifies how credit card data must be stored in encrypted form mandates key rotation, meaning that encrypted data must regularly be moved from an old to a fresh key.

Going beyond a “flat” encryption model, the work within ESCUDO-CLOUD has produced a hierarchical key-management implementation, where higher-level keys (closer to the root of the hierarchy) are used to wrap lower-level keys (closer to the leaves). A single root-key wraps all its direct child keys that in turn can wrap their direct child keys and so on.

The result is a flexible *hierarchical key management* scheme, in which each object stored in Swift is protected with an individual key, and these object keys are wrapped by keys that are on higher levels in the hierarchy. Besides the fact that this structure aligns well with Swift's hierarchy of accounts, containers, and objects, one main advantage of this approach is its flexibility. In particular, it allows for *updating the keys* in some branch of the hierarchy, while leaving the remaining part untouched. This efficient key-update operation is instrumental in implementing *secure deletion* in our data-at-rest protection in Swift.

This feature has been developed in for OpenStack Swift and is described in ESCUDO-CLOUD Work Document W1.3 (M30).

2.4 Scalable key management for distributed Cloud storage

As use of cryptography increases in all areas of computing, efficient solutions for key management in distributed systems are needed. Large deployments in the Cloud can require millions of keys for thousands of clients. The current approaches for serving keys are centralized components, which do not scale as desired.

The work within ESCUDO-CLOUD has realized a key manager that uses an untrusted distributed key-value store (KVS) and offers consistent key distribution over the Key-Management Interoperability Protocol (KMIP). To achieve confidentiality, it uses a key hierarchy where every key except a root key itself is encrypted by the respective parent key. The hierarchy also allows for key rotation and, ultimately, for secure deletion of data. The design permits key rotation to proceed concurrently with key-serving operations. This is further described in Deliverable D2.3 (M24).

A prototype was integrated with IBM Spectrum Scale, a highly scalable cluster file system, where it serves keys for file encryption. Linear scalability was achieved even under load from concurrent key updates. The implementation shows that the approach is viable, works as intended, and suitable for high-throughput key serving in Cloud platforms.

2.5 Provable security of updatable encryption

In this section, we describe recent work on stronger security guarantees for updatable encryption. In more detail, we describe formal security definitions for ciphertext-independent schemes, and analyze several schemes with respect to these definitions. This work takes a longer-term perspective and develops cryptography with a view toward future deployments. The mechanisms have not been implemented yet.

The trivial approach to update an existing ciphertext towards a new key is to decrypt the ciphertext and re-encrypt the underlying plaintext from scratch using the fresh key. Implementing this approach for secure Cloud storage applications where the data owner outsources his data in encrypted form to a potentially untrusted host is not trivial, though: Either the owner has to download, re-encrypt and upload all ciphertexts, which makes outsourcing impractical, or the encryption keys have to be sent to the host, violating security.

2.5.1 Updatable encryption

A better solution for updating ciphertexts has been proposed by Boneh et al. [BLMR13]: in what they call an *updatable encryption scheme*, the data owner can produce a short update tweak that allows the host to re-encrypt the data himself, while preserving the security of the encryption, i.e., the tweak allows to migrate ciphertexts from an old to a new key, but does not give the host an advantage in breaking the confidentiality of the protected data. Boneh et al. also proposed a construction (BLMR) based on key-homomorphic PRFs, which is essentially a symmetric proxy re-encryption scheme (PRE) where one sequentially re-encrypts data from one epoch to the next.

While being somewhat similar in spirit, PRE and updatable encryption do have different security requirements: PRE schemes often keep parts of the ciphertexts static throughout re-encryption, as there is no need to make a re-encrypted ciphertexts independent from the original ciphertext it was derived from. In updatable encryption, however, the goal should be that an updated ciphertext is as secure as a fresh encryption; in particular, it should look like an independently computed ciphertext even given previous ones. Thus, any scheme that produces linkable ciphertexts, such as the original BLMR construction, cannot guarantee such a security notion capturing post-compromise security of updated ciphertexts.

Ciphertext-independence vs. ciphertext-dependence. In the full version of their paper, Boneh et al. [BLMR15] provide security notions for updatable encryption, which aim to cover the desired indistinguishability of updated ciphertexts. To satisfy that notion they have to remove the linkability from the BLMR scheme which they achieve by moving to the setting of *ciphertext-dependent* updates. In ciphertext-dependent schemes, the owner no longer produces a single tweak that can update *all* ciphertexts, but produces a dedicated tweak for each ciphertext. Therefore, the owner has to download all outsourced ciphertexts, compute a specific tweak for every ciphertext, and send all tweaks back to the host.

Clearly, ciphertext-dependent schemes are much less efficient and more cumbersome for the data owner than *ciphertext-independent* ones. They also increase the complexity of the update procedure for the host, who has to ensure that it applies the correct tweak for each ciphertext—any mistake renders the updated ciphertexts useless. Another, more subtle disadvantage of ciphertext-independent schemes is that they require the old and new keys to be present together for a longer time, as the owner needs both keys to derive the individual tweaks for all of his ciphertexts. Deleting the old key too early might risk loosing the ability of decrypting ciphertexts that have not been upgradet yet, whereas keeping the old key too long makes an attack at that time more lucrative—the adversary obtains two keys at the same time. In a ciphertext-independent scheme, the old key can and should be deleted immediately after the tweak has been derived.

In a recent work [EPRS17], Everspaugh et al. provide a systematic treatment for such ciphertext-dependent schemes and observe that computing the tweak often does not require access to the full ciphertext, but only to a short ciphertext header, which allows to moderately improve the efficiency of this approach. Everspaugh et al. also show that the security notions from [BLMR15] do not cover the desired property of post-compromise security of updated ciphertexts. They provide two new security notions and propose schemes that can provably satisfy them. As a side-result, they also propose a security definition for ciphertext-independent schemes and suggest a simple xor-based scheme (XOR-KEM) for this setting.

Ambiguity of security models. Interestingly, both previous works phrase the algorithms and security models for updatable encryption in the flavor of normal proxy re-encryption. That leads to a mismatch of how the scheme is used and modeled—in practice, an updatable encryption scheme is used in a clear sequential setting, updating ciphertexts as the key progresses. The security model offers more and unrealistic flexibility, though: it allows to rotate keys and ciphertexts across *arbitrary* epochs, jumping back in forth in time. This flexibility gives the adversary more power than he has in reality and, most importantly, makes the security that is captured by the model hard to grasp, as it is not clear *when* the adversary is allowed to corrupt keys.

Non-intuitive security definitions increase the risk that proofs are flawed or that schemes are unintentionally used outside the security model. The way that Everspaugh et al. [EPRS17] define security for (ciphertext-independent) schemes is ambiguous, and only the weaker interpretation of their model allows their scheme XOR-KEM to be proven secure. However, this weaker interpretation does not guarantee any confidentiality *after* a secret key got compromised, as it allows key corruption only after the challenge epoch. Thus, an updatable scheme that is secure only in such a weak model does not provide the intuitive security one would expect from key rotation: namely that after migrating to the new key, the old one becomes useless and no longer of value to the adversary. To the contrary, all previous keys still require strong protection or secure deletion.

Importance of post-compromise security. Realizing secure deletion in practice is virtually impossible, as keys may be copied or moved across the RAM, swap partitions, and SSD memory blocks, and thus we consider *post-compromise security* to be an essential property of updatable schemes. Avoiding the assumption of securely deleted keys has recently inspired numerous works on how to achieve post-compromise security in other encryption settings [BSJ+17, CGCD+17, GM17, CGCG16]. Note that an updatable encryption scheme that is not post-compromise secure can even reduce the security compared with a scheme where keys are never rotated: as one expects old keys to be useless after rotation, practitioners can be mislead to reduce the safety measures for "expired" keys, which in turn makes key compromises more likely. For the example of Everspaugh et al.'s simple XOR-KEM scheme [EPRS17], a single compromised old key allows to fully recover the fresh key.

This leaves open the important question how to design a ciphertext-independent scheme that achieves post-compromise security, capturing the full spirit of updatable encryption and key rotation.

Contributions

In recent work [LT17], we provide a comprehensive treatment for *ciphertext-independent* updatable encryption schemes that have clear advantages in efficiency and ease-of-deployment over the ciphertext-dependent solutions. We model updatable encryption and its security in the natural sequential manner that is inherent in key rotation, avoiding the ambiguity of previous works, and clearly capturing all desired security properties. We also analyze the (in)security of a number of existing schemes and finally propose a construction that provably satisfies our strong security notions.

Strong security models. We define updatable encryption in its natural form where keys and ciphertexts sequentially evolve over time epochs. To capture security, we allow the adversary to *adaptively* corrupt secret keys, tweaks and ciphertexts in any combination of epochs as long as this does not allow him to trivially decrypt a challenge ciphertext. In our first notion, *indistiguishability of encryptions* (IND-ENC), such a challenge ciphertext will be a fresh encryption C_d of one of two messages m_0, m_1 under the current epoch key and the task of the adversary is to guess the bit d. This is the standard CPA game adapted to the updatable and adaptive corruption setting. Our second notion, *indistiguishability of updates* (IND-UPD), returns as a challenge the re-encryption C'_d of a ciphertext either C_0 or C_1, and an adversary has again to guess the bit d.

We stress that this second property is essential for the security of updatable encryption schemes, as it captures confidentiality of *updated* encryptions, whereas IND-ENC only guarantees security

	SE-KEM	2ENC	BLMR	BLMR+	RR-EIG
IND-ENC	(✓) no tweaks near a challenge	(✓) either tweak near challenge, or secret key	✓	✓	✓
IND-UPD	✘	(✓) no tweaks near a challenge	✘	(✓) at most one tweak	✓

Table 2.1: Overview of results in [LT17]. (Corruption of secret keys in challenge epochs is forbidden by the IND-ENC and IND-UPD definitions. The symbol (✓) denotes that a scheme requires additional constraints on the tweaks that can be corrupted to achieve the security notion.)

for ciphertexts that originate from a fresh encryption. While IND-ENC is similar to the security of symmetric proxy re-encryption schemes, IND-UPD is a property that is special to the context of key rotation. And thus, contrary to the common believe, a symmetric PRE scheme cannot directly be used for secure updatable encryption [BLMR13, EPRS17, MS17]!

In the ciphertext-independent setting, capturing the information that the adversary can infer from a certain amount of corrupted tweaks, keys and ciphertexts is a delicate matter, as e.g., an update tweak allows the adversary to move *any* ciphertext from one epoch to the next. We observe that all existing constructions leak more information than necessary. Instead of hard-coding the behavior of the known schemes into the security model, we propose a set of leakage profiles, and define both the optimal and currently achievable leakage.

We then compare our model to the existing definition for encryption indistinguishability by Everspaugh et al. [EPRS17]. We argue that their definition can be interpreted in two ways: the weaker interpretation rules out post-compromise security, but allows the XOR-KEM construction to be secure, whereas the stronger interpretation is closer to our IND-ENC model. However, in their stronger version, as well as in our IND-ENC notion, we show that XOR-KEM cannot be secure by describing a simple attack that allows to recover the challenge secret key after compromising one old key. We further show that IND-ENC is strictly stronger than the weak interpretation of [EPRS17], but incomparable to the stronger one, due to the way both models handle adversarial ciphertexts.

Provably secure constructions. We further analyze several schemes according to the new definitions, the results are summarized in Table 2.1. First, we consider a simple construction (called 2ENC) that is purely based on symmetric primitives. Unfortunately, the scheme cannot satisfy our strong security notions. Yet, instead of simply labeling this real-world solution as insecure, we formulate the additional constraints on the adversarial behavior that suffice to prove its security in relaxed versions of our IND-ENC and IND-UPD models.

We then turn our attention to less efficient but more secure schemes, starting with the BLMR construction by Boneh et al. [BLMR13] that uses key-homomorphic PRFs. We show that the original BLMR scheme does satisfy our IND-ENC notion but not IND-UPD, and also propose a slight modification BLMR+ that improves the latter and achieves a weak form of update indistinguishability. While BLMR seems to be a purely symmetric solution on the first glance, any instantiation of the underlying key-homomorphic PRFs so far requires modular exponentiations or is built from lattices. The same holds for the recent ciphertext-dependent construction by Everspaugh et al. [EPRS17] that also relies on key-homomorphic PRFs and suggests a discrete-logarithm based

	Ciphertext independent	Encryption	Tweak Derivation	Update of n Ciphertexts
BLMR' [BLMR15]		2 exp.	$2n$ sym.	$2n$ exp.
ReCrypt [EPRS17]		2 exp.	$2n$ exp.	$2n$ exp.
BLMR [BLMR13]	✓	2 exp.	2 exp.	$2n$ exp.
BLMR+ (this work)	✓	2 exp.	2 exp.	$2n$ exp.
RR-EIG (this work)	✓	2 exp.	1 exp.	$2n$ exp.

Table 2.2: Comparison of computational efficiency measured by the most expensive operations for short (one-block) ciphertexts (exponentiation, symmetric cryptography).[1] Note that the ciphertext-dependent BLMR' variant of [BLMR15] is unlikely to have a security proof [EPRS17], and BLMR and BLMR+ achieve significantly weaker security than RR-EIG. (SE-KEM and 2ENC are omitted here as they are purely symmetric solutions.)

instantiation.

Acknowledging that secure updatable encryption schemes seem to inherently require techniques from the public-key world, we then build a scheme that omits the intermediate abstraction of using key-homomorphic PRFs which allows us to take full advantage of the underlying group operations. Our construction (RR-EIG) can be seen as the classic ElGamal-based proxy re-encryption scheme combined with a fresh re-randomization upon each re-encryption. We prove that this scheme fully achieves both of our strong security definitions.

We compare the schemes in terms of efficiency in Table 2.2. The costs for encryption and updates of our most secure RR-EIG scheme are—on the owner side—even lower than the costs in the less secure BLMR scheme and the recent ciphertext-dependent scheme ReCrypt by Everspaugh et al. [EPRS17]. The solution by Everspaugh et al. shifts significantly many expensive update operations to the data owner, who has to compute two exponentiation for each ciphertext (block) that shall be updated, whereas our scheme requires the owner to compute only a single exponentiation for the update of all ciphertexts.

We additionally analyze a "hybrid-encryption" scheme SE-KEM that is widely used in practical data-at-rest protection such as the one described in the previous sub-sections, where the encrypted plaintext is stored together with the encryption key wrapped under an epoch key. The scheme provides rather weak guarantees when viewed as an updatable encryption scheme, but is still useful in certain scenarios due to the efficient key update.

Other related work

Beyond the previous work on updatable encryption [BLMR13, BLMR15, EPRS17] that we already discussed above, the most closely related line of work is on (symmetric) proxy re-encryption (PRE) [BBS98, AFGH06, HRsV07, LV08a, LV08b, ABH09, CWYD10]. However, as stressed before, while being similar in the sense that PRE allows a proxy to move ciphertexts from one key to another, the desired security guarantees have subtle differences and the security property of IND-UPD that is crucial for updatable encryption is neither covered nor needed by PRE.

While this means that a secure PRE does not automatically yield a secure updatable encryption scheme, it does not prevent PREs from being secure in the updatable encryption sense as well—but this has to be proven from scratch. In fact, our schemes are strongly inspired by proxy re-encryption: For the simple double-encryption scheme discussed by Ivan and Dodis [ID03], we show that a weak form of security can be proven, and our most secure scheme RR-ElG combines the ElGamal-based PRE with re-randomization of ciphertexts. We also observe similar challenges in designing schemes that limit the "power" of the tweak, which is related to the long-standing problem of constructing efficient PRE's that are uni-directional, multi-hop and collusion-resistant.

In the context of tokenization, which is the process of consistently replacing sensitive elements, such as credit card numbers, with non-sensitive surrogate values, the feature of key rotation has recently been studied by Cachin et al. [CCFSL17]. Their schemes are inherently deterministic, and thus their results are not applicable to the problem of probabilistic encryption, but we follow their formalization of modeling key rotation in a strictly sequential manner.

Finally, a recent paper of Ananth et al. [ACJ17] provides a broader perspective on updatable cryptography, but targets generic and rather complex schemes with techniques such as randomized encodings. The definitions in their work have linkability hardcoded, as randomness has to remain the same across updates, which is in contrast to our goal of achieving efficient unlinkable schemes for the specific case of updatable encryption.

2.5.2 Formal model and security definitions

An updatable encryption scheme contains algorithms for a data *owner* and a *host*. The owner encrypts data using the UE.enc algorithm, and then outsources the ciphertexts to the host. To this end, the data owner initially runs an algorithm UE.setup to create an encryption key. The encryption key evolves with *epochs*, and the data is encrypted with respect to a specific epoch e, starting with $e = 0$. When moving from epoch e to epoch $e+1$, the owner invokes an algorithm UE.next to generate the key material k_{e+1} for the new epoch and an update tweak Δ_{e+1}. The owner then sends Δ_{e+1} to the host, deletes k_e and Δ_{e+1} immediately, and uses k_{e+1} for encryption from now on. After receiving Δ_{e+1}, the host first deletes Δ_e and then uses an algorithm UE.upd to update all previously received ciphertexts from epoch e to $e+1$, using Δ_{e+1}. Hence, during some epoch e, the update tweak from $e-1$ to e is available at the host, but update tweaks from earlier epochs have been deleted. (The host could already delete the tweak when all ciphertexts are updated, but as this is hard to model in the security game, we assume the tweak to be available throughout the full epoch.)

Security properties

The main goal of updatable encryption is twofold: First, it should enable efficient updates by a potentially corrupt host, i.e., the update procedure and compromise of the update tweaks must not reduce the standard security of the encryption. Second, the core purpose of key rotation is to reduce the risk and impact of key exposures, i.e., confidentiality should be preserved or even re-gained in the presence of *temporary* key compromises, which can be split into forward and post-compromise security. Furthermore, we aim for security against adaptive and retroactive corruptions, modeling that any key or tweak from a current or previous epoch can become compromised.

Tweak security: The feature of updating ciphertexts should not harm the standard IND-CPA security of the encryption scheme. That is, seeing updated ciphertexts or even the exposure of *all* tweaks does not increase an adversary's advantage in breaking the encryption scheme.

[1] The computation of the key-homomorphic PRF requires 2 exponentiations—one for hashing into the group and one for computing the PRF.

Forward security: An adversary compromising a secret key in some epoch e^* does not gain any advantage in decrypting ciphertexts he obtained in epochs $e < e^*$ *before* that compromise.

Post-compromise security: An adversary compromising a secret key in some epoch e^* does not gain any advantage in decrypting ciphertexts he obtained in epochs $e > e^*$ *after* that compromise.

Adaptive security: An adversary can adaptively corrupt keys and tweaks of the current epoch and all previous ones.

Given that updatable encryption schemes can produce ciphertexts in two ways—either via a direct encryption or an update of a previous ciphertext—we require that the above properties must hold for both settings. This inspires our split into two indistinguishability-based security notions, one capturing security of direct encryptions (IND-ENC) and one ruling out attacks against updated ciphertexts (IND-UPD). Both security notions are defined through experiments run between a challenger and an adversary $\mathscr{A}$. Depending on the notion, the adversary may issue queries to different oracles, defined in the next section. At a high level, $\mathscr{A}$ is allowed to adaptively corrupt arbitrary choices of secret keys and update tweaks, as long as they do not allow him to trivially decrypt the challenge ciphertext.

The importance of post-compromise security. We have formalized updatable encryption in the strict sequential setting it will be used in, and in particular modeled key derivation of a new key k_{e+1} as a sequential update $(k_{e+1}, \Delta_{e+1}) \leftarrow \mathsf{UE.next}(k_e)$ of the old key k_e. Previous works [BLMR13, EPRS17] instead model key rotation by generating fresh keys via a dedicated $k_{e+1} \leftarrow \mathsf{UE.kgen}(\lambda)$ algorithm at each epoch and deriving the tweak as $\Delta_{e+1} \leftarrow \mathsf{UE.next}(k_e, k_{e+1})$.

One impact of our sequential model is that post-compromise security becomes much more essential, as this property intuitively ensures that new keys are independent of the old ones (which is directly ensured in the previous formalization). Wthout requiring post-compromise security, $\mathsf{UE.next}(k_e)$ could generate the new key by hashing the old one: $k_{e+1} \leftarrow \mathsf{H}(k_e)$. If H is modeled as a random oracle, this has no impact for standard or forward security, but any scheme with such a key update looses all security in the post-compromise setting. An adversary compromising a single secret key k_e can derive all future keys himself.

What we do not model. The focus of this work is to obtain security against arbitrary key compromises, i.e., an adversary can steal secret keys, update tweaks, and outsourced ciphertexts at any epoch. We do not consider attacks where an adversary fully takes over the owner or host and starts manipulating ciphertexts, e.g., providing adversarially generated ciphertexts to the host, or tampering with the update procedure. Thus, we model passive CPA attacks but not active CCA ones, and assume that all ciphertexts and updates are honestly generated. We believe this still captures the main threat in the context of updatable encryption, namely smash-and-grab attacks aiming at compromising the key material.

In fact, this restriction to passive attacks allows us to be more generous when it comes to legitimate queries towards corrupted epochs, as we can distinguish challenge from non-challenge ciphertexts and only prohibit the ones that allow trivial wins. Interestingly, Everspaugh et al. use a similar appoach in their stronger CCA-like security notion for ciphertext-independent schemes where they are able to recognize whether a ciphertext is derived from the challenge and prevent these from being updated towards a corrupt key. They are able to recognize challenge ciphertexts as all keys are generated honestly, i.e., they are known to the challenger, and updates are required to be deterministic. The latter allows the challenger to trivially keep track of the challenge ciphertext, but it also makes misuse of the schemes more likely: if a scheme is implemented with probabilistic updates, which intuitively seems to only increase security, then one steps outside of the model and loses all security guarantees. In our model, we allow updates to be probabilitiy, and in fact, the security of our strongest construction crucially relies on the re-randomization of updated ciphertexts.

2.5.3 Constructions

We analyze several constructions of updatable encryption with respect to our security notions of indistinguishability of encryptions and updates.

Double Encryption (2ENC)

An approach that is based only on symmetric encryption is to first encrypt the plaintext under an "inner key," and subsequently encrypt the resulting ciphertext under a second, "outer key." In each epoch, the outer key is changed, and the ciphertext is updated by decrypting the outer encryption and re-encrypting under the new key. This scheme has been proposed by Ivan and Dodis [ID03] as symmetric uni-directional proxy re-encryption.[2] It has also appeared in other contexts, such as so-called "over-encryption" for access revocation in Cloud storage systems [BdVF+16]. More formally, this scheme can be phrased as an updatable encryption scheme $\mathsf{UE}_{\mathsf{2ENC}} = \mathsf{UE}$ as follows.

$\mathsf{UE.setup}(\lambda)$**:** ${k_0}^o \leftarrow \mathsf{SE.kgen}(\lambda)$, $k^i \leftarrow \mathsf{SE.kgen}(\lambda)$, return $k_0 \leftarrow ({k_0}^o, k^i)$

$\mathsf{UE.next}(\boldsymbol{k}_e)$**:** parse $k_e = ({k_e}^o, k^i)$, get ${k_{e+1}}^o \leftarrow \mathsf{SE.kgen}(\lambda)$, $\Delta_{e+1} \leftarrow ({k_e}^o, {k_{e+1}}^o)$, $k_{e+1} \leftarrow ({k_{e+1}}^o, k^i)$, return (k_{e+1}, Δ_{e+1})

$\mathsf{UE.enc}(\boldsymbol{k}_e, m)$**:** parse $k_e = ({k_e}^o, k^i) \leftarrow k_e$, return $C_e \leftarrow \mathsf{SE.enc}({k_e}^o, \mathsf{SE.enc}(k^i, m))$

$\mathsf{UE.upd}(\Delta_{e+1}, C_e)$**:** parse $\Delta_{e+1} = ({k_e}^o, {k_{e+1}}^o)$, return $C_{e+1} \leftarrow \mathsf{SE.enc}({k_{e+1}}^o, \mathsf{SE.dec}({k_e}^o, C_e))$

$\mathsf{UE.dec}(\boldsymbol{k}_e, C_e)$**:** parse $k_e = ({k_e}^o, k^i) \leftarrow k_e$, return $\mathsf{SE.dec}({k_e}^o, \mathsf{SE.dec}(k^i, C))$

Clearly this scheme does not achieve our desired IND-ENC security. A ciphertext can be decrypted if an adversary sees the secret key of *some* epoch and one of the tweaks relating to the epoch where he learned the ciphertext. However, we show that this is the only additional attack, i.e., if the adversary never sees such a combination of tweaks and keys, then the scheme is secure, which is formalized by the following theorem.

Theorem 2.5.1 ($\mathsf{UE}_{\mathsf{2ENC}}$ **is weakly** IND-ENC **secure)** *Let* SE *be a CPA-secure encryption scheme, then* $\mathsf{UE}_{\mathsf{2ENC}}$ *is* IND-ENC*-secure if the following additional condition holds: If there exist a challenge-equal epoch e where $\mathcal{A}$ obtains a tweak in epochs e or $e+1$, then $\mathcal{A}$ must not make* any *query to obtain the owner secret.*

The proof of this theorem turns out to be surprisingly subtle. As intuitively expected, it consists of two reductions to the security of SE, but the reduction for the outer encryption part is

[2] It is uni-directional in a proxy re-encryption scheme; the proxy removes the outer layer. As an updatable scheme, which replaces the outer layer, it is bi-directional.

complicated by the fact that $\mathscr{A}$ may obtain either the tweak or the challenge ciphertext adaptively and in multiple epochs. Instead of guessing all epochs, which would lead to a large loss in tightness, we devise a specific hybrid argument and formalize the intuition that only epochs with a challenge-ciphertext query can help $\mathscr{A}$ in gaining advantage.

It is also easy to see that the double encryption scheme is not $\mathsf{IND\text{-}UPD}$ secure: The inner ciphertext remains static and an adversary seeing tweaks that allow him to unwrap the outer encryption can trivially link ciphertexts across epochs. But we again show that this is the only attack, i.e., $\mathsf{2ENC}$ achieves a weak form of $\mathsf{IND\text{-}UPD}$ security if the adversary is restricted to learn at most one update tweak Δ_e for an epoch e for which he also obtained the challenge ciphertext in epochs e or $e-1$.

Theorem 2.5.2 ($\mathsf{UE}_{\mathsf{2ENC}}$ is weakly $\mathsf{IND\text{-}UPD}$ secure) *Let SE be an $\mathsf{IND\text{-}CPA}$-secure encryption scheme, then $\mathsf{UE}_{\mathsf{2ENC}}$ is $\mathsf{IND\text{-}UPD}$-secure if the following additional condition holds: Adversary $\mathscr{A}$ makes at most one update-tweak query for all e where e or $e-1$ that are challenge-equal.*

Schemes from Key-Homomorphic PRFs (BLMR and $\mathsf{BLMR+}$)

Boneh et al. [BLMR13] proposed an updatable encryption scheme based on key-homomorphic pseudorandom functions, to which we will refer as BLMR-scheme. We first recall the notion of key-homomorphic PRFs and then present the BLMR and our improved $\mathsf{BLMR+}$ scheme.

Definition 2.5.1 (Key-homomorphic PRF [BLMR15]) *Consider an efficiently computable function $\mathsf{F} : \mathscr{K} \times \mathscr{X} \to \mathscr{Y}$ such that $(\mathscr{K}, \oplus)$ and $(\mathscr{Y}, \otimes)$ are both groups. We say that F is a key-homomorphic PRF if the following properties hold:*

1. *F is a secure pseudorandom function.*
2. *For every $k_1, k_2 \in \mathscr{K}$, and every $x \in \mathscr{X}$: $\mathsf{F}(k_1, x) \otimes \mathsf{F}(k_2, x) = \mathsf{F}((k_1 \oplus k_2), x)$*

A simple example of a secure key-homomorphic PRF is the function $\mathsf{F}(k,x) = h(x)^k$ where $\mathscr{Y} = \mathbb{G}$ is an additive group in which the DDH assumption holds, and h is a random oracle [NPR99].

Based on such a key-homomorphic PRF F, the BLMR construction is described as the following scheme $\mathsf{UE}_{\mathsf{BLMR}} = \mathsf{UE}$.

$\mathsf{UE.setup}(\lambda)$**:** compute $k_0 \leftarrow \mathsf{F.kgen}(\lambda)$, return k_0

$\mathsf{UE.next}(\boldsymbol{k}_e)$**:** $k_{e+1} \leftarrow \mathsf{F.kgen}(\lambda)$ return $(k_{e+1}, (k_e \oplus k_{e+1}))$

$\mathsf{UE.enc}(\boldsymbol{k}_e, m)$**:** $N \leftarrow \mathscr{X}$, return $((\mathsf{F}(k_e, N) \otimes m), N)$

$\mathsf{UE.dec}(\boldsymbol{k}_e, C_e)$**:** parse $C_e = (C_1, N)$, return $m \leftarrow C_1 \otimes \mathsf{F}(k_e, N)$.

$\mathsf{UE.upd}(\Delta_{e+1}, C_e)$**:** parse $C_e = (C_1, N)$, return $((C_1 \otimes \mathsf{F}(\Delta_{e+1}, N)), N)$

Indeed, the subsequent theorem shows that BLMR is $\mathsf{IND\text{-}ENC}$-secure.

Theorem 2.5.3 ($\mathsf{UE}_{\mathsf{BLMR}}$ is $\mathsf{IND\text{-}ENC}$-secure) *Let F be a key-homomorphic PRF where $\mathsf{F.kgen}(\lambda)$ returns uniformly random elements from $\mathscr{K}$, then $\mathsf{UE}_{\mathsf{BLMR}}$ is $\mathsf{IND\text{-}ENC}$-secure.*

The proof is a combination of the original proof in [BLMR13] and the techniques already used in the proofs of the 2ENC scheme. The BLMR scheme does not achieve the notion IND-UPD of update-indistinguishability though, as the second part of the ciphertext remains static throughout the updates. This might have inspired the change to the ciphertext-dependent setting in the full version of Boneh et al.'s paper [BLMR15]. Ciphertext-dependent updates, however, have the disadvantage that the key owner must produce one update tweak for each ciphertext to be updated. We show that a mild form of IND-UPD security can be achieved in the ciphertext-independent setting via a simple modification to the BLMR scheme.

The BLMR+ scheme. The BLMR+ scheme follows the basic structure of BLMR, but additionally encrypts the nonce. In more detail, in every epoch the owner also generates a second key $k'_e \leftarrow \mathsf{SE.kgen}(\lambda)$ of a symmetric encryption scheme and encrypts the nonce-part N of each ciphertext under that key. In BLMR+, we simply include the old and new symmetric key into the update tweak and let the host re-encrypt the full ciphertext.

The choice to simply reveal both keys might seem odd, but it is not revealing much more information to a corrupt host than the ciphertext-dependent scheme. The only information it reveals to a corrupt host is which old ciphertext belongs to which new one, which he learns anyway as he generates the new ciphertexts. What we gain with BLMR+ is that an adversary seeing only (updated) ciphertexts of different epochs cannot tell anymore which of them belong together. Clearly, this unlinkability is limited, though, as it is only guaranteed if the adversary learns at most one update tweak that is related to an epoch where he also learned the challenge ciphertext.

In more detail, this modification is the following scheme $\mathsf{UE}_{\mathsf{BLMR+}} = \mathsf{UE}$:

$\mathsf{UE.setup}(\lambda)$**:** $k_0^1 \leftarrow \mathsf{F.kgen}(\lambda)$, $k_0^2 \leftarrow \mathsf{SE.kgen}(\lambda)$, return $k_0 \leftarrow (k_0^1, k_0^2)$

$\mathsf{UE.next}(\boldsymbol{k}_e)$**:** parse $k_e = (k_e^1, k_e^2)$, get $k_{e+1}^1 \leftarrow \mathsf{F.kgen}(\lambda)$, $k_{e+1}^2 \leftarrow \mathsf{SE.kgen}(\lambda)$, $k_{e+1} \leftarrow (k_{e+1}^1, k_{e+1}^2)$, $\Delta_{e+1} \leftarrow (k_e^1 \oplus k_{e+1}^1, (k_e^2, k_{e+1}^2))$, return (k_{e+1}, Δ_{e+1})

$\mathsf{UE.enc}(\boldsymbol{k}_e, m)$**:** parse $k_e = (k_e^1, k_e^2)$, get $N \leftarrow \mathscr{X}$, $C^1 \leftarrow \mathsf{F}(k_e^1, N) \otimes m$, $C^2 \leftarrow \mathsf{SE.}\mathtt{enc}(k_e^2, N)$, return $C_e \leftarrow (C^1, C^2)$

$\mathsf{UE.dec}(\boldsymbol{k}_e, C_e)$**:** parse $k_e = (k_e^1, k_e^2)$ and $C_e = (C^1, C^2)$, get $N \leftarrow \mathsf{SE.D}((,k)_e^2, C^2)$, return $m \leftarrow C^1 \otimes \mathsf{F}(k_e^1, N)$

$\mathsf{UE.upd}(\Delta_{e+1}, C_e)$**:** parse $\Delta_{e+1} = (\Delta_{e+1}^1, (k_e^2, k_{e+1}^2))$ and $C_e = (C_e^1, C_e^2)$, get $N \leftarrow \mathsf{SE.D}((,k)_e^2, C^2)$, $C_{e+1}^1 \leftarrow C_e^1 \otimes \mathsf{F}(\Delta_{e+1}^1, N)$, $C_{e+1}^2 \leftarrow \mathsf{SE.}\mathtt{enc}(k_{e+1}^2, N)$, return $C_{e+1} \leftarrow (C_{e+1}^1, C_{e+1}^2)$.

We first state the following corollary as an easy extension of Theorem 2.5.3 on BLMR. The encryption of the nonce can be easily simulated in the reduction.

Theorem 2.5.4 *The* $\mathsf{UE}_{\mathsf{BLMR+}}$ *scheme is* IND-ENC *secure.*

We then prove that the modified BLMR+ scheme described above indeed achieves a weak form of IND-UPD security. The intuition behind the level of security specified in the following theorem is that knowing either a tweak or the key of the ciphertexts later used in the challenge in a round prior to the challenge allows the adversary to decrypt the nonce. Also, obtaining the challenge ciphertext and a tweak after the challenge query allows the adversary to decrypt the nonce. To obtain unlinkability, we cannot allow the adversary to access the nonce both before and after the challenge query, but the theorem formalizes that we have security unless the adversary gains this access in *both* cases.

Theorem 2.5.5 ($\mathsf{UE}_{\mathsf{BLMR+}}$ **is weakly** $\mathsf{IND\text{-}UPD}$ **secure.)** *Let* F *be a key-homomorphic PRF, and assume that all elements of* $\mathscr{X}$ *are encoded as strings of the same length. Let* SE *be a* $\mathsf{IND\text{-}CPA}$*-secure symmetric encryption scheme. Then, the scheme* $\mathsf{UE}_{\mathsf{BLMR+}}$ *is (weakly)* $\mathsf{IND\text{-}UPD}$*-secure against adversaries for which at most one of the two conditions may hold:*

- *Let e denote the epoch in which the first ciphertext that is later used as challenge C_0 or C_1 was encrypted. For some $e^* \in \{e, \dots, \tilde{e}-1\}$, $\mathscr{A}$ knows the secret key k_{e^*}*
- *For some epoch $e \geq \tilde{e}$, the challenge ciphertext $\tilde{C}_e$ and (at least) one of the tweaks Δ_e, Δ_{e+1} are known to $\mathscr{A}$.*

The proof of this theorem is essentially a combination of the techniques used in the proofs of Theorems 2.5.2 and 2.5.3. It is provided in the supplementary material.

Updatable Encryption based on ElGamal (RR-EIG)

We finally present a scheme that achieves both strong notions of indistinguishability of encryptions ($\mathsf{IND\text{-}ENC}$) and updates ($\mathsf{IND\text{-}UPD}$). This scheme uses the classical proxy re-encryption idea based on ElGamal that was originally proposed by Blaze et al. [BBS98], but uses it in the secret-key setting. This alone would not be secure though, as parts of the ciphertext would remain static. What we additionally exploit is that ElGamal ciphertexts can be re-randomized by knowing only the public key. Thus, we add the "public-key" element of the epoch to the tweak and perform a re-randomization whenever a ciphertext gets updated. This makes it the first of the considered schemes where the update algorithm is probabilistic. Interestingly, this is not allowed in the ciphertext-dependent work by Everspaugh et al. [EPRS17] which require updates to be deterministic such that the challenger in the security game can keep track of the challenge ciphertexts.

The use of public-key techniques for secret-key updatable encryption may appear unnecessary. We emphasize, however, that previous constructions are based on key-homomorphic PRFs, all instantiations of which are based on such techniques as well. By contrast, the direct use of the group structure without the intermediate abstraction allows us to implement the re-randomization and thereby achieve full $\mathsf{IND\text{-}UPD}$ security.

In fact, in terms of exponentiations, an encryption in our $\mathsf{RR\text{-}EIG}$ scheme is as efficient as in BLMR and in Everspaugh et al.'s ReCrypt scheme [EPRS17], whereas the computations of update tweaks and ciphertext updates are even more efficient than in [EPRS17] due to the ciphertext-independent setting of our work.

Let $(\mathbb{G}, g, q)$ be system parameters available as CRS such that the DDH problem is hard w.r.t. λ, i.e., q is a λ-bit prime. The scheme is described as $\mathsf{UE}_{\mathsf{RR\text{-}EIG}} = \mathsf{UE}$ as follows.

$\mathsf{UE.setup}(\lambda)$**:** $x \leftarrow \mathbb{Z}_q^*$, set $k_0 \leftarrow (x, g^x)$, return k_0

$\mathsf{UE.next}(\boldsymbol{k}_e)$**:** parse $k_e = (x,y)$, $x' \leftarrow \mathbb{Z}_q^*$, $k_{e+1} \leftarrow (x', g^{x'})$, $\Delta_{e+1} \leftarrow (x'/x, g^{x'})$ return (k_{e+1}, Δ_{e+1})

$\mathsf{UE.enc}(\boldsymbol{k}_e, m)$**:** parse $k_e = (x,y)$, $r \leftarrow \mathbb{Z}_q$, return $C_e \leftarrow (y^r, g^r m)$

$\mathsf{UE.dec}(\boldsymbol{k}_e, C_e)$**:** parse $k_e = (x,y)$ and $C_e = (C_1, C_2)$, return $m' \leftarrow C_2 \cdot C_1^{-1/x}$

$\mathsf{UE.upd}(\Delta_{e+1}, C_e)$**:** parse $\Delta_{e+1} = (\Delta, y')$ and $C_e = (C_1, C_2)$, $r' \leftarrow \mathbb{Z}_q$, $C_1' \leftarrow C_1^{\Delta} \cdot y'^{r'}$, $C_2' \leftarrow C_2 \cdot g^{r'}$, return $C_{e+1} \leftarrow (C_1', C_2')$

The keys x for the encryption scheme are chosen from $\mathbb{Z}_q^*$ instead of $\mathbb{Z}_q$ as usual. The reason is that the update is multiplicative, and this restriction makes sure that each key is uniformly random in $\mathbb{Z}_q^*$. As this changes the distribution only negligibly, the standard Diffie-Hellman argument still applies. However, the adaptation simplifies the security proof.

Theorem 2.5.6 ($\mathsf{UE}_{\mathsf{RR\text{-}EIG}}$ **is** $\mathsf{IND\text{-}ENC}$ **secure)** *The updatable encryption scheme* $\mathsf{UE} = \mathsf{UE}_{\mathsf{RR\text{-}EIG}}$ *is* $\mathsf{IND\text{-}ENC}$ *secure under the DDH assumption.*

We also show that the scheme $\mathsf{RR\text{-}EIG}$ is unlinkable. This property is achieved by the re-randomization of the updates.

Theorem 2.5.7 ($\mathsf{UE}_{\mathsf{RR\text{-}EIG}}$ **is** $\mathsf{IND\text{-}UPD}$ **secure)** *The updatable encryption scheme* $\mathsf{UE} = \mathsf{UE}_{\mathsf{RR\text{-}EIG}}$ *is* $\mathsf{IND\text{-}UPD}$ *secure under the DDH assumption.*

One might wonder whether one could more generally build a secure updatable encryption scheme from any secure symmetric proxy re-encryption that additionally allows public re-randomization of ciphertexts. For that analysis one would need a security notion for symmetric proxy re-encryption schemes that allows *adaptive* corruptions as in our models. However, so far, adaptive corruptions have only been considered for schemes that are uni-directional and single-hop, i.e., where the re-encryption capabilities would not be sufficient for updatable encryption.

Symmetric Key-Encapsulation ($\mathsf{SE\text{-}KEM}$)

We additionally analyze a scheme that can be considered as a symmetric key-encapsulation mechanism (KEM) together with a standard symmetric encryption scheme. The KEM has one key k_e per epoch e, and for each ciphertext it wraps an "inner" key x under which the actual message is encrypted. During an update, where the tweak is given by two keys (k_e, k_{e+1}) of subsequent epochs, all inner keys are simply un-wrapped using k_e and re-wrapped under the new key k_{e+1}.

This scheme is used in practical data-at-rest protection at Cloud storage providers. The keys are, however, managed within the Cloud storage systems. Not all nodes are equal; there are nodes that have access to the keys, and nodes that store the encrypted data.[3] In this scenario, it is acceptable to have the proxy nodes perform the updates. The scheme is not applicable for outsourcing encrypted data.

We describe the algorithms in a slightly different way to consider $\mathsf{SE\text{-}KEM}$ as a ciphertext-independent updatable encryption scheme. The algorithms are described in more detail as the scheme $\mathsf{UE}_{\mathsf{SE\text{-}KEM}} = \mathsf{UE}$ as follows.

$\mathsf{UE.setup}(\lambda)$**:** return $k_0 \leftarrow \mathsf{SE.kgen}(\lambda)$.

$\mathsf{UE.next}(\boldsymbol{k_e})$**:** $k_{e+1} \leftarrow \mathsf{SE.kgen}(\lambda)$, $\Delta_{e+1} \leftarrow (k_e, k_{e+1})$, return (k_{e+1}, Δ_{e+1}).

$\mathsf{UE.enc}(\boldsymbol{k_e}, m)$**:** $x \leftarrow \mathsf{SE.kgen}(\lambda)$, return $C_e \leftarrow (\mathsf{SE.enc}(k_e, x), \mathsf{SE.enc}(x, m))$.

$\mathsf{UE.upd}(\Delta_{e+1}, C_e)$**:** parse $C_e = (C^1, C^2)$, and $\Delta_{e+1} = (k_e, k_{e+1})$,
return $C_{e+1} \leftarrow (\mathsf{SE.enc}(k_{e+1}, \mathsf{SE.D}((,k)_e, C^1)), C^2)$.

$\mathsf{UE.dec}(\boldsymbol{k_e}, C_e)$**:** parse $C_e = (C^1, C^2)$, return $\mathsf{SE.D}((,\mathsf{SE}).\mathsf{D}((,k), C^1), C^2)$.

[3] In OpenStack Swift, for instance, the "proxy server" nodes have access to the keys, whereas the role of the "object server" nodes is to store the ciphertext.

While this scheme is very similar to the hybrid AE as described by Everspaugh el al. [EPRS17], our description differs in that the tweak is independent of the ciphertext, and consists of the keys (k_e, k_{e+1}) used for encryption in epochs e and $e+1$. In Cloud storage systems where the keys for data-at-rest encryption are managed within the Cloud, this is a faithful description of the real behavior.

The security that can be offered by such a solution is necessarily limited. First, if the adversary obtains a challenge in epoch e and also sees one of the tweaks in epochs e or $e+1$, the IND-ENC security is immediately broken. Furthermore, as the ciphertext update does not re-encrypt the second component, the ciphertexts are linkable through the epochs, i.e., SE-KEM cannot achieve any form of IND-UPD security. Still, we show that under the described (strict) constraints, the scheme guarantees IND-ENC security.

Theorem 2.5.8 ($\mathsf{UE}_{\mathsf{SE\text{-}KEM}}$ **is weakly** IND-ENC **secure)** *Let* SE *be an* IND-CPA*-secure encryption scheme, then* $\mathsf{UE}_{\mathsf{SE\text{-}KEM}}$ *is* IND-ENC*-secure if the following additional condition holds: For any challenge-equal epoch* $e \in \mathscr{C}^*$, $\mathscr{A}$ *may not obtain a tweak in epochs* e *or* $e+1$.

2.5.4 Conclusion and open problems

We provided a comprehensive model for ciphertext-independent updatable-encryption schemes, complementing the recent work of Everspaugh et al. [EPRS17] that focuses on ciphertext-dependent schemes. Ciphertext-independent schemes are clearly superior in terms of efficiency and ease-of-use when key rotation is required for large volumes of ciphertexts, whereas ciphertext-dependent solutions give a more fine-grained control over the updatable information.

We formalized updatable encryption and its desired properties in the strict sequential manner it will be used, avoiding the ambiguity of previous security models. Our two notions IND-ENC and IND-UPD guarantee that fresh encryptions and updated ciphertext are secure even if an adversary can adaptively corrupt several keys and tweaks before and after learning the ciphertexts.

Somewhat surprisingly, and contradictory to the claim in [EPRS17], the XOR-KEM scheme is not a secure ciphertext-independent schemes in such a strong sense. For the (existing) schemes 2ENC, BLMR, BLMR+, and SE-KEM, we formalized the security of the schemes by specifying precisely the conditions on the adversary under which a weak form of IND-ENC and IND-UPD security is achieved. We also specified a scheme that builds on ElGamal and is based on previous work on proxy re-encryption. By additionally exploiting the algebraic structure of the underlying groups, instead of using the key-homomorphic PRF abstraction as in previous works, we were able to build a scheme that fully achieves our strong security notions while being at least as efficient as exiting schemes that are either weaker or require ciphertext-dependent tweaks.

All schemes we analyze allow to infer tweaks from keys, and enable bi-directional updates of ciphertexts and keys, whereas an ideal updatable encryption scheme should only allow uni-directional updates of ciphertexts. Building such an ideal scheme is related to the open challenge of building proxy re-encryption schemes that are uni-directional multi-hop and collusion-resistant. Yet, while most proxy re-encryption work is in the public-key setting, updatable encryption has secret keys, so the construction of schemes with similar properties may be easier and is an interesting and challenging open problem.

3. Efficient and private data access

Task T2.3 focuses on efficient and private access to data stored at an external provider. Task T2.3 aims to develop: *i)* techniques for enabling effective and efficient access to data stored in the Cloud, while protecting the confidentiality of access operations, and *ii)* solutions providing integrity guarantees of outsourced data. The work in T2.3 first focused on the effective and efficient access objective. The innovative contribution by ESCUDO-CLOUD for the design of indexing techniques that guarantee access confidentiality is two-fold (Section 3.2): we extended the shuffle index technique for supporting range queries as well as update operations; and we proposed an alternative tree-based dynamically allocated data structure that does not require any storage at the client side. The work in T2.3 then addressed the integrity problem. The innovative contribution by ESCUDO-CLOUD for the design of solutions for data integrity consists in the definition of a novel encryption mode that enables the data owner to verify the integrity of externally stored data (Section 3.3).

3.1 ESCUDO-CLOUD Innovation

Task T2.3 produced several advancements over the state-of-the-art.

- An extended version of the shuffle index structure (illustrated in Deliverable D6.2) which supports range queries as well as update operations (i.e., insertion, removal, and update). The proposed solution is based on the idea of splitting in a probabilistic way the nodes visited while performing access operations in such a way to make read and write accesses indistinguishable.
- A novel dynamically allocated tree-based data structure (presented in Deliverable D2.3), alternative to the shuffle index, that provides data and access confidentiality while not requiring storage at the client side.
- A novel encryption mode, Mix&Slice (presented in Section 3.3 of this deliverable), that guarantees complete interdependence (mixing) among the bits of the encrypted content of the input plaintext data, that is, the value of each bit in the resulting encrypted content depends on every bit of the original plaintext content. This encryption mode can be profitably used to provide integrity guarantees of data stored in the Cloud.

The results obtained by this task have been published in [BDF$^+$16b, DFM$^+$16, DFP$^+$15].

3.2 A Dynamic Tree-Based Data Structure for Access Confidentiality

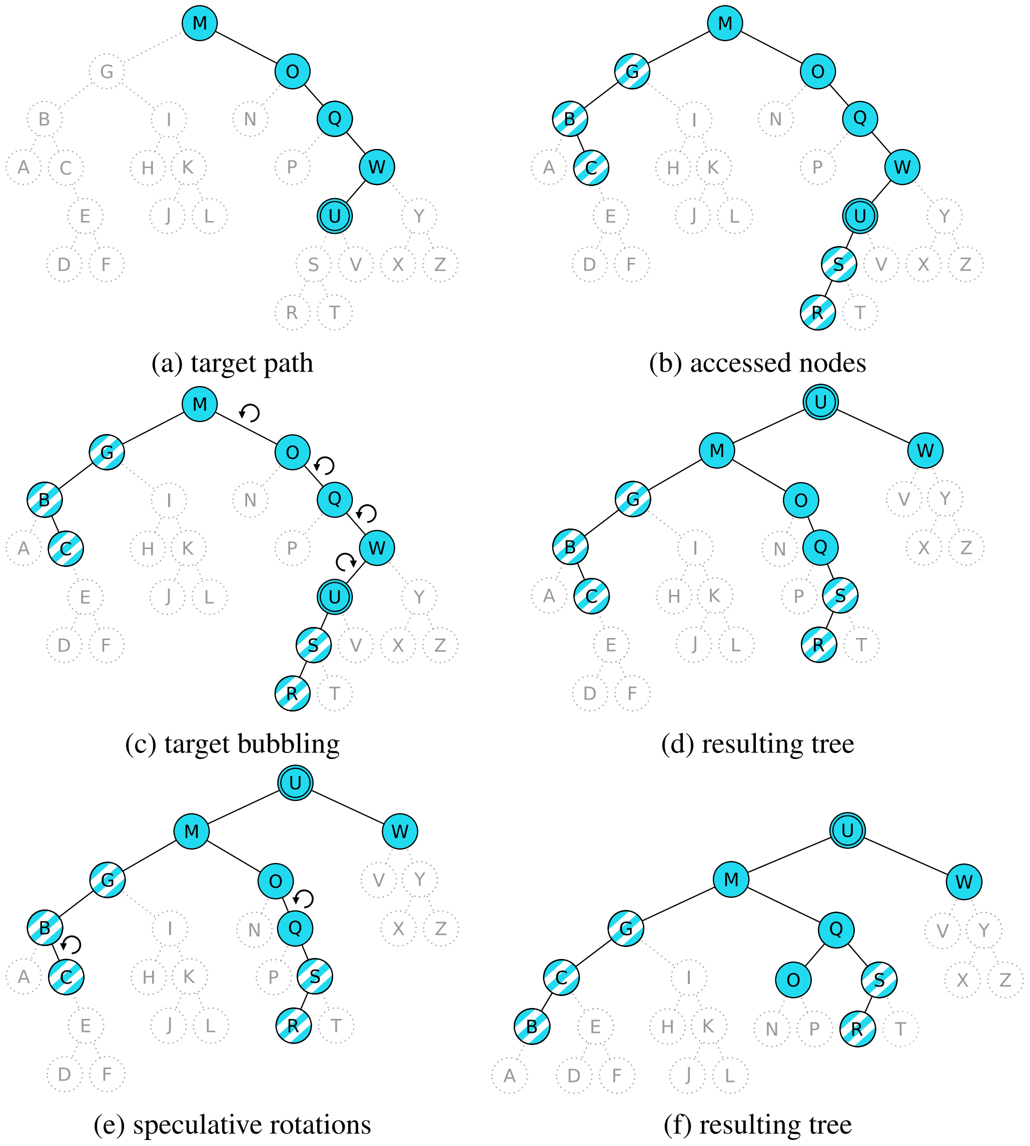
The first issue addressed in this task is the definition of novel approaches for protecting access confidentiality, while providing efficient access to the data stored at an external provider. The approach proposed within ESCUDO_CLOUD to address this problem, and discussed in the details in Deliverable D2.3, is based on the definition of a novel tree-based dynamically allocated data structure. Differently from existing solutions protecting access confidentiality (e.g., ORAM-based approaches [SvS$^+$13, SS13] and the shuffle index [DFP$^+$15]), our solution does not rely on client-side storage [DFM$^+$16]. This provides a two-fold advantage: *i)* it does not require the client to commit storage resources for accessing data; and *ii)* it supports accesses by multiple clients.

To the aim of supporting efficient accesses, outsourced data are organized in a binary search tree. The nodes in the tree are buckets, each storing a set of resources. The mapping function, associating each resource with the bucket storing it, is a non-invertible and non-order preserving function, defined in such a way to guarantee a balanced distribution of the resources among the buckets. Since the mapping function is not invertible, exposure of the bucket index does not expose sensitive values. Also, since the mapping function is not order preserving, the binary search tree efficiently supports searches over the outsourced data collection without revealing the relative order among data. The buckets composing the binary search tree are encrypted at the client side before storage at the provider. The provider receives a set of encrypted blocks for storage and it can neither access its plaintext content nor visit the tree structure, as pointer to children are encrypted together with the node content.

Encryption guarantees protection of the confidentiality of data at rest and of single accesses to data, while it does not provide protection to access confidentiality. Indeed, a provider observing accesses to encrypted blocks can reconstruct the frequency of accesses to physical blocks and exploit such a knowledge to infer the corresponding plaintext data content. The proposed approach combines the following four protection techniques to provide access confidentiality.

- **Uniform accesses**. All the accesses download from the provider the same number of blocks, independently from the level where the target of the search operation is located. The number of accessed blocks is fixed to $h+2$, where $h = \lfloor 2\log(N) \rfloor$ is the maximum height fixed for the binary search tree (with N the number of nodes in the tree). If the path to the target node is shorter than $h+2$, the access algorithm downloads a set of *filler nodes*, that is, of nodes that are not along the path to the target. To guarantee that filler nodes are not recognizable as such, they are randomly chosen among the children of already accessed nodes, and nodes (be them along the path to the target or fillers) are dowloaded level by level. This guarantees that any of the $h+2$ accessed nodes could be the target of the access. Consider, as an example, a search for value U in the binary search tree in Figure 3.1 with $N = 26$ and $h+2 = 10$. Since the path to the target node includes only 5 nodes (the blue ones in Figure 3.1(a)), the search is complemented with 5 filler nodes (striped nodes in Figure 3.1(b)). Note that any of the 10 downloaded nodes could be the target of the access since nodes along the path to the target are indistinguishable from filler nodes.

- **Target bubbling**. After each access, the target node is moved up (close to the root) in the tree by properly rotating the nodes along its path. This technique protects against repeated accesses. Indeed, if two subsequent searches look for the same target, the second access will find the target high in the tree and will therefore choose a high number of filler nodes. Hence, the two searches will visit two different sets of nodes, reducing the effectiveness of intersection attacks (i.e., of attacks that exploit the common downloaded blocks in subsequent accesses to infer the target of the searches). Target bubbling has also the advantage of changing the topology of the binary tree structure, further enhancing protection. With reference to the example in Figure 3.1, the nodes along the path to U are rotated as illustrated in



(a) target path

(b) accessed nodes

(c) target bubbling

(d) resulting tree

(e) speculative rotations

(f) resulting tree

Figure 3.1: An example of access to the binary search tree

Figure 3.1(c), obtaining the binary tree in Figure 3.1(d), where U is the root. A search for U over this tree could visit any subtree of 10 nodes rooted at U, thus considerably enhancing protection guarantees.

- **Speculative rotations**. Each access operation, because of target bubbling, can increase or decrease the heigh of the tree by one. To guarantee that the height of the tree remains within the limit of $h = \lfloor 2\log(N) \rfloor$, speculative rotations possibly rotate accessed nodes, when it could be useful for reducing the height of the tree. Clearly, speculative rotations do not operate on the target node (or its ancestors) because this would possibly nullify (or mitigate the advantages of) target bubbling. Even if speculative rotations do not represent a protection technique per se, they provide benefits as they change the tree topology (and hence paths reaching nodes). With reference to the example in Figure 3.1, the rotations in Figure 3.1(e) could reduce the height of the tree. The tree resulting after the application of speculative rotations is illustrated in Figure 3.1(f), which has a completely different topology than the

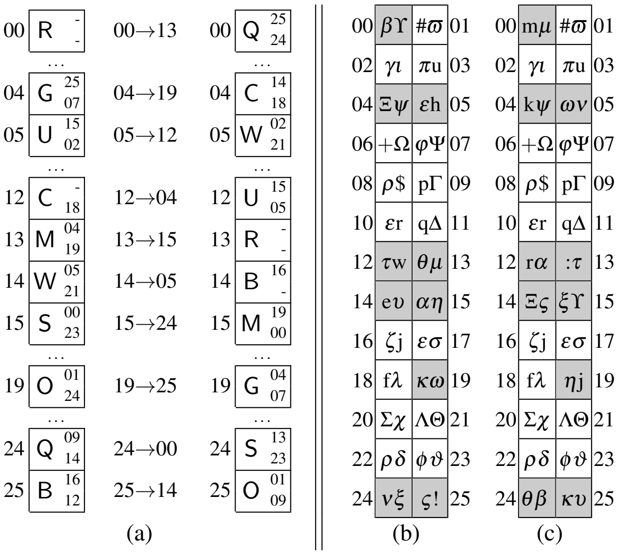


Figure 3.2: An example of physical re-allocation (a) and of view of the server before (b) and after (c) the access in Figure 3.1

tree in Figure 3.1(a) on which the access operated.

- **Physical re-allocation**. At each access, the allocation of all the accessed nodes to physical blocks is changed. Re-allocation implies the need to decrypt and re-encrypt, concatenated with a different random salt to make the re-allocation untraceable by a possible observer, all the accessed nodes. Also, it requires to update the pointer to children in the parents of re-allocated nodes. Note that, since all the accessed nodes are in a parent-child relationship, this does not require to download additional nodes. By changing the node-block correspondence at every access, physical re-allocation prevents the provider from determining whether two accesses visited the same node (sub-path) by observing accesses to physical blocks, and hence it prevents accumulating information on the topology of the tree. Indeed, accesses aimed at the same node will visit different blocks (and vice versa). Figure 3.2(a) illustrates an example of physical re-allocation of the nodes/blocks accessed by the search Figure 3.1, illustrating the nodes content before and after re-allocation. Figure 3.2(b) illustrates the view of the provider over the blocks composing the binary search tree, and its observations of accessed blocks (in gray).

As discussed in detail in Deliverable D2.3, the combined adoption of the protection techniques illustrated above, which imply both physical re-allocation and logical restructuring of the binary search tree, guarantees access confidentiality. Indeed, it makes skewed profiles of access to the plaintext data statistically indistinguishable from uniform access profiles [DFM^{+}16].

3.3 Mix&Slice for Resource Protection

The second issue addressed in this task aims at providing solutions for protecting data integrity in the Cloud. ESCUDO-CLOUD developed a novel encryption mode, which enables the data owner to verify the integrity of her resources. The basic idea of our approach is to provide an encrypted representation of the resources that guarantees complete interdependence (*mixing*) among the bits of the encrypted content. Such a guarantee is ensured by using different rounds of encryption, while carefully selecting their input to provide complete mixing, meaning that the value of each bit in the resulting encrypted content depends on every bit of the original plaintext content. In this way, tampering or unavailability of even a small portion of the encrypted version of a resource completely prevents the reconstruction of the resource or even of portions of it, and signals integrity violations [BDF$^+$16b].

3.3.1 Blocks, mini-blocks, and macro-blocks

The basic building block of our approach is the application of a symmetric block cipher. A symmetric cryptographic function operating on blocks guarantees complete dependency of the encrypted result from every bit of the input and the impossibility, when missing some bits of an encrypted version of a block, to retrieve the original plaintext block (even if parts of it are known). The only possibility to retrieve the original block would be to perform a brute-force attack attempting all the possible combinations of values for the missing bits. For instance, modern encryption functions like AES guarantee that the absence of i bits from the input (plaintext) and of o bits from the output (ciphertext) does not permit, even with knowledge of the encryption key k, to properly reconstruct the plaintext and/or ciphertext, apart from performing a brute-force attack generating and verifying all the $2^{\min(i,o)}$ possible configurations for the missing bits [ABM14].

Clearly, the larger the number of bits that are missing in the encrypted version of a block, the harder the effort required to perform a brute-force attack, which requires attempting 2^x possible combinations of values when x bits are missing. Such *security parameter* is at the center of our approach and we explicitly identify a sequence of bits of its length as the atomic unit on which our approach operates, which we call *mini-block*. Applying block encryption with explicit consideration of such atomic unit of protection, and extending it to a coarser-grain with iterative rounds, our approach identifies the following basic concepts.

- *Block*: a sequence of bits input to a block cipher (it corresponds to the classical block concept).

- *Mini-block*: a sequence of bits, of a specified length, contained in a block. It represents our *atomic unit* of protection (i.e., when removing bits, we will operate at the level of mini-block removing all its bits).

- *Macro-block*: a sequence of blocks. It allows extending the application of block cipher on sequences of bits larger than individual blocks. In particular, our approach operates *mixing* bits at the macro-block level, extending protection to work against attacks beyond the individual block.

Our approach is completely parametric with respect to the size (in terms of the number of bits) that can be considered for blocks, mini-blocks, and macro-blocks. The only constraints are for the size of a mini-block to be a divisor of the size of the block (aspect on which we will elaborate later on) and for the size of a macro-block to be a product of the size of a mini-block and a power of the number of mini-blocks in a block (i.e., the ratio between the size of a block and the size of a mini-block). In the following, for concreteness and simplicity of the figures, we will illustrate our examples assuming the application of AES with blocks of 128 bits and mini-blocks of 32 bits, which corresponds to having 4 mini-blocks in every block and therefore operating on macro-blocks of size $32 \cdot 4^x$, with x arbitrarily set. In the following, we will use *msize*, *bsize*, *Msize*

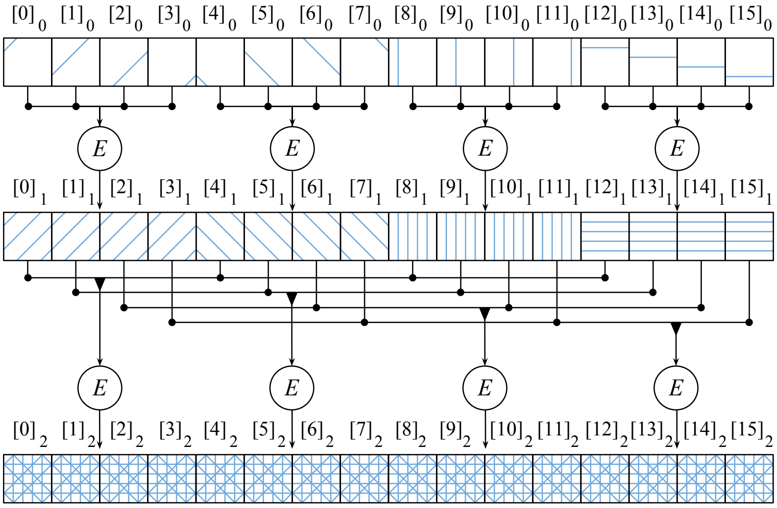


Figure 3.3: An example of mixing of 16 mini-blocks assuming $m = 4$

to denote the size (in bits) of mini-blocks, blocks, and macro-blocks, respectively. We will use $b_j[i]$ ($\mathrm{M_j}[i]$, resp.) to denote the i-th mini-block in a block b_j (macro-block $\mathrm{M_j}$, resp.). We will simply use notation $[i]$ to denote the i-th mini-block in a generic bit sequence (be it a block or macro-block), and $[[j]]$ to denote the j-th block. In the encryption process, a subscript associated with a mini-block/block denotes the round that produced it.

3.3.2 Mixing

The basic step of our approach (on which we will iteratively build to provide complete mixing within a macro-block) is the application of encryption at the block level. This application is visible at the top of Figure 3.3, where the first row reports a sequence of 16 mini-blocks ($[0],\ldots,[15]$) composing 4 blocks. The second row is the result of block encryption on the sequence of mini-blocks. As visible from the pattern-coding in the figure, encryption provides mixing within each block so that each mini-block in the result is dependent on every mini-block in the same input block. In other words, each $[i]_1$ is dependent on every $[j]_0$ with (i div 4) = (j div 4).

One round of block encryption provides mixing only at the level of block. With reference to our example, mixing is provided among mini-blocks $[0]_0\ldots[3]_0$, $[4]_0\ldots[7]_0$, $[8]_0\ldots[11]_0$, and $[12]_0\ldots[15]_0$, respectively. Absence of a mini-block from the result will prevent reconstruction only of the plaintext block including it, while not preventing the reconstruction of all the other blocks. For instance, with reference to our example, absence of $[0]_1$ will prevent reconstruction of the first block (mini-blocks $[0]_0,\ldots,[3]_0$) but will not prevent reconstruction of the other three blocks (mini-blocks $[4]_0,\ldots,[15]_0$). Protection at the block level is clearly not sufficient in our context, where we expect to manage resources of arbitrarily large size and would like to provide the guarantee that the lack of any individual mini-block would imply the impossibly (apart from performing a brute-force attack) of reconstructing any other mini-block of the resource. The concept of macro-block, and accurate extension of block ciphering to operate across blocks, allows us to provide mixing on an arbitrarily long sequence of bits (going much above the size of the block).

The idea is to extend mixing to the whole macro-block by the iterative application of block encryption on, at each round, blocks composed of mini-blocks that are representative (i.e., belong to the result) of different encryptions in the previous round. Before giving the general definition of our approach, let us discuss the simple example of two rounds illustrated in Figure 3.3,

```
**Mix**(M)
for $i := 1,\ldots,x$ do                                              /* at each round $i$ */
    $span := m^i$                                                     /* number of mini-blocks in a mixing */
    $distance := m^{i-1}$                                             /* leg of mini-blocks input to an encryption */
    for $j := 0,\ldots,b-1$ do                                        /* each $j$ is an encryption */
                                                                      /* identify the input to the $j$-th encryption picking, */
                                                                      /* within each span, mini-blocks at leg distance */
        let block be the concatenation of all mini-blocks $[l]$
            s.t. $(l \mod distance) = j$ and
                $(j \cdot m)$ div $span$ = $l$ div $span$
        $[[j]]_i := E(k, block)$                                      /* write the result as the $j$-th block in output */
```

Figure 3.4: Mixing within a macro-block M

where $[0]_1,\ldots,[15]_1$ are the mini-blocks resulting from the first round. The second round would apply again block encryption, considering different blocks each composed of a representative of a different computation in the first round. To guarantee such a composition, we define the blocks input to the four encryption operations as composed of mini-blocks that are at distance 4 (=*m*) in the sequence, which corresponds to say that they resulted from different encryption operations in the previous round. The blocks considered for encryption would then be $\langle[0]_1[4]_1[8]_1[12]_1\rangle$, $\langle[1]_1[5]_1[9]_1[13]_1\rangle$,$\langle[2]_1[6]_1[10]_1[14]_1\rangle$,$\langle[3]_1[7]_1[11]_1[15]_1\rangle$. The result would be a sequence of 16 mini-blocks, each of which is dependent on each of the 16 original mini-blocks, that is, the result provides mixing among all 16 mini-blocks, as visible from the pattern-coding in the figure. With 16 mini-blocks, two rounds of encryption suffice for guaranteeing mixing among all of them. Providing mixing for larger sequences clearly requires more rounds. This brings us to the general formulation of our approach operating at the level of macro-block of arbitrarily large size (the example just illustrated being a macro-block of 16 mini-blocks).

To ensure the possibility of mixing, at each round, blocks composed of mini-blocks resulting from different encryption operations of the previous round, we assume a macro-block composed of a number of mini-blocks, which is the power of the number (*m*) of mini-blocks in a block. For instance, with reference to our running example where blocks are composed of 4 mini-blocks (i.e., *m*=4), macro-blocks can be composed of 4^x mini-blocks, with an arbitrary *x* (*x*=2 in the example of Figure 3.3). The assumption can be equivalently stated in terms of blocks, where the number of blocks *b* will be 4^{x-1}. Any classical padding solution can be employed to guarantee such a requirement, if not already satisfied by the original bit sequence in input.

Providing mixing of a macro-block composed of *b* blocks with b=m^{x-1} requires *x* rounds of encryption each composed of *b* encryptions. Each round allows mixing among a number *span* of mini-blocks that multiplies by *m* at every round. At round *i*, each encryption *j* takes as input *m* mini-blocks that are within the same *span* (i.e., the same group of m^i mini-blocks to be mixed) and at a *distance* (m^{i-1}). Figure 3.4 presents the mixing procedure. To illustrate, consider the example in Figure 3.3, where blocks are composed of 4 mini-blocks (*m*=4) and we have a macro-block of 16 mini-blocks, that is, 4 blocks (*b*=4). Mixing requires $x = 2$ rounds of encryption ($16 = 4^2$), each composed of 4 (*b*) encryptions operating on 4 (*m*) mini-blocks. At round 1, the *span* is 4 (i.e., mixing operates on chunks of 4 mini-blocks) and mini-blocks input to an encryption are taken at distance 1 within each span. At round 2, the *span* is 16 (all mini-blocks are mixed) and mini-blocks input to an encryption are taken at *distance* 4 within each *span*. Let us consider, as an another example, a macro-block composed of 64 mini-blocks (i.e., 16 blocks). Mixing requires

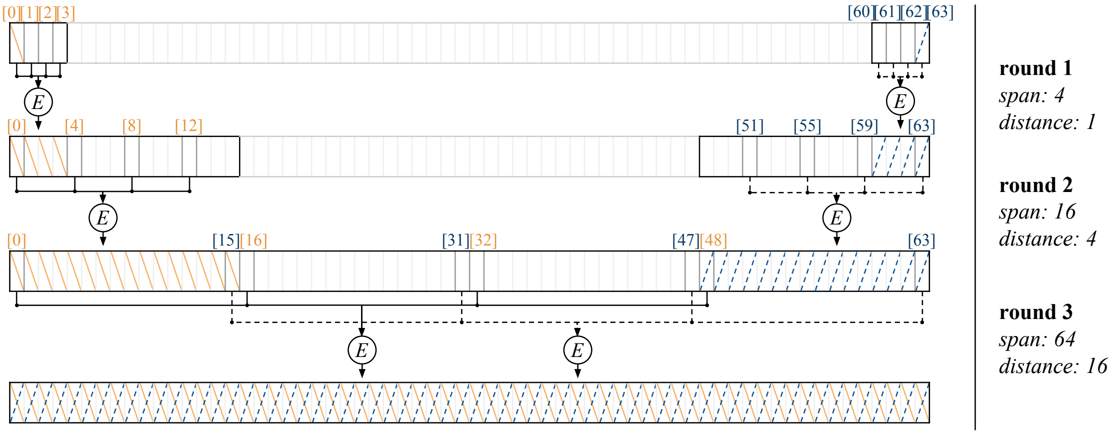


Figure 3.5: Propagation of the content of mini-blocks [0] and [63] in the mix process

3 rounds. The first two rounds would work as before, with the second round producing mixing within chunks of 16 mini-blocks. The third round would then consider a *span* of all the 64 mini-blocks and mini-blocks input to an encryption would be the ones at *distance* 16.

At each round i, mini-blocks are mixed among chunks of m^i mini-blocks, hence ensuring at round x, mixing of the whole macro-block composed of m^x mini-blocks.

Figure 3.5 captures this concept by showing the mixing of the content of the first ([0]) and last ([63]) mini-blocks of the macro-block at the different rounds, given by the encryption to which they (and those mini-blocks mixed with them in previous rounds) are input, showing also how the two meet at the step that completes the mixing. While for simplicity the figure pictures only propagation of the content of two mini-blocks, note that at any step they (just like other mini-blocks) actually carry along the content of all the mini-blocks with which they mixed in previous rounds. Given a macro-block M with m^x mini-blocks (corresponding to b blocks), the following two properties hold: *1)* a generic pair of mini-blocks $[i]$ and $[j]$ mix at round r with i div $m^r = j$ div m^r; and *2)* x rounds bring complete mixing. In other words, the number of encryption rounds needed to mix a macro-block with $m \cdot b$ mini-blocks is $\log_m(m \cdot b)$.

An important feature of the mixing is that the number of bits that are passed from each block in a round to each block in the next round is equal to the size of the mini-block. This guarantees that the uncertainty introduced by the absence of a mini-block at the first round (2^{msize}) maps to the same level of uncertainty for each of the blocks involved in the second round, and iteratively to the next rounds, thanks to the use of AES at each iteration. This implies that a complete mixing of the macro-block requires at least $\log_m(m \cdot b)$ rounds, that is, the rounds requested by our technique.

Another crucial aspect is that the representation after each round has to be of the same size as the original macro-block. Since, in our approach, each round produces a representation that has the same macro-block size, the user has no benefit in aiming to attack one round compared to another.

We note that an interpretation of the proposed mixing is that it extends the ability of protecting the correspondence between input and output of a block cipher to blocks of arbitrary size. An alternative approach that we considered to obtain this result was based on the use of a Feistel architecture [LR88], which is known to be an effective technique for the construction of block ciphers. The approach uses, as the *round* function of the Feistel architecture, a block cipher. The approach can be applied iteratively, doubling the block size at every iteration. The analysis we performed showed that this approach would lead to less efficiency compared to the solution proposed in this section, with a number of invocations of the basic block cipher equal to $2 \cdot \log_m(m \cdot b)$. The Feistel-based approach can be adopted when the mini-block size desired for security goes beyond the block size of the available block cipher. Similarly, symmetric cryptosystems operating on large blocks can support larger mini-blocks and also reduce the number of rounds of our approach. For instance, AESQ [BK14] shuffles 4 AES blocks and could be used as a 512-block cipher in our structure.

When resources are extremely large (or when access to a resource involves only a portion of it) considering a whole resource as a single macro-block may be not desirable. Even if only with a logarithmic dependence, the larger the macro-block the more the encryption (and therefore decryption to retrieve the plaintext) rounds required. Also, encrypting the whole resource as a single macro-block implies its complete download at every access, when this might actually not be needed for service.

Accounting for this, we do not assume a resource to correspond to an individual macro-block, but assume instead that any resource can be partitioned into M macro-blocks, which can then be mixed independently. The choice of the size of macro-blocks should take into consideration the performance requirements of both the data owner (for encryption) and of clients (for decryption), and the possible need to serve fine-grained retrieval of content. This requirement can be then efficiently accommodated independently encrypting (i.e., mixing) different portions of the resource, which can be downloaded and processed independently.

Encryption of a resource would then entail a preliminary step cutting the resource in different, equally sized, macro-blocks on which mixing operates. To ensure the mixed versions of macro-blocks be all different, even if with the same original content, the first block of every macro-block is XORed with an *initialization vector* (*IV*) before starting the mixing process. Since mixing guarantees that every block in a macro-block influences every other block, the adoption of a different initialization vector for each macro-block guarantees indistinguishability among their encrypted content. The different initialization vectors for the different blocks can be obtained by randomly generating a vector for the first macro-block and then incrementing it by 1 for each of the subsequent macro-blocks in the resource, in a way similar to the CTR mode [Dwo01]. Figure 3.6(a) illustrates such process.

3.3.3 Slicing

The starting point for introducing mixing is to ensure that each single bit in the encrypted version of a macro-block depends on every other bit of its plaintext representation, and therefore that removing any one of the bits of the encrypted macro-block would make it impossible (apart from brute-force attacks) to reconstruct any portion of the plaintext macro-block. Such a property operates at the level of macro-block. Hence, if a resource (because of size or need of efficient fine-grained access) has been partitioned into different macro-blocks, removal of a mini-block would only guarantee protection of the macro-block to which it belongs, while not preventing reconstruction of the other macro-blocks (and therefore partial reconstructions of the resource). Resource protection can be achieved if, for each macro-block of which the resource is composed, a mini-block is removed. This observation brings to the second concept giving the name to our approach, which is *slicing*. Slicing the encrypted resource consists in defining different *fragments* such that a fragment contains a mini-block for each macro-block of the resource, no two fragments contain the same mini-block, and for every mini-block there is a fragment that contains it. To ensure all this, as well as to simplify management, we slice the resource simply putting in the same frag-

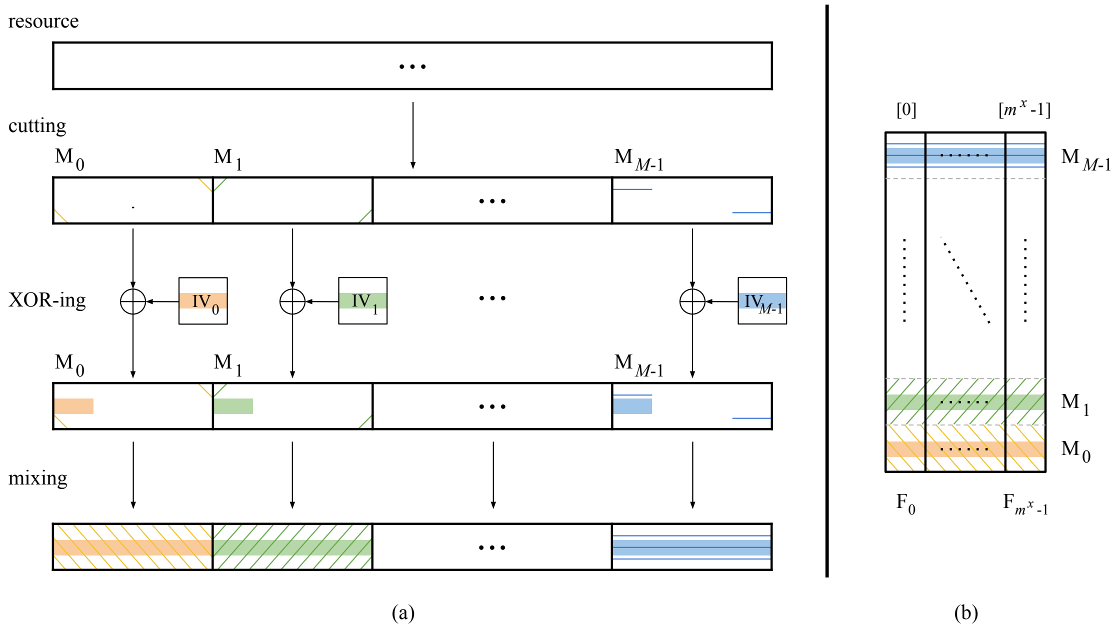


Figure 3.6: From resource to fragments

```
Encrypt
cut R in $M$ macro-blocks $M_0,\ldots,M_{M-1}$
apply padding to the last macro-block $M_{M-1}$
$IV$ := randomly choose an initialization vector
for $i = 0,\ldots,M-1$ do                                    /* encrypt macro-blocks */
    $M_i[[1]] := M_i[[1]] \oplus IV$                        /* XOR the first block with the IV */
    Mix($M_i$)                                               /* encrypt the macro-block */
    $IV := IV + 1$                           /* initialization vector for the next macro-block */
    for $j = 0,\ldots,m^x-1$ do                                            /* slicing */
        $F_j[i] := M_i[j]$
```

Figure 3.7: Algorithm for encrypting a resource R

ment the mini-blocks that occur at the same position in the different macro-blocks. Slicing and fragments are defined as follows.

Definition 3.3.1 (Slicing and fragments) *Let* R *be a resource and* $M_0,\ldots,M_{M-1}$ *be its (individually mixed) macro-blocks, each composed of* $(m \cdot b)$ *mini-blocks. Slicing produces* $(m \cdot b)$ *fragments for* R *where* $F_i = \langle M_0[i],\ldots,M_{M-1}[i]\rangle$*, with* $i = 1,\ldots,(m \cdot b)$.

Figure 3.6(b) illustrates the slicing process and Figure 3.7 presents the procedure for encrypting a resource R. R is first cut into M macro-blocks and an initialization vector is randomly chosen. The first block of each macro-block is then XOR-ed with the initialization vector, which is incremented by 1 for each macro-block. The macro-block is then encrypted with a mixing process (as shown in Figure 3.4). Encrypted macro-blocks are finally sliced into fragments.

3.4 Summary

The work in Task T2.3 first focused on providing solutions for effectively and efficiently accessing encrypted data stored in the Cloud, while preserving the confidentiality of data and accesses to the provider. ESCUDO-CLOUD analyzed and extended the shuffle index technique (Deliverable D6.2), and proposed a novel alternative dynamically allocated structure (Deliverable D2.3) that provides access confidentiality, while not requiring local storage at the client side.

Task T2.3 also analyzed the problem of providing integrity guarantees. ESCUDO-CLOUD proposed a novel encryption mode (introduced in this deliverable). This encryption mode guarantees complete interdependence among the bits of the ciphertext and can be used by the data owner to verify the integrity of data stored in the Cloud.

4. Requirements-based threat analysis

The work in Task 2.4 explored the potential of developing a Use Case (UC) requirements based threat analysis (RBTA) approach. The intent was to investigate if a threat analysis process on the UC requirements could provide sufficient information to estimate the risks of data ownership breaches and related vulnerabilities. The task resulted in two significant accomplishments. Firstly, the developed RBTA process was able to establish the viability of identifying threats based on the UC requirements analysis. Building on the basic viability of RBTA result, D2.4 developed a systematic analytic technique that enumerates the set of UC requirements and then determines all possible direct/indirect dependencies across them to conduct a generalized threat analysis from their requirements. The approach was validated for its effectiveness on the ESCUDO-CLOUD UCs (in D2.4), and was subsequently generalized to a holistic Cloud model to apply to other UCs external to ESCUDO-CLOUD as based on the availability of their requirements.

D2.4 reported on the T2.4 activities upto M27 that explored the potential of conducting threat analysis based on requirements. The summary of D2.4, and also the activities subsequent to M27 are detailed therein.

4.1 ESCUDO-CLOUD Innovation

Task T2.4 produced several advancements over the state of the art.

- *Novel 3-stage Dependency Model* T2.4 established the viability of RBTA for its applicability to conduct threat analysis on the ESCUDO-CLOUD UC's. The innovation was in developing a novel 3-stage dependency model [TMT$^+$16] and showing its capability for (a) validating each UC requirements by checking the existence of conflicts that occur due to to different dependency relations between requirements, and (b) identifying threats by ascertaining and visualizing the dependencies across the UC requirements.

  The applicability of the proposed model was demonstrated on each of the ESCUDO-CLOUD UC's and also shown to be able to identify specific cases of requirements violation. This was complemented by an UC level quantification of the threat likelihood/severity. These results were reported in D2.4 at M27.

- *Automation of Threat Analysis* T2.4 developed the basis behind the automation of threat analysis. A limitation of the current threat modeling approaches is that they require significant manual work to identify and assess potential threats. Additionally, the analysis often lacks focus on the primary assets that require protection. T2.4 developed an efficient architectural threat modeling approach supported by a tool that allows partly automating and focusing security assessment based on assets and security objectives in the system and high-level design enriched with security aspects. This was reported in D2.4 at M27.

- *Holistic Multi-Level Cloud Threat Model* Each ESCUDO-CLOUD UC represents a unique instantiation of the Cloud. In order to generalize the proposed RBTA to a fuller spectrum of Cloud UC's (including external to ESCUDO-CLOUD), we have proposed (a) a holistic Cloud model at the operational level, and (b) correspondingly developed a novel generalized multi-level threat assessment approach that can consider both insider and outsider attackers across the varied Cloud UC's. This innovation is reported in the subsequent subsections and was also published in [MLS17].

4.2 Extending RBTA to a Holistic Cloud Threat Model - From Dependency Analysis to Exploring Cloud Attack Surfaces

In D2.4, the UC requirements and risk assessment tables of each UC (individually and collectively) were specified to establish the viability of a requirements based threat analysis. However, the typical dependency relations across requirements increases the difficulty for the customers to specify their requirements as these relations can easily introduce conflicts. Accordingly, we developed techniques to systematically conduct the process of ascertaining and visualizing the dependencies across the UC requirements to identify threats. However, this is specific to each UC and a holistic approach to address generic Cloud UC's is desired.

One of the advocated methods to provide security assurance and alleviate customer's security concerns is performing threat analysis i.e., evaluating the system for vulnerabilities that can be exploited by adversaries. Threat Analysis (TA) [SS04, MLY05, OSC06], is an approach to investigate potential attacks that can undermine security goals. However, due to the plethora of services/technologies involved in the Cloud operational stack and interfaces across varied resources and customers, the current threat analysis approaches of Cloud typically focus on a particular service or consider only a particular technology [SHJ$^+$11, VS12, Rut08, PBSL13]. Therefore, most of these schemes reveal attack surfaces that are pertinent to only that particular service/technology without contemplating the holistic operations of the Cloud. Consequently, the current TA approaches fail to analyze behavioral repercussions of the vulnerable service/technology across the varied levels of operational stack in the Cloud ecosystem.

Hence, a comprehensive threat analysis of Cloud is desired that can (a) comprehensively analyze malicious behaviors stemming from interactions of the vulnerable service across the multi-level operational stack, and (b) correspondingly enumerate the multi-level attack surface exploitability by attackers.

To address this, a design of a multi-level modeling of the Cloud operations is devloped in the subequent sections. The devolped design comprehensively delineates the lifecyle and interaction of the services involved in the Cloud ecosystem. We surveyed the process of launching a virtual machine in multiple open source Cloud computing environments [SAE12, NWG$^+$09, MLM11] and (a) profile behavioral characteristics of the services and their interconnections, (b) analyze detailed flow of information among the services across varied levels of the Cloud, and (c) how the information is processed to launch a virtual machine. We utilize this information in the approach proposed in this section, using Petri Nets, to identify the base operational states, enumerating the “normal" sequence of Cloud operations along with the triggers that provide the state transitions.

The obtained Petri Nets model is the basis to identify multi-level vulnerabilities not recognizable by traditional/single-level threat analysis. Thus, the next aspect is to investigate anomalous sequences of operations from varied attacker profiles e.g., insider/outsider attacker. This delineation is desired as different attackers can have disparate states visible to them and can have different impact on the Cloud's operational behavior. Therefore, we segregate the criticality of services with respect to the different attacker profiles.

In contrast to the state of the art, the proposed approach is able to (a) identify anomalous sequence of operations resulting from vulnerable service interaction across the multi-level Cloud operational stack, and (b) identify anomalous state transitions based on the level of “visibility" of the states to the attackers.

Table 4.1: Description and Data Type of Places in the Cloud Model

Place	**Description**	**Data type**
Usr_int	Interface to enter user credentials (user_namexPassword).	$String \times String$
Ath_srv	Authentication Service from the CSP (stored user_name and Password).	$String \times String$
VM_req	VM requested instance (CPUxRAMxDisk).	$Int \times Int \times Int$
$Sess$	Session details of the customer (Sess_IDxSess_PolicyxSess_Exp).	$Int \times String \times String$
VM_buff	Temporarily stores VM request value and session details for verification.	$Sess \times VM\_req$
VM_conf	Holds VM data after successful session verification.	$Sess \times VM\_req$
Ad_conf	Holds quota and access policy of the customer and administrator configurations.	$String \times String \times String$
DB_ent	Initial entry of the VM in the database (VM_IDxVM_req).	$Int \times VM\_req$
$Schd$	Receives VM details to find a potential server.	$DB\_ent \times VM\_req$
$Serv\_loc$	Selection of the server for the VM (LocationxData_Center).	$String \times String$
VM_srvr	Receives server and VM details.	$Serv\_loc \times Schd$
NIC_map	Generates MAC address and mapping between virtual and physical network interface.	$String$
$DHCP$	Assigns dynamic IP to the instance.	$String$
DI	Holds Disk Image of the VM.	$Machine\ Disk\ Image$
Hyp	Receives all the configurations and launches the VM.	$DI \times DHCP \times NIC \times VM\_srvr$
Phy_HW	VM is started on the server.	Hyp

4.3 Operational Modeling of the Holistic Cloud for Threat Analysis

In this section, the Cloud operations are modeled by considering the essential services and their interaction in launching a Virtual Machine (VM). As a VM is the elemental component behind Cloud services, hence we focus on the services involved in launching a VM. We surveyed multiple open source Cloud computing environments [SAE12, NWG^{+}09, MLM11] and identified the essential services involved in the process of launching an instance of a VM in these environments.

We profile the behavioral characteristics of the services with a focus on information flow and interaction among these services. We utilize the acquired information to develop our multi-level Cloud model using High-Level Petri Nets (HLPN). The HLPN forms the basis to analyze behavioral and structural properties of the Cloud under normal operations and under varied attackers manipulation. The obtained model is shown in Figure 4.1, while the description and data type of the places are shown in Table 4.1. Furthermore, we define the rules (pre/post conditions) that govern the flow of information among these places. These rules enable the firing of transitions and place token (information) from input place(s) to the respective output place(s). Before describing the details of the model, we first overview the process of initiating a VM instance in the Cloud.

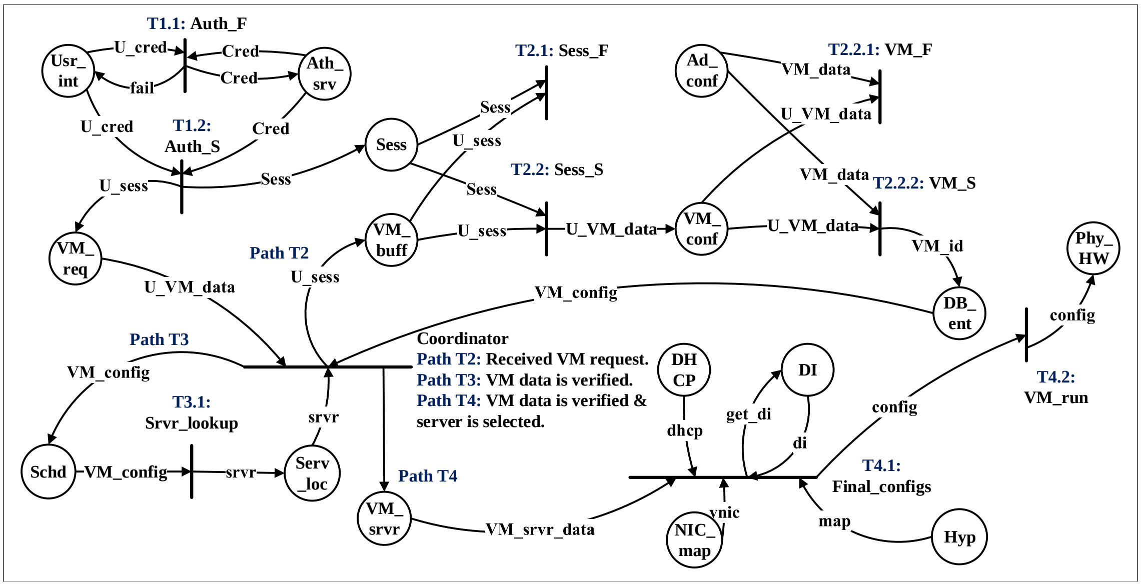


Figure 4.1: Petri Nets based Cloud Model

4.3.1 Instantiation of a VM

The process of initiating a VM begins with authenticating the customer (T1.1 in Figure 4.1- top left). The customer provides the credentials which are then validated by the CSP. Following a successful authentication, the customer provides the desired set of properties for the VM (e.g., CPU, RAM, disk space). If the session is active, and the VM properties comply with quota and access policy of the customer, then the coordinator service invokes the scheduler service to determine a potential data center and a corresponding physical machine for the requested VM. The coordinator service is responsible for managing this information and invoking the relevant services based on the fulfillment of the precondition(s). Once the scheduler selects and provides information about the data center and the server, the coordinator service next invokes multiple services to, assign disk image, initialize a virtual network interface card and assign MAC/IP addresses. These configurations are pushed onto the hypervisor which accordingly configures the VM accordingly and initiates the VM on the selected server.

This is a simplistic overview of services involved in the VM instantiation process. In the following section, we describe the details of services interaction and model their behavioral characteristics by defining rules that govern the flow of information to launch an instance of the VM in a Cloud.

4.3.2 Information Flow in the Cloud

The process of initiating a VM follows a characteristic set of services that interact with each other to successfully launch a VM. In this section, we define the rules that describe the information flow and processing of the information in the services to trigger the relevant transitions shown in Figure 4.1. The set of transitions are $T = \{Auth\_F, Auth\_S, Coordinator, Sess\_F, Sess\_S, VM\_F, VM\_S, Server\_lookup, Final\_config, VM\_run\}$. The prefix $Tx.y$ in the figure is used to identify the traversal, where, x indicates the path to which the transition belongs and y indicates the sequence of the transition in that path.

A new token is generated each time the customer tries to log in through the interface of the Cloud and therefore, initiates the execution of transitions in the path $T1$. The transitions $Auth\_F$ and $Auth\_S$ determine whether the customer's credentials are valid or invalid. These transitions are mapped according to the rules shown in equations 4.1 and 4.2.

$$\begin{aligned} R(Auth\_F) = \forall cred \in Cred : U\_cred[1] \neq cred[1] \\ \vee U\_cred[2] \neq cred[2] \vee U\_cred \neq cred \end{aligned} \tag{4.1}$$

$$\begin{aligned} R(Auth\_S) = \forall cred \in Cred : U\_cred[1] = cred[1] \\ \wedge U\_cred[2] \ = cred[2] \wedge U\_cred = cred \end{aligned} \tag{4.2}$$

The equation 4.1 depicts a mismatch in the credentials provided by the customer and stored at the provider, and therefore the customer is again required to enter correct credentials. The transition $Auth\_S$ is fired after the customer provides valid credentials and hence, a session is initiated and access privileges are granted to the customer based on the access policy and quota of the customer.

The next place VM_req allows the customer to enter desired VM data such as CPU, memory, and storage. These values are passed to the coordinator whose primary job is to manage and share respective information with the next place that satisfies the precondition(s). The conditions for next places and their description are shown in Table 4.2.

Since the condition for path $T2$ is satisfied, the next place VM_buffer receives and stores VM and session details temporarily for the next transitions to validate the session and verify the VM properties. The rules for session validation is shown in equations 4.3 and 4.4.

$$\begin{aligned} R(Sess\_F) = \forall sess \in Sess : U\_sess[1] \neq sess[1] \\ \vee U\_sess[2] \neq sess[2] \vee U\_sess[3] > sess[3] \end{aligned} \tag{4.3}$$

$$\begin{aligned} R(Sess\_S) = \forall sess \in Sess : U\_sess[1] = sess[1] \\ \wedge U\_sess[2] = sess[2] \wedge U\_sess[3] < sess[3] \end{aligned} \tag{4.4}$$

The transition 4.3 is fired (a) if the session is expired or (b) if the customer's access policy contradicts with the provider's and therefore, no further transitions are fired. The equation 4.4 successfully verifies the session identity, expiration time and policy of the session. This leads to the firing of next transitions to verify the VM requested data with the quota of the customer using rules 4.5 and 4.6.

Table 4.2: Condition for Selecting Next Place from Coordinator

Next Path/Place	Condition	Description
$T2/VM\_buff$	Transition $T1.2$ is already fired.	VM data is available.
$T3/Schd$	Transitions $T2.2$ and $T2.2.1$ are already fired.	Session is active and VM data is verified with the customer's quota.
$T4/VM\_srvr$	Transitions $T2.2$, $T2.2.1$ and $T3.1$ are already fired.	VM data is verified and the data center and server is selected.

$$\begin{aligned} R(VM\_F) = \forall VM\_data \in VM\_data : U\_VM\_data[1] \\ \notin VM\_data[1] \vee U\_VM\_data[2] \notin VM\_data[2] \\ \vee U\_VM\_data[3] \notin VM\_data[3] \end{aligned} \tag{4.5}$$

$$\begin{aligned} R(VM\_S) = \forall VM\_data \in VM\_data : U\_VM\_data[1] \\ \in VM\_data[1] \wedge U\_VM\_data[2] \in VM\_data[2] \\ \wedge U\_VM\_data[3] \in VM\_data[3] \end{aligned} \tag{4.6}$$

The equation 4.5 indicates an invalid VM request by the customer. This may happen due to insufficient privileges for the requested VM or if the configuration of the requested VM does not match with the quota of the customer. On the other hand, the rule 4.6 indicates a successful VM request and assigns a unique value for the identification (ID) of the VM request. This ID enables rest of the services to distinguish between multiple VM requests from the same customer. The combined information of the VM and the session ($VM\_ID \cup U\_VM\_data \cup U\_Sess$) is then written in the database.

Since the VM data is verified, hence the condition for path $T3$ is satisfied. Consequently, the scheduler service is invoked to find an appropriate data center location and a server where the VM can be successfully instantiated. The selection of the server enables the second condition (cf., Table 4.2) of the path $T4$. Thus, multiple services are invoked with the following information ($VM\_config \cup srvr$) to assign different configurations. The rule of $final\ configs$ transition is shown in equation 4.7

$$\begin{aligned} R(Final\_configs) = \exists im \in DI : get\_di = im, \mid im \rightarrow map[1], \\ \mid \forall mac \exists vnic : ((mac \leftrightarrow dhcp[1]) \leftrightarrow map[2]) \\ config := (VM\_config \cup srvr \cup map) \end{aligned} \tag{4.7}$$

There are multiple services involved in finalizing the configurations of the VM. The disk image service is responsible for providing a VM image from the repository. The network service initiates a virtual network interface card, assigns a MAC address and allocates a mapping between the virtual and the physical interfaces of the machine. DHCP is responsible to assign and manage IP address and a mapping between the IP and the MAC address of the virtual instance. These configurations are pushed onto the hypervisor, which then starts the VM on the physical hardware according to the received configurations.

These rules enable the flow of information in the Cloud ecosystem and describe the behavioral interactions of the services. We emphasize that the whole procedure is repeated each time a customer requests a new VM or when a VM is migrated from one physical machine to another.

Having established how the services interact with each other in the Cloud ecosystem, the next step is to validate our Cloud model and utilize the state space analysis to determine normal behaviors and to identify behavioral changes in the presence of a vulnerable service.

4.4 Validation of the Cloud Model

In this section, we validate the Cloud model and also analyze the behavioral properties of the services using CPN tools [JKW07]. These tools simulate and validate the HLPN to verify the model properties under a variety of conditions. One of the characteristic feature of the CPN tools is the ability to perform state space analysis of the given model. The analysis identifies strongly connected components, generates successors and predecessors of a particular node and/or develops a complete state transition diagram of the model under given conditions. Also, we can apply a customized query to focus on the states that satisfy the criterion in the query.

In the following sections, we demonstrate the effectiveness of CPN tools by developing a state transition schema consisting of all “normal" behaviors and state transitions of potential misbehaviors resulting from services manipulation by varied attackers profiles.

4.4.1 State Space Analysis of the Cloud Model

We begin our behavioral analysis by simulating the Cloud model and validating the sequence of transitions under the rules described in Section 4.3.2. These rules describe the benign interaction of services under normal Cloud operations. For example, equation 4.8 illustrates one of the possible correct sequence of operations. In contrast, equation 4.9 denotes a sequence that deviates from the normal behavior of the Cloud. The sequence is anomalous due to misaligned verification of the VM data before authenticating the customer. Moreover, the network interface and MAC address are mapped to the wrong image of the VM resulting in highlighting duplicitous behavior.

$$\begin{gathered} R(crct\_op) = \exists U\_cred \in cred, \mid VM\_data \in Ad\_conf, \mid \\ im \in DI, \mid ((mac \leftrightarrow dhcp) \leftrightarrow im) \end{gathered} \tag{4.8}$$

$$\begin{gathered} R(wrng\_op) = VM\_data \in Ad\_conf, \mid \nexists U\_cred \in cred, \mid \\ im \notin DI, \mid ((mac \leftrightarrow dhcp) \leftrightarrow im) \end{gathered} \tag{4.9}$$

In order to demonstrate the comprehensive “all" sequences of operations, we utilize the state space analysis of the CPN tools. The analysis results in a huge number of states and their interconnections, making illustrating the complete state diagram to be impractical. Therefore, we show a partial state diagram of the model in Figure 4.2[1] and analyze the structural and behavioral properties in this partial state diagram.

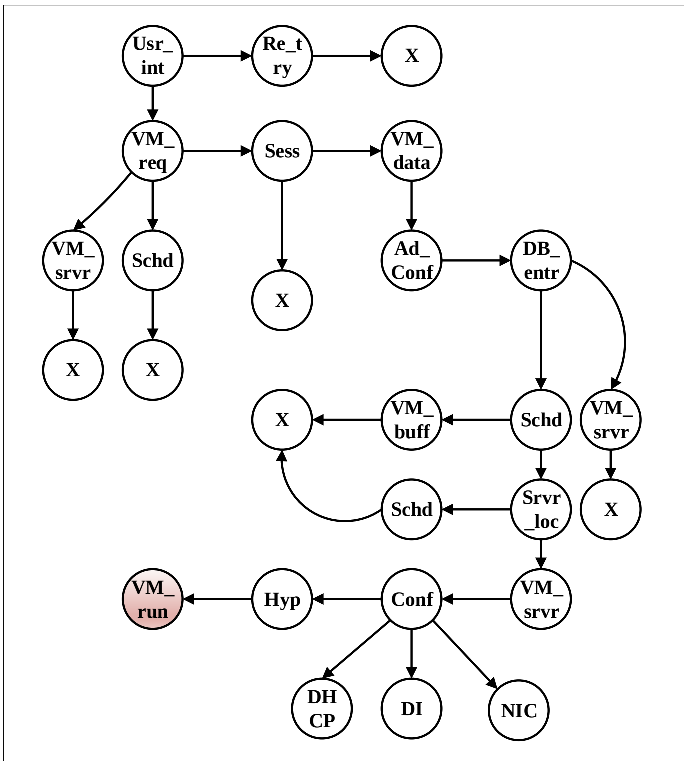


Figure 4.2: Services behavior in the Cloud IaaS

Figure 3 shows state transitions that are trivial to the normal Cloud operations, and also the state transitions that are invalid and hence should not lead to a successful instantiation of a VM. As an example of the nefarious behavior for the state *DB_entr* would be to interact directly with *VM_srvr* state and this interaction should not lead to further transitions. This prevents the state *DB_entr* to pass wrong information (e.g., server details and VM configurations) directly to *VM_srvr* under normal interactions.

However, in presence of a vulnerability or a misconfiguration in the service, certain behaviors may arise that are invalid but could lead to instantiation of the VM. Therefore, the assessment of these resulting malicious behaviors should be analyzed in the Cloud ecosystem. To investigate this, we classify states according to their accessibility to the attacker and partition state space accordingly. For example, services such as scheduler and configurations are only accessible from inside the Cloud (insider attacker), while, services such as authentication and session details are visible from outside the Cloud (outsider attacker). Thus, we analyze the malignant behaviors of a service according to the class of the attacker. This inherently reduces the number of states involved and the analysis is focused on the relevant states with respect to the attacker profile.

[1] The CPN tools assign numeric numbers to the nodes in state space which makes the analysis and diagram difficult to comprehend. Thus, we map these numeric numbers to the respective state description of the IaaS model to make it understandable for the reader.

4.4.2 Insider Attacker

An insider attack is orchestrated by people that are responsible for managing the Cloud operations. They usually have elevated access to the services and also possess intimate knowledge of the infrastructure. This makes an insider attack to have a high potential of damage, and therefore, more appealing to analyze possible misbehaviors from an insider perspective.

We analyze behavioral interactions of multiple services that are only accessible to an insider[2]. We begin our investigation with Ad_Conf service which is responsible for holding administration configurations and access policy of the customer. The equation 4.10 shows the interaction of misconfigured Ad_Conf with other services, while Figure 4.3 graphically depicts the interactions.

$$\begin{aligned} R(wr\_ad) = \exists U\_cred \in cred, \mid Ad\_conf[1] = \\ VM\_data, \mid Ad\_conf[2] = Sess, \mid \\ Ad\_conf \rightarrow VM\_srvr, \mid VM\_srvr \rightarrow Conf \end{aligned} \tag{4.10}$$

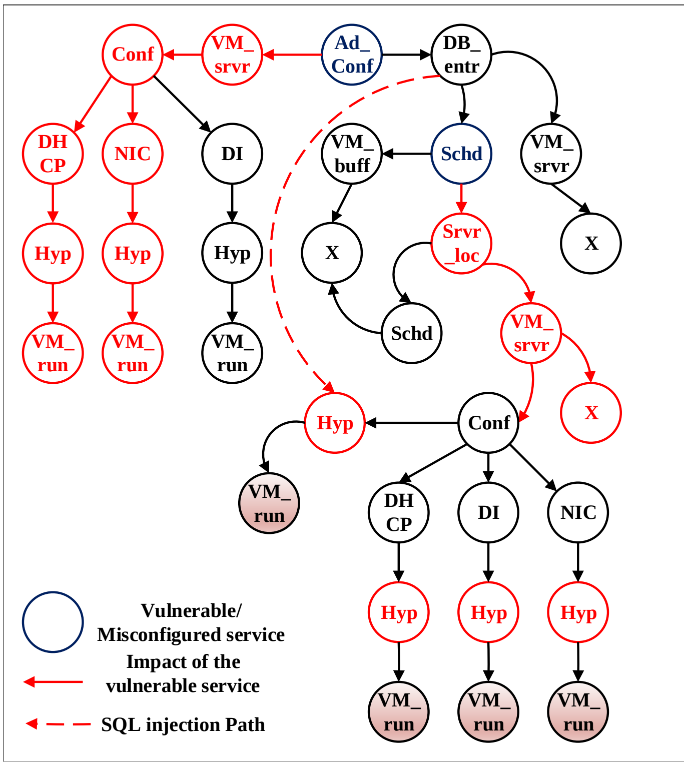


Figure 4.3: Insider attacker state space

As evident from Figure 4.3, the behavior of the transitions changes when an insider attacker misconfigures the Ad_Conf service. The misconfiguration in the Ad_Conf leads to the state VM_srvr which violates the normal sequence as the VM_srvr must always be followed by $Srvr\_loc$. The impact of this misconfiguration also affects $Conf$ which is responsible for the VM configurations (i.e., mapping between the virtual and physical network interfaces, and assigning IP/MAC address). However, Ad_Conf does not have any affect on the disk image repository which is responsible for providing the disk images for the VM.

[2] Figures 4.2 and 4.3 are extracted from the Cloud model in Figure 4.1

The second service we consider for the analysis is the *Schd* service, responsible for finding an appropriate server for the requested VM. The mis-configured *Schd* service has a limited impact as it can only change the location of the server. The implication of selecting a wrong server is severe as it may fail to instantiate the requested VM. The other impact of manipulating the *Schd* is in migrating a running VM of a client to another physical machine under the attacker's influence.

For completeness, a path in Figure 4.3 is presented for the case of an SQL injection attack in which an insider writes malignant entries directly to the database. This results in skipping a number of transitions and directly pushing the maligned server and VM configurations onto the hypervisor which will then instantiate the VM with wrong configurations on an attacker controlled server.

4.4.3 Outsider Attacker

In this section, we explore attack surfaces that can be exploited from an outsider attacker. An outsider attacker has a different (and limited) externally set of services visible to him/her versus an insider attacker. The most common outsider interface is the authentication interface that can be compromised to gain full access privileges [SHJ+11]. However, we consider a different approach and evaluate the impact of a misconfigured disk image on the Cloud operations. We analyze the behavior of the services in presence of a misconfigured disk image. The equation 4.11 depicts the behavior of the disk image in the Cloud ecosystem.

$$\begin{aligned} R(wr\_DI) = \exists U\_cred \in cred \mid VM\_data \in Ad\_conf \mid \\ DI \nrightarrow Ad\_conf \vee Ad\_conf \nrightarrow DI \\ \mid ((mac \leftrightarrow dhcp) \leftrightarrow im) \end{aligned} \tag{4.11}$$

The disk image service interacts critically with DHCP service and manages the mapping of virtual and physical network interfaces. Thus disk image can have an impact on these states and no further states are explored by the vulnerable disk image state. The behavior is shown graphically in Figure 4.4.

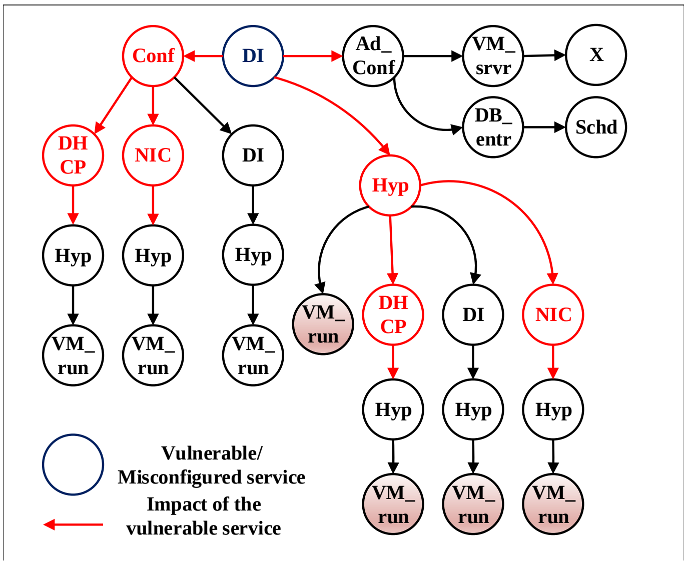


Figure 4.4: Behavioral impact of vulnerable outsider state.

However, the consequences of the disk image misconfiguration could be severe if the mapping of the networking interfaces and MAC/IP addresses are also manipulated. This could lead to network traffic being diverted to another machine.

The *DI* state interacts with Ad_conf state to evaluate the VM properties sufficiency to run the selected disk image. Thus *DI* has no vulnerable influence on the Ad_conf service. Therefore, following transitivity, the impact of the *DI* on $Conf$ is different since the interaction happens with different information and rules.

The analysis of both the insider and outsider attacker gives insights to the services that are critical according to the class of the attacker. Further analysis can be done by creating rules for a different attacker profile, e.g., we can unify the insider and outsider rules to explore attack surfaces for a collaboration attack. Conversely, we can define rules by associating different vulnerable services and analyze their impact and resulting malicious behaviors in the Cloud.

This development of a generalized Cloud model and associated multi-level modeling allows the RBTA approach to apply also for generic Cloud scenarios beyond the ESCUDO-CLOUD Use Cases.

4.5 Summary

The work of D2.4 was reported at M27 and resulted in the publications [TMT$^+$16]. The work subsequent to M27 resulted the publication [MLS17]. Overall we were able to achieve the following:

- Develop an analytical process that is able to systematically capture both direct and indirect dependencies at the level of the UC requirements. Reported in D2.4.
- Identify potential threats that could occur for the ESCUDO-CLOUD UCs, and specify the critical areas where services should be protected. Reported in D2.4.
- Explore attack surfaces in the generalized Cloud model, for applicability to generic Cloud UCs, by modeling Cloud operations using Petri nets and analyzing behavioral and structural properties of the model. Published in [MLS17].

5. Conclusion

This document presented the results and work realized in ESCUDO-CLOUD Work Package 2 from M1 to M34. WP2 addressed protection solutions for the management of outsourced data, empowering a data owner to ensure confidentiality, integrity, and availability of her data, and perform fine-grained data access. ESCUDO-CLOUD addressed this topic along four different directions in four tasks. T2.1 addressed the basic protection of data at rest; T2.2 provided solutions for key management suitable for data-at-rest encryption and other uses; T2.3 developed methods to keep the data access patterns private in a storage system private; and T2.4 provided a threat analysis based on the specified requirements. The work in these directions yielded the following results within ESCUDO-CLOUD.

Protection of data at rest. The goal of this task was to consider the basic scenario that represented the starting point for the work in the project. The goal was to focus on the support for confidentiality and integrity in a scenario where only basic upload and download operations are available over objects and where encryption is applied at the client side. Concretely the work in the task has considered a data-at-rest protection enhancement for the OpenStack Swift system, which is one of the most successful platforms for object storage in the Cloud. A significant contribution of the work in this task has been the development of the EncSwift tool.

Key-management solutions. This task has developed methods and tools for key management, mostly addressing the OpenStack Swift object storage platform. It is closely connected to the Use Case 1, which integrates data protection solutions into OpenStack. The work contributed multiple features to OpenStack and provided the basis for exploiting the results in an industrial context.

Efficient and private data access. This task extended the shuffle index approach to support dynamic data collections (i.e., a data collection that is subject to insertion, deletion, and update operations) and range queries. The task also proposed an alternative dynamically allocated data structure for data and access confidentiality, while not requiring the client to store anything. The task analyzed the protection guarantees provided by both the solutions. This task also aimed to provide solutions for integrity guarantees and, to this purpose, it proposed a novel encryption mode. This encryption mode guarantees complete mixing of the bits of the plaintext, hence even the absence of a small chunk of the ciphertext prevents the reconstruction of any portion of the plaintext.

Requirements-based threat analysis. This task established the viability of conducting a Use Case requirements based threat analysis (RBTA) to provide an input for estimating the risks based on such a threat analysis. The task also developed a systematic analytic technique that enumerates the set of UC requirements and then determines all possible direct/indirect dependencies across them to conduct a generalized threat analysis from their requirements. The approach was validated on the ESCUDO-CLOUD UCs, and has also generalized to apply to other general Cloud-based UCs external to ESCUDO-CLOUD as based on the availability of their requirements.

Bibliography

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 279–294, San Francisco, CA, USA, April 20–24, 2009. Springer, Heidelberg, Germany.

[ABM14] E. Andreeva, A. Bogdanov, and B. Mennink. Towards understanding the known-key security of block ciphers. In *Proc. of FSE*, Hong Kong, November 2014.

[ACJ17] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 445–472, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

[BCES17] Mathias Björkqvist, Christian Cachin, Felix Engelmann, and Alessandro Sorniotti. Scalable key management for distributed cloud storage. Manuscript, September 2017.

[BDF$^+$16a] E. Bacis, S. De Capitani di Vimercati, S. Foresti, D. Guttadoro, S. Paraboschi, M. Rosa, P. Samarati, and A. Saullo. Managing data sharing in OpenStack Swift with Over-Encryption. In *Proc. of WISCS*, pages 39–48, Vienna, Austria, October 2016.

[BDF$^+$16b] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Mix&Slice: Efficient access revocation in the cloud. In *Proc. of CCS*, pages 217–228, Vienna, Austria, October 2016.

[BdVF$^+$16] Enrico Bacis, S De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. Access control management for secure cloud storage. In *Proc. of SecureComm 2016*, pages 353–372, Guangzhou, China, October 2016.

[BK14] A. Biryukov and D. Khovratovich. PAEQ: Parallelizable permutation-based authenticated encryption. In *Proc. of ISC 2014*, pages 72–89, Hong Kong, China, October 2014.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BLMR15] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. Cryptology ePrint Archive, Report 2015/220, 2015. `http://eprint.iacr.org/2015/220`.

[BRS17] Enrico Bacis, Marco Rosa, and Ali Sajjad. Encswift and key management: An integrated approach in an industrial setting. In *Proc. of SPC*, Las Vegas, NV, USA, October 2017. To Appear.

[BSJ$^+$17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 619–650, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[CCFSL17] Christian Cachin, Jan Camenisch, Eduarda Freire-Stoegbuchner, and Anja Lehmann. Updatable tokenization: Formal definitions and provably secure constructions. Cryptology ePrint Archive, Report 2017/695, 2017. `http://eprint.iacr.org/2017/695`.

[CGCD$^+$17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *EuroS&P*, 2017.

[CGCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. Cryptology ePrint Archive, Report 2016/221, 2016. `http://eprint.iacr.org/2016/221`.

[CWYD10] Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10: 3rd International Conference on Cryptology in Africa*, volume 6055 of *Lecture Notes in Computer Science*, pages 316–332, Stellenbosch, South Africa, May 3–6, 2010. Springer, Heidelberg, Germany.

[DDD$^+$16] Gery Ducatel, Joshua Daniel, Theo Dimitrakos, Fadi Ali El-Moussa, Robert Rowlingson, and Ali Sajjad. Managed security service distribution model. In *Proc. of the 4th International Conference on Cloud Computing and Intelligence Systems (CCIS 2016)*, Beijing, China, August 2016.

[DFJ+07] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proc. of VLDB*, pages 123–134, Vienna, Austria, September 2007.

[DFJ+10a] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati. Encryption-based policy enforcement for cloud storage. In *Proc. of SPCC*, pages 42–51, Genova, Italy, June 2010.

[DFJ+10b] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM TODS*, 35(2):1–46, April 2010.

[DFM+16] S. De Capitani di Vimercati, S. Foresti, R. Moretti, S. Paraboschi, G. Pelosi, and P. Samarati. A dynamic tree-based data structure for access privacy in the cloud. In *Proc. of IEEE CloudCom*, pages 391–398, Luxembourg, December 2016.

[DFP+15] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Shuffle index: Efficient and private access to outsourced data. *ACM TOS*, 11(4):19:1–19:55, October 2015.

[Dwo01] M. Dworkin. Recommendation for block cipher modes of operation, methods and techniques. Technical Report NIST Special Publication 800-38A, National Institute of Standards and Technology, 2001.

[EPRS17] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 98–129, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[FR17] S. Foresti and M. Rosa. Final tools for the protection of integrity and confidentiality of data and access. Deliverable D2.5, ESCUDO-CLOUD, March 2017.

[GM17] Felix Günther and Sogol Mazaheri. A formal treatment of multi-key channels. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 587–618, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[HRsV07] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 233–252, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[ID03] Anca Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *ISOC Network and Distributed System Security Symposium – NDSS 2003*, San Diego, CA, USA, February 5–7, 2003. The Internet Society.

[JKW07] K. Jensen, L. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9:213–254, 2007.

[LR88] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comp.*, 17(2):373–386, April 1988.

[LT17] Anja Lehmann and Björn Tackmann. Updatable encryption with adaptive corruptions and post-compromise security. Manuscript, September 2017.

[LV08a] Benoît Libert and Damien Vergnaud. Multi-use unidirectional proxy re-signatures. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08: 15th Conference on Computer and Communications Security*, pages 511–520, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press.

[LV08b] Benoît Libert and Damien Vergnaud. Tracing malicious proxies in proxy re-encryption. In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008: 2nd International Conference on Pairing-based Cryptography*, volume 5209 of *Lecture Notes in Computer Science*, pages 332–353, Egham, UK, September 1–3, 2008. Springer, Heidelberg, Germany.

[MLM11] D. Milojičić, I. Llorente, and R. Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15:11–14, 2011.

[MLS17] S. Manzoor, J. Luna, and N. Suri. Attackdive: Diving deep into the cloud ecosystem to explore attack surfaces. In *Proc. of SCC*, pages 499–502, 2017.

[MLY05] S. Myagmar, A. Lee, and W. Yurcik. Threat modeling as a basis for security requirements. *Symposium of SREIS*, pages 1–8, 2005.

[MS17] Steven Myers and Adam Shull. Efficient hybrid proxy re-encryption for practical revocation and key rotation. Cryptology ePrint Archive, Report 2017/833, 2017.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[NWG+09] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, D. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. of CCGRID*, pages 124–131, 2009.

[Ope] OpenStack Project. `http://www.openstack.org`.

[OSC06] E. Oladimeji, S. Supakkul, and L. Chung. Security threat modeling and analysis: A goal-oriented approach. In *Proc. of IASTED*, pages 13–15, 2006.

[PBD+16] S. Paraboschi, E. Bacis, S. De Capitani di Vimercati, S. Foresti, D. Guttadoro, S. Mutti, M. Rosa, and A. Saullo. Tools for protecting confidentiality and integrity of data and access. Deliverable D2.2, ESCUDO-CLOUD, June 2016.

[PBSL13] D. Perez-Botero, J. Szefer, and R. Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proc. of SCC workshop @Cloud*, pages 3–10, 2013.

[PCI16] PCI Security Standards Council. Requirements and security assessment procedures. PCI DSS v3.2, 2016.

[Rut08] J. Rutkowska. Security challenges in virtualized environments. In *Proc. of RSA*, pages 260–273, 2008.

[SAE12] O. Sefraoui, M. Aissaoui, and M. Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55:38–42, 2012.

[SHJ+11] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono. All your clouds are belong to us: security analysis of cloud management interfaces. In *Proc. of CCSW*, pages 3–14, 2011.

[SS04] F. Swiderski and W. Snyder. *Threat modeling*. Microsoft Press, 2004.

[SS13] E. Stefanov and E. Shi. ObliviStore: High performance oblivious cloud storage. In *Proc. of IEEE S&P*, pages 253–267, San Francisco, CA, May 2013.

[SvS+13] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: An extremely simple Oblivious RAM protocol. In *Proc. of CCS*, pages 299–310, Berlin, Germany, November 2013.

[TMT+16] A. Taha, P. Metzler, R. Trapero, J. Luna, and N. Suri. Identifying and utilizing dependencies across cloud security services. In *Proc. of AsiaCCS*, pages 329–340, 2016.

[VS12] S. VivinSandar and S. Shenai. Economic denial of sustainability (edos) in cloud services using http and xml based ddos attacks. *International Journal of Computer Applications*, 41:11–16, 2012.

[WLOB09] W. Wang, Z. Li, R. Owens, and B. Bhargava. Secure and efficient access to outsourced data. In *Proc. of the 2009 ACM Workshop on Cloud Computing Security (CCSW 2009)*, Chicago, IL, USA, November 2009.