Practical Active Revocation

Stefan Contiu
Scille
France
stefan.contiu@scille.fr

Laurent Réveillère
University of Bordeaux
France
laurent.reveillere@u-bordeaux.fr

Etienne Rivière
ICTEAM, UCLouvain
Belgium
etienne.riviere@uclouvain.be

Abstract

We propose Knob, a practical active revocation scheme allowing to efficiently revoke users' access to encrypted data banks stored in public clouds. Knob leverages Trusted Execution Environments and All-or-Nothing Data Transforms in order to re-encrypt only small portions of the content directly in the cloud, using a scalable swarm of re-encryption workers. It prevents malicious users from being able to predict which portions of the files will be re-encrypted upon a revocation, effectively disabling pre-provisioning attacks. Our evaluation using industry workloads shows that Knob outperforms active revocation using full re-encryption by up to 3 orders of magnitude while being on average 3 to 7 times faster than state-of-the-art partial re-encryption.

CCS Concepts • **Information systems → Cloud based storage**; • **Security and privacy → Information accountability and usage control**; **Access control**;

Keywords Cloud Storage, Access Control, Revocation

ACM Reference Format:
Stefan Contiu, Laurent Réveillère, and Etienne Rivière. 2020. Practical Active Revocation. In *21st International Middleware Conference (Middleware '20), December 7–11, 2020, Delft, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3423211.3425667

1 Introduction

Data vendors such as commercial satellite image or human genomic data analysis providers often cater to sheer volumes of data, typically in the Peta-Bytes range. This data is made available under a subscription model, in which clients pay for accessing datasets. Given the complexity and cost of setting up in-house storage and delivery, data vendors benefit greatly from using object storage services offered by cloud providers, with their automated scaling, pay per use model, and high availability. For example, Google Cloud Storage is used by Planet [37], a commercial satellite imagery provider storing more than 7 PB of data [46] with 7+TB of data added every day, and by Autism Speaks [2], a genome processing organization storing more than 100 PB of data [22].

Public cloud solutions have, however, a record of data breaches caused by malicious and unauthorized users, inside workforce or administrators, or exploits at the lowest level of the software stack [20, 26, 39, 42]. Data vendors are understandably reluctant to delegate access control to their data to cloud providers, which could expose business-critical information such as the list of customers identifiers and their subscribed datasets. The natural solution is to encrypt the data at the vendor side before sending it to the cloud [8], and to rely on cryptographically-enforced access control that remains managed by the vendor itself, *i.e.*, outsourcing data delivery without outsourcing control [17]. Access to data is granted for a group of users with the same credentials using group key management, where all authorized users for a given dataset are provisioned with a common key [21].

A key requirement for data vendors is the ability to support *revocation* operations, *e.g.*, when a customer's subscription ends, or if some user is suspected of breaching the terms of use for the data. This requires the generation and distribution of a new group key to all members, the revoked user having access only to the previous key(s). While *new* data is always encrypted with the new key, existing approaches to revocation differ in how they protect *existing* data from further access by revoked users. Lazy revocation requires that only data produced after the revocation be protected, while active revocation requires this also for data available at the time of the revocation.

As active revocation traditionally required prohibitive operations of *full* re-encryption of the dataset, research focused primarily on efficient lazy revocation [6, 12, 45, 49]. Yet, protecting existing data from revoked users is important: In the example scenario of a satellite images bank, the data vendor will want revoked users to lose access to all the data whenever a subscription ends or suspicious behavior is detected. Furthermore, given the sometimes infrequent update rate of some of the covered regions, lazy revocation can offer to a revoked user a large window of opportunity to continue accessing data encrypted with the previous key(s), until it eventually is replaced by newer versions.

Partial re-encryption can mitigate the cost of implementing active revocation. Files are split in fixed-size blocks, and only a subset of these blocks need to be re-encrypted upon

Middleware '20, December 7–11, 2020, Delft, Netherlands

ACM ISBN 978-1-4503-8153-6/20/12…$15.00
https://doi.org/10.1145/3423211.3425667

a revocation. A partial re-encryption scheme must guarantee that (1) revoked users accessing the blocks that have not been re-encrypted do not even get a *partial* access to files' content and (2) that the identity of the blocks that will be re-encrypted cannot be predicted, preventing attackers from implementing a pre-provisioning attack, *i.e.*, where they would preventively download such blocks in advance of their revocation. Partial re-encryption requires the use of a specific encryption scheme. To this day, *Mix&Slice* [4] is the only example of such a scheme: It guarantees that : (1) all bits of the encrypted representation of every fixed-size block are interdependent (*mixed*); (2) mixed blocks are *sliced* all together into *fragments*, such that each fragment contains a part of every block; and (3) all the *fragments* are necessary to be able to decrypt the file, such that the re-encryption of only one fragment is sufficient to prevent access to revoked users. Yet, *Mix&Slice* also suffers from a number of limitations that can make its deployment and use cumbersome in practice: (1) re-encrypting a single fragment per file yields a level of security that is less than 128-bit AES; (2) fixed-size blocks require padding to a number of Bytes that is a power of 2 or 4, potentially leading to a high internal fragmentation; (3) the decryption at the client side requires a logarithmic number of linear passes and needs the entire file to be stored in memory, which may be problematic for large files and/or light clients.

Contributions. We are interested in this work in proposing an *efficient* and *practical* support for active revocation. We make two key contributions in this direction:

1. We propose Knob, a novel partial re-encryption scheme that allows re-encrypting only a fraction of a file to enable active revocation. In contrast with *Mix&Slice*, decryption operations in Knob take a linear time in the length of the file, and encryption does not require padding beyond the use of an integer number of blocks.
2. We propose an implementation of Knob that runs entirely in the cloud and does not require data owners to download and re-encrypt blocks locally. Instead, we use a horizontally-scalable set of servers supporting the Intel SGX Trusted Execution Environment, to efficiently re-encrypt blocks directly in the public cloud.

Knob offers better performance compared to *Mix&Slice* by considering slightly different assumptions on the adversary power, that we believe align with practical deployment settings. We consider, specifically, that a user who previously accessed a file in its entirety can cache a *copy* of that file's cleartext and continue accessing it past their revocation, without requiring to re-download the encrypted copy to decrypt it against cached cryptographic material.

Files stored by Knob are split in a number of fixed-size blocks, typically ranging from few hundreds of KB to a few MB. Only a small subset of these blocks, called *super blocks*, needs to be transferred to and from SGX enclaves for re-encryption, limiting processing costs, and the duration of the revocation process. Super blocks are randomly chosen among the output of an All-or-Nothing Transform [41] applied to the input file, and super encrypted such that only members provided with the corresponding group key can decipher them. Knob protects against pre-provisioning attacks by ensuring that the index of super blocks for a file can only be revealed to group members after downloading *all* of the blocks for this file. This protection is based on the use of a second AONT transform. Knob can leverage and orchestrate a large number of SGX-enabled re-encryption workers, offering horizontal scalability. The security analysis of Knob shows that the power of an attacker employing a pre-provisioning attack can never be more than the full download of the files during its tenure in that group, making the attack irrelevant in practice.

Our evaluation of Knob indicates that it can perform active revocation on 43 TB worth of satellite images or 132 TB worth of genomic data in just 11 minutes using only 5 modest servers, compared to the several days that would be required for a full re-encryption. With fragments of 256 KBytes, *Mix&Slice* would require the client to download, re-encrypt and re-upload 24 GB worth of data. Moreover, Knob is computationally faster than *Mix&Slice*, being on average 7.6 times faster for write operations and 3.5 faster for reads.

The rest of this paper is structured as follows. We start by reviewing related work in Section 2. Section 3 presents our system and threats models. Section 4 details the fundamental concepts and mechanisms used in the design and implementation of Knob, which are detailed in Section 5. Section 6 analyzes the security of Knob, and Section 7 discusses implementation details. Section 8 presents our evaluation results, including our comparison to *Mix&Slice*. Section 9 finally concludes the paper.

2 Related Work

We position our work with respect to partial re-encryption, specifically *Mix&Slice*, and the use of All-or-Nothing Transformation and TEEs for storage in untrusted clouds.

Partial Re-encryption *Mix&Slice* [4] is the only other example of a partial re-encryption scheme for active revocation. It has been integrated with OpenStack Swift [3, 5].

The basic idea of *Mix&Slice* is to provide an encrypted representation of each file that guarantees complete interdependence among its bits. The file is first partitioned into equally sized chunks called macro-blocks. An All-or-nothing transform (AONT) [41] called *mixing* is performed on each macro-block to construct a strongly interdependent ciphertext. Every individual bit in the input eventually has an impact on each of the bits in the encrypted output of the macro-block. The mixing phase requires several encryption rounds. During each round, the entire macro-block content

is visited by a block cipher, which encrypts in the same block cipher size (*e.g.*, 128 bits) small pieces of data called *mini-blocks* of 32 or 64 bits, and located in different places of the macro-block. The required number of rounds is logarithmic in the size of the macro-block. Macro-blocks are padded to have a size which is a power of 4 or 2, for mini-blocks of 32 or 64 bits respectively. In order to revoke access to a macro-block, it is sufficient to super-encrypt a single mini-block.

To touch all the macro-blocks constituting a resource at once, *Mix&Slice* introduces the concept of *fragments*. Mixed macro-blocks are *sliced* into fragments so that fragments provide complete coverage of the resource content. Each fragment represents a minimal unit of revocation. Fragments can be visualized as vertical cuts when the macro-blocks of the resource are stacked one on top of each other. The height of this cut is the number of macro-blocks while the width is the size of a mini-block. During the revocation operation, the data owner selects a random fragment and re-encrypts it using a key unknown to the revoked users.

The fragments' representation of *Mix&Slice* leads to trade-offs between security and costs. First, brute forcing the decryption of a single mini-block in a fragment can reveal the content of a macro-block, thus a portion of the resource. As a mini-block has a size of 32 bits or 64 bits, brute forcing it falls under the security strength of AES which requires brute forcing of 128 bits. If higher security is desired, *Mix&Slice* requires re-encrypting a higher number of fragments, linearly increasing the transmission I/O during the revocation operation. Second, a client needs to download all fragments and hold them all in memory to compute the macro-block representation, due to the non-linear memory access patterns involved in the decryption. For large files exceeding client device available memory, the fragmenting technique can pose serious problems. The issue could be fixed by dividing the files in sub-files, linearly decreasing required memory by linearly increasing I/O requirements.

Our work also implements partial re-encryption for active revocation and improves upon the performance and ease of deployment of *Mix&Slice*. Note, however, that we consider a slightly different adversary model: We accept that a user having downloaded a file in its entirety while being a member of the group is able to cache information, allowing to possibly access *portions* of the file later that have not been re-encrypted. *Mix&Slice* prevents this from happening: revoked users cannot access even partial content using cryptographic material they could cache prior to revocation. Since the user could simply cache the *content* of the file itself, we consider this difference will have only a modest impact in practice.

Use of AONT in Untrusted Cloud Storage The All-or-Nothing Transform used by Knob was proposed by Rivest [41]. It guarantees that a ciphertext can be decrypted only after being received in its entirety [38]. AONT has been used in cloud storage to ensure that an adversary with only access to a subset of the data is not able to partially decrypt it.

A first scenario of use is cloud-of-clouds, or *dispersed* storage systems [8]. It allows storing splits of data at different cloud providers with the guarantee that the compromission of one will not leak usable data. Kapusta *et al.* [31] leverage the properties of the AONT transform in this context to implement revocation when one of the clouds may act maliciously; a previous proposal [30] in the same direction involves keeping one of the shares at the user side. AONT was used together with Reed Solomon (RS) error encoding to disperse file content over multiple clouds [27], creating a (k, n) threshold security scheme (AONT-LT [7] follows a similar approach but using rateless Luby transform codes). CDStore [34] extends the same blend of AONT and RS by using a deterministic value instead of the randomized AONT encryption key, permitting secure deduplication. REED [33] extends CDStore for the re-keying operation by reducing the re-encryption of an entire file to a small 64 Bytes packet. However, REED does not protect past data against users that bulk copied the small super-encrypted packets together with keys metadata while being a member of the group.

Use of TEEs Trusted Execution Environments (TEEs) are CPU extensions capable of isolating computations and data while guaranteeing the confidentiality and authenticity of computations and their results. SGX [19] is a TEE available on modern x86 CPUs manufactured by Intel. A SGX secure execution space is called an *encalve*. Enclaves can generate *proofs* attesting their code and hosting hardware as genuine, by relying on a manufacturer hosted service: the Intel Attestation Service (IAS). Enclaves can persist secrets outside of their own secure space using a sealing mechanism. For a detailed functioning of Intel SGX we defer the reader to the seminal work of *Costan et. al.* [19].

IBBE-SGX [18] leverages TEEs for optimizing the performance of access control over untrusted storage. IBBE-SGX shifts from the traditional assumptions made for identity-based broadcast encryption by relying on TEEs that store a master secret utilized instead of a costly public key encryption. The scheme makes however no assumptions on what happens to the data when revocation happens, and only focuses on revoking key material.

SecureKeeper[15] provides confidentiality guarantees to coordination data stored in Apache ZooKeeper [28]. Similarly to our solution, a first protocol phase attests and provides keys to the distributed workers set. We use ZooKeeper in Knob to orchestrate the distributed set of re-encryption workers, but we do not use it to store confidential information (*e.g.*, the identity of re-encrypted blocks).

3 Model

We start by detailing our system and threat models. The system is composed of the following actors.

Correct Users. These users are legitimate members of a group, only accessing files they actually need during their membership. They are considered fault-free.

Malicious Users. As for correct ones, malicious users fetch data from the store under the limit of their network and storage capacities, and of any bandwidth limitation that the cloud storage itself may impose. Unlike correct users who only access the data they currently need, malicious users are interested in maximizing the amount of data they can illegitimately access after a revocation (*e.g.*, after they stop paying the subscription). For this purpose, malicious users can implement a strategy to pre-provision material from the group (encryption keys, encrypted and plaintext data, *etc.*). The lower bound of the power of a malicious user using the pre-provisioning attack is given by how much plaintext files this user could download under a hypothetical and perfect active revocation scheme that would re-encrypt all the files instantaneously upon its revocation. Malicious users can manifest an individual arbitrary behavior.

Revoked Users. These users have been unauthorized for accessing data. Revoked users should be prevented from deciphering group data they have not yet downloaded, as well as any future data. We do not prevent revoked users from continuing accessing data that they have previously downloaded in full, considering that they have the power to keep a cached copy. They can manifest an arbitrary behavior but we assume that revoked users do not collude.

Untrusted Cloud Storage. We assume that the cloud object storage satisfies liveness assumptions and does not deny service. We also consider that it does not keep a history of versions of the files stored by the data vendor, but only maintains the latest version. However, the cloud object storage may be attacked by an (insider or outsider) adversary interested in learning the composition of the group or accessing actual data without a subscription. We consider, therefore, that it cannot be trusted with information in the clear or with the enforcement of access control. This is the same model as for *Mix&Slice* [4] and other storage systems for untrusted clouds [36].

Availability of Trusted Execution Environments. We use computing resources along with object storage and, similarly as for object storage, consider that these could be subject to attacks. We assume, however, that the public cloud provider offers hardware-based SGX-capable servers, as it is already the case for some big players such as Microsoft with *Azure confidential computing*. We do not require the SGX-enabled machines to be highly available as we do for the object storage: These machines are never involved on the path to serving client requests for data or for the upload of new data by vendors. There is no requirement for TEEs at the client side.

4 Building Blocks

We detail in this section the fundamental concepts and mechanisms used in the design and implementation of Knob. We start by detailing how group encryption regulates access to data, and the principles of active revocation by partial re-encryption, *i.e.*, the re-encryption of only a subset of each file. We then present the All-or-Nothing Transform, and detail how the use of Trusted Execution Environments can allow data locality for re-encryption operations.

4.1 Group Encryption

Any group of users authorized to access a given dataset is provided with a symmetric encryption key GK. The revocation of a member of the group requires the generation of a new key GK' that is distributed to the remaining group members. We consider the key distribution as a side problem and focus this paper on the design of the revocation mechanism itself. Group key establishment and distribution can employ a number of techniques such as Hybrid Encryption [21], Broadcast Encryption [9] or Attribute Based Encryption [24].

4.2 Active Revocation Using Partial Re-encryption

A revocation mechanism primarily defines what happens to the data that is already encrypted with GK when a new key GK' is generated for the group.

When using *lazy* revocation, data created after revocation is encrypted using GK', but existing data remains encrypted by GK. The scheme sets the lower bound in terms of computational costs, requiring no re-encryption. However, under our threat model, revoked users can continue to decipher data until this data is replaced by newer versions.

Active revocation requires on the other hand that all prior data be re-encrypted with the new key GK'. This approach offers the best possible security guarantees. Revoked users who keep a copy of GK are no longer able to use it to decipher content generated prior to their revocation, and only have access to the plaintext data they could obtain legitimately during their membership to the group. Active revocation schemes that adopt the re-encryption of all the files upon a revocation are highly impractical due to their high processing and data access costs. We denote this approach as *full re-encryption* in the rest of this paper. As we demonstrate in our evaluation, active revocation by full re-encryption can take up to several days for a 43 TB dataset of satellite images.

We propose instead to implement active revocation with the partial re-encryption of only a small subset of the data. Data files are split in fixed-size blocks. All these blocks are necessary in order to reconstruct the file at the client side. Only a few of the blocks, however, are re-encrypted using GK' upon a revocation. We call them *super-blocks*. Other blocks remain encrypted with the original key. In order to prevent pre-provisioning attacks, it is necessary to enforce that all blocks be necessary to reconstruct the file but also

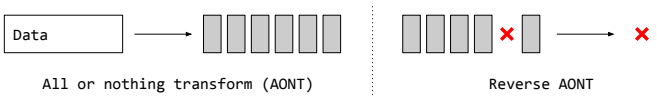


Figure 1. AONT is reversible iff all of its output is known.

to possibly determine which are the super blocks. We leverage for this the All-or-Nothing Transform detailed in the following.

4.3 All-or-Nothing Transform

Our original inspiration has been in related work in the area of Cryptographic Secure Deletion. We note that the problem of secure deletion is equivalent to revoking access to the last member of a group. A straw-man approach is to encrypt the data and then *"forget"* the encryption key [11, 40]. For example, secure deletion by randomized keys [36] encrypts every file with a random symmetric key (called File Key or *FK*), and then over-encrypt *FK* with a second key (called Group Key or *GK*). When a revocation happens, only the group-key *GK* changes but not the underlying keys *FK*. This approach allows for fast revocations but does not satisfy our requirements. Malicious users can indeed provision all the file keys and then, once revoked, retrieve the encrypted content without requiring *GK*.

An alternative is to use an All-or-Nothing Transformation (AONT) [41]. This transformation is reversible only if *all* of its resulting output is known (Fig. 1). In addition, any incomplete subset of this output reveals nothing about the input data. Rivest introduced AONT for hardening against brute force key determination attacks for block ciphers. The transformation employs a symmetric encryption coupled with the hashing of each encrypted block. All encrypted block hashes are *xor*-ed with the symmetric key, resulting in a small packet that we commonly refer to as *tail*, appended to the output of the transformation. AONT is also used for defining Optimal Asymmetric Encryption Padding (OAEP) [13] and is standardized as a padding technique for RSA [29].

AONT is used for cryptographically secure deletion [36] by overwriting only the tails of data blocks subjected to AONT, thereby preventing the reversing of the transformation. A similar mechanism of tail over-writing can be used for revoking encrypted deduplicated content [33].

Intuitively, the use of AONT can significantly lower the processing cost of an active revocation scheme by requiring to re-encrypt only the tail and not an entire file. It is however sensitive to pre-provisioning attacks, which can considerably augment the power of the attacker under our threat model. A malicious user can indeed selectively pre-provision the sensitive information that is the *tail* blocks. In a second phase, once revoked, this user could download the remaining blocks and decrypt them using the previously downloaded tail. Our goal is to build a re-encryption scheme that is robust to a curious user arbitrarily provisioning sensitive information. Interestingly, this involves the use of a second AONT for hiding the part of the file that is subject to re-encryption.

4.4 Data Locality and Trusted Execution Environments

While security is the most important aspect for active revocation, the data locality of the re-encryption operation has a strong impact on performance, which may in turn increase the duration of the revocation operation and increase the power of a malicious revoked user. A naive solution would be to download all data blocks subject to re-encryption at the data vendor side, re-encrypt them locally, and upload them again to the cloud. This approach satisfies confidentiality requirements, but it is impractical for large data sets.

Proxy re-encryption [1] allows an untrusted cloud storage to re-encrypt a piece of data using a token sent from a trusted user. This enables both confidentiality and locality. The storage acts as a proxy and learns nothing about the data and encryption keys. This fulfills both the confidentiality and locality guarantees. However, proxy re-encryption is used as a public key cryptosystem, and as such it can be used to protect an intermediary randomized key but not the actual data [21]. Considering our threat model, revoked users that pre-provisioned sensitive information could make use of it even after the proxy re-encrypts the randomized keys. Key-homomorphic PRFs [10] are constructs that can mitigate this issue and achieve both data confidentiality and locality, but they come with unpractical computational cost [21].

A different avenue is represented by the use of Trusted Execution Environments (TEEs). TEEs are processor extensions that can provide shielded code execution while guaranteeing the isolation, confidentiality, and integrity of the computations. Intel SGX [19] has been successfully used for performing trusted computations in untrusted environments [15, 44]. The basic operational level of the Intel SGX TEE is called an *enclave*. Data manipulated in the clear by the enclaves cannot be observed from outside the CPU boundary, even when observing the main memory content or bus exchanges. A sealing mechanism allows the persistence and the exchange of data outside of the enclave. Moreover, enclaves can be attested as genuine through an attestation mechanism that either involves an interaction with a remote service hosted by Intel, or the set up of a local attestation service (DCAP [43]). SGX enclaves were recently documented as being potentially subject to side channel [14, 23, 32, 48] or speculative execution [47] attacks, but we consider that these do not dismiss the concept of TEEs in general: They are likely to be addressed by evolutions of the concept's implementation, and solutions already exist to detect or protect against such attacks [16, 25, 35].

5 System Design

We are now ready to describe the design of Knob, a practical active revocation scheme leveraging All-or-Nothing Transform and Trusted Execution Environments. Knob is composed of three main components: (1) The trust establishment protocol attesting the re-encryption workers. (2) The protocol that the clients follow to read and write files. (3) The distributed re-encryption protocol leveraging SGX enclaves.

5.1 Prerequisites

Our construction uses standard cryptographic primitives, making it both easy to parse and implement. We denote by $E_k(d)$ a symmetric encryption primitive employing key k on data d. $D_k(d)$ is the corresponding decryption operation. We denote by $h(d)$ a one-way cryptographic function. We denote by $AE_{pub}(d)$ and $AD_{pri}(d)$ asymmetric encryption and decryption primitives that use a public and respectively a private key. In the same manner, $AS_{pri}(d)$ and $AV_{pub}(d)$ are asymmetric signature and verification primitives. We denote as $\oplus$ the *exclusive or* operation.

We assume the existence of past (*i.e.*, *GK*) and new (i.e *GK'*) group keys. The latter is known only to non-revoked members of the group.

5.2 Trust Establishment

The bootstrapping of the system consists of establishing trust among the agent triggering the revocation (*i.e.*, the data vendor *administrator*) and the re-encryption workers that enable it. Re-encryption workers need to certify the validity of the administrator, while the administrator must certify that the workers have SGX capabilities and are instantiated with the Knob code. Attested SGX enclaves are provided with a long term Knob private key (*i.e.*, `knob-pri-key`). This key will be subsequently used by the re-encryption protocol. Trust can be established with an arbitrary number of storage-side re-encryption workers, that we simply denote to as *workers* in the following.

Figure 2 illustrates the methodology for trust establishment. A higher trusted certifying authority (CA) is being used as a point of trust for both administrators and workers.

In a first step, the administrator authenticates its public key, signed by the CA. Upon the successful verification by using the CA's public key, workers retain the administrator public key with the scope of accepting re-encryption requests exclusively signed by this administrator key.

In a second step, each worker produces two items : (1) a digest of the code and data instantiated in the SGX enclave (commonly refereed to as a *quote* in the SGX literature) and (2) the public component of an asymmetric key generated inside the enclave. The worker signs both the digest and the enclave public key using the fused SGX key `sgx-key`.

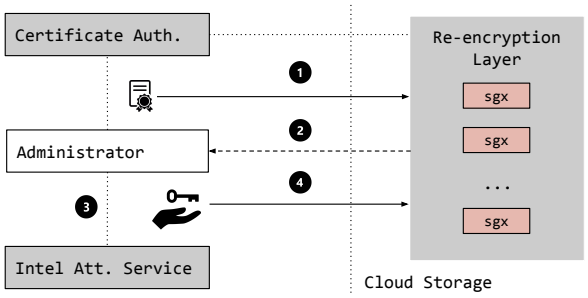


Figure 2. Trust establishment.

In the third step, the administrator contacts the Intel Attestation Service and validates that the signature of the quote is indeed a genuine SGX key.

In the final step, the administrator provides to the enclave the long term `knob-pri-key`. To do so, the administrator encrypts `knob-pri-key` using the public key sent by the enclave, and signs it using the administrative private key. Upon receiving the ciphertext, the workers decipher `knob-pri-key` after validating the administrator signature. To persist the `knob-pri-key`, enclaves use the standard sealing mechanism offered by SGX [19].

5.3 Client Write with Knob

In the following we describe the protocol that the data vendor follows to write a file to the untrusted storage, such that Knob can be used to implement active revocation operations.

The core of the solution is to employ an All-or-Nothing Transform together with the super-encryption of some of the resulted blocks by using the group key *GK*.

Let $\mathcal{D}$ represent an input data file. The write operations proceeds by splitting $\mathcal{D}$ in equal size blocks $d_0, d_1, ..., d_n$. An AONT follows, by generating a random symmetric key, that we refer to as the File Key *FK*. This key is used for encrypting each block of $\mathcal{D}$:

$$(c_0, c_1, ...c_n) = (E_{FK}(d_0), E_{FK}(d_1), ..., E_{FK}(d_n)) \quad (1)$$

The hashes of the resulted ciphertexts are then chained together with *FK* using successive exclusive-or operation, forming a tail packet, that we refer to as metadata:

$$meta_{FK} = h(c_0) \oplus h(c_1) \oplus ... \oplus h(c_n) \oplus FK \quad (2)$$

Note that possessing all ciphertexts $(c_0, ..., c_n)$ and $meta_{FK}$ allows deriving *FK*. In order to prevent the finding of the entire set of hashes unless *GK* is known, we randomly choose a subset of ciphertext blocks, and super encrypt them by using *GK*. We call these blocks the *super blocks*:

$$(se_i, ..., se_k) = (E_{GK}(c_i), ..., E_{GK}(c_k)) \quad (3)$$

Non-super blocks are using the single encrypted version obtained in Eq. 1. We push to the cloud storage the super blocks, the metadata packet, and the non-super blocks. When

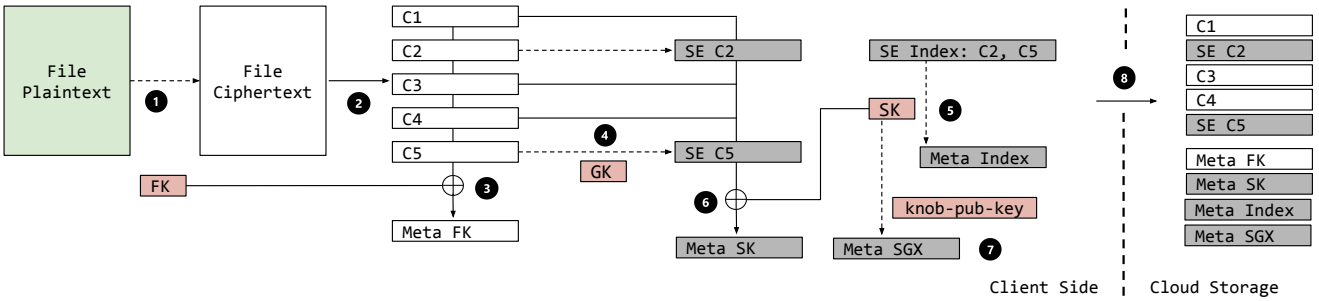


Figure 3. Knob client writing a file to the storage.

a re-keying happens, only the super blocks need to be re-encrypted from *GK* to *GK′*.

Our scheme is so far secure against malicious users who are not in the possession of *GK*. However, knowing the indices of super blocks ($i, ..., k$ in Eq. 3) prior to downloading a file allow easily implementing a pre-provisioning attack by selectively downloading a version of these blocks and use this stored version to access the file after their re-encryption, by deriving *FK*. To protect against such pre-provisioning attacks, and make sure that the power of malicious users is not greater than with full-file re-encryption, we constrain the active users to download an entire file before being able to find out which of the file's blocks actually are super blocks. We employ for this purpose a *second* AONT that allows only revealing the indices of the super blocks once the whole file has been downloaded, but not before. Let *SK* be a symmetric key, used for encrypting the indices of the super blocks:

$$meta_{index} = E_{SK}(i, ..., k) \quad (4)$$

The second AONT xor-chains *SK* together with the hashes of the blocks that were stored in the cloud:

$$meta_{SK} = h(c_0) \oplus ... \oplus h(se_i) \oplus ... \oplus h(se_k) \oplus ... \oplus h(c_n) \oplus SK \quad (5)$$

Only the super blocks are subject to re-encryption when *GK* changes to *GK′*. This means that a re-encryption worker needs to know the indices of these super blocks. Rather than requiring a worker to perform the reverse AONT, requiring a processing cost linear in the file size, we use a fact-track, constant-cost alternative. We ask that during the Knob write operation, the key *SK* that reveals the indices be encrypted by the public key associated with `knob-pri-key`. Re-encryption workers are in the possession of the corresponding decryption key, provided at the end of trust establishing protocol (Subsection 5.2). Therefore, a new metadata package is constructed:

$$meta_{sgx} = AE_{\texttt{knob-pub-key}}(SK) \quad (6)$$

Finally, the metadata packages are appended to the stored content on the cloud. Figure 3 illustrates the complete Knob write procedure with two super blocks for a file. Steps 1 to 3 correspond to the first AONT. Step 4 performs the super encryption of two blocks. Step 5 encrypts the super block index, and step 6 implements the second AONT. Step 7 asymmetrically encrypts the access to super block indices, while step 8 lists the packages that are copied to the cloud.

5.4 Client Read with Knob

We describe the protocol a client needs to follow for reading a file from the cloud storage. We consider that this client is a legitimate group member in possession of *GK*. We assume that the data vendor provides the client with the list of files and corresponding blocks when registering it to a dataset group.

Once all the blocks for a file are downloaded from the cloud, the client decrypts $meta_{SK}$ by using *GK*. Next, the client performs the reverse of the second AONT in order to find *SK*, the key that allows the decryption of the indices of super blocks. Upon knowing the indices of super blocks, the client can decipher them by using *GK*. The super blocks plaintext is used with the rest of the blocks for the reverse of the first AONT, which reveals *FK* and allows decrypting the file plaintext.

For both *read* and *write* operations, Knob similarly to *Mix&Slice* requires a number of I/O operations that is linear in the number of file blocks. However, during the decryption pass, Knob requires O(n) operations for the second AONT, while *Mix&Slice* requires O($n \log n$) operations (as detailed in Section 2).

5.5 Knob Re-encryption Protocol

Active revocation can take place as soon as re-encryption workers are validated and provisioned with `knob-pri-key`, which they can use for determining and accessing super-blocks. Re-encryption is performed in parallel over a number of workers. As the number of workers increases, the probability of a crash of one of the workers also increases. The distributed re-encryption must therefore tolerate faults and ensure the completeness of the operation, *i.e.*, that all files' super blocks have been properly re-encrypted with the new key. Knob builds on a replicated coordination kernel (Apache ZooKeeper in our implementation) to assign and monitor the completion of the distributed re-encryption tasks.

The re-encryption protocol is triggered by the data vendor administrator, who starts by splitting the list of files shared to the group into batches, forming re-encryption tasks. These tasks are orchestrated by a fault-tolerant master process built over the coordination kernel. Each task is associated with metadata, including the values of keys GK and GK' encrypted with the provisioned Knob key, only allowing the attested enclaves to access plaintext keys:

$$meta_{task} = AS_{admin-key}(AE_{\texttt{knob-pub-key}}(GK, GK')) \quad (7)$$

Next, workers receive the task metadata from the master. They check its authenticity, *i.e.*, that it has been created by an administrator, by validating its signature. They decrypt keys GK and GK' as

$$GK, GK' = AD_{\texttt{knob-pri-key}}(AV_{admin-key}(meta_{task})) \quad (8)$$

At this point the worker starts processing the task. For each file in the task, the worker fetches the index metadata ($meta_{sgx}$ and $meta_{index}$).

In order to determine the indices of the super blocks, it calculates SK (Eq. 9) and uses it to symmetrically decrypt the index (Eq. 10):

$$SK = AD_{\texttt{knob-pri-key}}(meta_{sgx}) \quad (9)$$

$$(i, ..., j) = D_{SK}(meta_{index}) \quad (10)$$

Knowing the super blocks identities, a worker fetches these blocks for processing within the enclave. It decrypts each super-block by using the old GK (Eq. 11) and encrypts it again using the new key GK' (Eq. 12):

$$(c_i, ..., c_k) = (D_{GK}(se_i), ..., D_{GK}(se_k)) \quad (11)$$

$$(se_i, ..., se_k) = (E_{GK'}(c_i), ..., E_{GK'}(c_k)) \quad (12)$$

Moreover, as the hashes of super blocks are changed, $meta_{SK}$ needs to be updated, so that the clients can perform an AONT to find SK. Therefore, $meta_{SK}$ is downloaded from the storage to the enclave, and decrypted by using GK. Old digests are hashed out from $meta_{SK}$ while new ones are hashed in, by executing Eq. 13 twice, before and after the super blocks re-encryption. Finally, $meta_{SK}$ is encrypted by GK' and pushed to the storage:

$$meta_{SK} = meta_{SK} \oplus h(se_i) \oplus ... \oplus h(se_k) \quad (13)$$

The worker finally pushes back the re-encrypted super blocks to the storage. It then signals task completion by notifying the master using the coordination kernel.

The master process monitors the progress of tasks by the workers, and can re-assign tasks of failed workers to alive ones if necessary. This may result in some duplicated work but does not bear risks of data corruption. A failure of the master is mitigated by the spawn of a new master, who recovers from the state stored in the replicated coordination kernel (this follows the classical master-worker task distribution described by Junqueira and Reed [28]).

We note that a balance of the load between workers is easier to obtain than with full re-encryption. The revocation requires re-encryption of a fixed number of super blocks per file, and the super block identifiers are available to the enclave. This results in a fixed cost per file, regardless of its size, allowing balancing the load by balancing the number of files in the assigned tasks. Differences in execution speed between workers and resulting imbalances could be addressed by implementing a form of work stealing, which we leave to future work.

Revocation operations in both Knob and *Mix&Slice* have the same constant I/O and computation complexity with respect to the number of super blocks. However, Knob leverages trusted enclaves to keep I/O local to the cloud and improve performance and practicality.

5.6 Size and Number of Super Blocks

The number of super blocks, and the fixed size of all blocks, are parameters that can be set by the data vendor.

As we detail in our security analysis in the next section, a single super block is actually sufficient if we consider the attacker's benefit expectation in terms of accessible files after revocation. A malicious user implementing a pre-provisioning attack cannot gain access to more files in expectation than a malicious user downloading, decrypting and saving complete files for later use. Using more super blocks only increases the difference between these two strategies, in favor of the latter. However, using more super blocks is a way to increase the cost of the pre-provisioning attacks, requiring more storage space per pre-provisioned file. Using more super blocks increases the cost of the re-encryption at the server side, and marginally increases the cost of the decryption at the client side. Our recommendation is that in the majority of cases, a single super block should be sufficient.

The size of the blocks is a compromise between the costs imposed at the cloud storage side (and, ultimately, on the data vendor paying for virtual machines), and the performance observed by the clients when accessing the store. A small block size reduces the cost of re-encryption on the server side, but also results in more I/O operations and calls to the cloud storage for clients, which often limits bandwidth efficiency.

6 Security Analysis

We provide in this section an analysis of the security of Knob. We consider varying number of super blocks, and focus on the fault model described in Section 3. The interest of malicious users is to get access to data past their revocation from the group. New data will always be encrypted using the new key GK', and is therefore protected. We are interested specifically in existing data that is not entirely re-encrypted with GK', due to our design choice of only re-encrypting a subset of super blocks. The maximization of accessible files depends on the success of the pre-provisioning attack,

which prepares for illegitimate access to a large number of files while minimizing download bandwidth budget.

A malicious user implementing a pre-provisioning attack will seek to maximize its gain, defined as the number of files it will have access to past its revocation. The baseline strategy is the download of entire files to local storage during the malicious user's membership period. To be of interest, the pre-provisioning attack must allow accessing more files after revocation for the same bandwidth budget.

Metadata packages are small in size (a few Bytes). We consider that these are all pre-provisioned by the malicious user. The interest of the attacker is therefore to pre-provision the super blocks prior to their re-encryption.

It is not possible to obtain the indices of the super blocks without access to all blocks, thanks to the second All-or-nothing Transform. The only possible strategy is to randomly download a subset of the blocks, in the hope that these will contain all the super blocks and allow accessing the file past revocation by downloading missing regular blocks. We analyze how much blocks of data an attacker needs to pre-provision on average in order to fetch the super blocks.

Theorem 6.1. *Let $s > 0$ be the number of super-encrypted blocks out of a total of n blocks composing a file, with $s \leq n$. Then the average number of block downloads necessary for an attacker to obtain all s super blocks is:*

$$A(n, s) = \frac{s}{s+1}(n+1) \tag{14}$$

Proof. We prove by induction. We proceed from the base case $n = 1$, therefore $s = 1$. Clearly, the attacker needs to download the *single* composing block that happens to be a super block (*i.e.*, $A(1, 1) = 1$).

We prove next the inductive step for $n + 1$, *i.e.*:

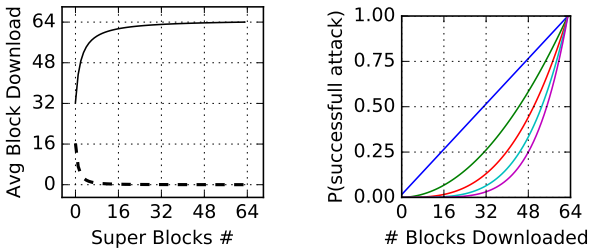
$$A(n+1, s) = \frac{s}{s+1}(n+2) \tag{15}$$

When one extra non super block is added, one falls in two outcomes: with probability $\frac{s}{n+1}$ there are $s-1$ out of n blocks left, and differently with probability $\frac{n+1-s}{n+1}$ there are left s out of n blocks. Therefore:

$$\begin{aligned} A(n+1, s) &= \frac{s}{n+1}(1 + A(n, s-1)) + \frac{n+1-s}{n+1}(1 + A(n, s)) \\ &= 1 + \frac{s}{n+1}\frac{s-1}{s}(n+1) + \frac{n+1-s}{n+1}\frac{s}{s+1}(n+1) \\ &= 1 + (s-1) + \frac{(n+1-s)}{s+1}s = s\frac{n+2}{s+1} = A(n+1, s) \end{aligned}$$

The inductive step for $s + 1$ is omitted as it is proved similarly. □

We zoom within the effect of Theorem 6.1 when using a relatively small number of super blocks. Intuitively, the values of A grow fast for small consecutive values of s (*e.g.*, 1, 2, and 3), as the coefficient of $(n + 1)$ goes down from $\frac{1}{2}$ to $\frac{2}{3}$ and to $\frac{3}{4}$. This means that when one super block is utilized



(a) The average number of blocks that an attacker has to download (—) and its rate of change (- - -).

(b) The probability that the attacker is successful given a fixed amount of download trials. Super blocks count (left→right): — 1, — 2, — 3, — 4, — 5.

Figure 4. Analysis of a 64 blocks file case for which an attacker tries to provision super blocks.

the attacker would roughly have to download half of the data, for two blocks: two thirds, for three: three fourths and so on. We illustrate this effect in Figure 4a using an example of a file composed from 64 blocks. When utilizing just a single super block the attacker needs to download in average 32.5 blocks. With just a few increases in the number of super blocks the average number of downloads increases quickly to values approaching 60 out of 64 blocks when utilizing 12 blocks. Therefore, with just a few increments in the complexity of the system to put up with, the attacker needs to download an increasingly larger number of blocks to stand a chance of owning useful data comprising all super-blocks. This effect can be concretely observed by illustrating the first derivative $A(n, s)\,dn$., a rate of change that is inverse proportional to a quadratic number of super blocks (observed with a dashed line in Figure 4a).

In conclusion, opting for a small number of super blocks can already protect against attackers that would have to inefficiently download large volumes of data. Additionally, small increases when utilizing a small number of super blocks have large implications on the number of blocks an attacker needs to pre-provision.

We generalize now our analysis by considering the total number of block downloads that the attacker is bounded to.

Theorem 6.2. *Let t be the number of block downloads that an attacker is bound to, n the total number of blocks and s the number of super blocks, where $s \leq t \leq n$. Then the probability that all s blocks were fetched by the attacker is :*

$$P(n, s, t) = \frac{t!\,(n-s)!}{n!\,(t-s)!} \tag{16}$$

Proof. Denote by $o = n - s$ the number of ordinary (*i.e.*, not super) blocks. We notice that the probability of discovering all the super blocks is equal to the probability of consecutively

discovering only ordinary blocks. Therefore:

$$P(n,s,t) = \frac{n-s}{n} \cdot \frac{n-s-1}{n-1} \cdot \ldots \cdot \frac{n-s-o+1}{n-o+1}$$
$$= \frac{(n-o)!\,(n-s)!}{n!\,(n-s-o)!} = \frac{t!\,(n-s)!}{n!\,(t-s)!}$$

□

Figure 4b shows the value of P for the same case of a file of 64 blocks. We consider up to 5 super blocks. This figure actually allows estimating the expectation of the number of accessible files after revocation, for a given fraction of blocks downloaded at random. With a single super block, downloading 50% of the blocks yields an expectation of 0.5 available files. Only with 100% of the blocks can the attacker guarantee an expectation of one available file, falling back to the arguably simpler strategy of downloading entire files in the clear prior to revocation. Adding more super blocks greatly decreases the effectiveness of this partial random download attack: the expectation of accessible files when downloading 75% of the blocks of one file is actually 0.25 file, when using five super blocks. In this case, the strategy of downloading entire files is clearly superior to a pre-provisioning attack, making the latter ineffective.

Malicious users who downloaded a file in full prior to revocation are considered to store a local copy of the file and Knob does not prevent future access to this file (Section 2). In the opposite scenario, if malicious users do not keep local copies, they can return to the cloud storage past revocation, retrieve the ciphertext and decrypt the content of the non-super blocks. To do so, they can make use of the *FK* directly, bypassing the two AONTs. However, the hypothesis of not storing local copies is impractical given the cost of storage becoming conveniently small.

7 Implementation

Our system architecture (see Figure 5) relies on Apache ZooKeeper to coordinate the distributed re-encryption and on Apache Cassandra for the cloud storage. The clients are considered to communicate directly with the storage cluster, while the administrator communicates with the ZooKeeper service.

The Cassandra (v3.11.3) ring stores both files metadata and blocks in a structured manner in two tables linked through a foreign key. The cluster is deployed over 4 physical machines. It is configured to use one replica per node and two seed nodes. The data directory of each Cassandra node points to a folder on the local SSD drive, with an available capacity of 2 TB.

Our ZooKeeper (v3.4.12) deployment relies on a single master node. A production deployment would employ several replicated nodes. The distributed master-slave orchestration uses a hierarchy of four types of ZooKeeper *znodes*. The `workers` znodes store the identifiers of workers that have an active connection to the master. The `tasks` znodes store

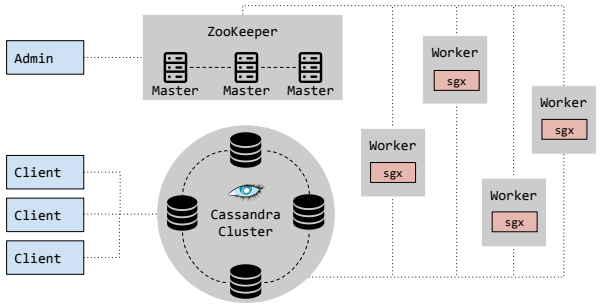


Figure 5. System Architecture.

the re-encryption batches created by the administrator. The `assign` znodes are filled by the master and contain the assignment of re-encryption batches for each worker. Finally, the `status` znodes are filled by re-encryption workers signaling the completion of a batch. The administrator watches these last znodes and decides of a revocation termination when all initial tasks batches have positive termination status. To tolerate failures, the implementation relies on ZooKeeper exceptions in combination with notifications of state changes signaling failure codes.

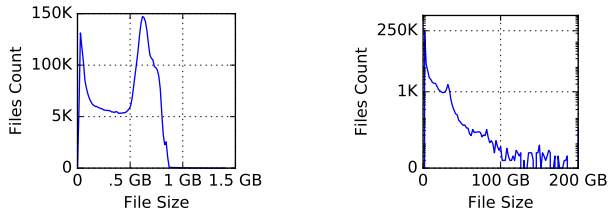
We use AES-256 for symmetric encryption, the Galois Counter Mode (GCM) authentication tag as keyed hash function for AONT making the hash key publicly known [41], and asymmetric RSA encryption with OAEP padding with keys of 4,096 bits.

All machines hosting re-encryption workers have CPUs supporting Intel SGX. The re-encryption operation utilizes a single *ecall* while four *ocalls* are utilized for fetching and writing file metadata and blocks. We leverage the `user-check` attribute for data blocks that pass the enclave border, as they are already in ciphertext form. This optimizes enclave transitioning time. We use the `sgx-ssl` library to perform cryptographic operations within an enclave, while the clients rely on `openssl` library. The workers make use of the asynchronous version of C bindings for ZooKeeper.

8 Evaluation

In this section, we evaluate the performance of Knob. We first describe the results of micro-benchmarks showing how Knob outperforms full re-encryption on a single worker node. We then illustrate the scalability of our approach on a cluster of SGX-enabled nodes using two realistic workload datasets. Finally, we compare Knob to *Mix&Slice*.

All our experiments are performed on small-form NUC servers with a 2-core (hyper-threading disabled) Intel(R) Core(TM) i7-7567U @ 3.50GHz CPU, 32 GB of RAM, and the Ubuntu 16.04.3 LTS operating system. This configuration is one of the platforms recommended by Intel for SGX development. All servers are connected via Gigabit Ethernet.



(a) ESA Satellite Images **(b)** 1,000 Genome Project

Figure 6. File sizes distribution

8.1 Datasets

We consider as data vendors the satellite imagery provider and a genome processing organization. The organizations mentioned in our introduction do not make their dataset publicly available for obvious reasons. We rely instead, for our experiments, on publicly available datasets from the same application domains.

For the satellite imagery, we use images acquired by the European Space Agency's satellite Sentinel 2[1]. The dataset contains more than four million images, ranging in size from 2 MB to 1.4 GB, with an average size of 454 MB and a median size of 522 MB (see Figure 6a). The satellite images dataset totals more than 2 PB of data.

For the genome processing organization, we use the data from the *1,000 genomes project*[2]. The dataset contains almost 500,000 files ranging in size from a few KB to up to 600 GB, with an average of 1.3 GB and a median of 19 MB (see Figure 6b). This sums up to more than 650 TB of data.

8.2 Micro-benchmarks

We first present the results of micro-benchmarks, highlighting the performance of Knob on a single re-encryption worker.

We consider the median of 10 successive executions for each operation. We do not present standard deviations as these happen to be negligible in all considered experiments. We use a single file of 522 MB, representing the median file size of the satellite images dataset (Figure 6a). We vary the block size from 256 KB to 4 MB. We vary the number of super blocks from 1 to 3.

Performance at the client side. As a client comparison baseline, we use a plain encryption scheme, in which similarly to Knob files are split into blocks, but where each block is encrypted using AES. We denote as `Crypto` the time spent in cryptographic operations (*e.g.*, symmetric and asymmetric encryption or decryption, hashes, and xor-ing operations) and as `I/O` the time spent accessing the Cassandra cluster and performing serialization and deserialization of data. We do not report operations such as memory allocation as their cumulative execution time is negligible.

Figure 7a and 7b show that the Knob execution time at the client side is dominated by `I/O` operations. As illustrated in Figure 7c, writing a file using Knob induces a small overhead compared to a simple encryption of each block. Knob performs 1.11 times slower for writes when considering a single super block of 256 KB. Similarly, Figure 7d shows the performance degradation of Knob when reading a file. Compared to a simple decryption of each block, Knob introduces a slowdown of 1.18 when using a single super block of 256 KB. The performance penalty of Knob at the client side comes mostly from the use of hashing and xor operations performed for the All-or-Nothing Transformations. There is, however, only a negligible impact on memory and bandwidth usage for retrieving the file from the cloud storage, as only a small metadata element is fetched in addition to the file's blocks.

Performance at the storage side. As a service provider comparison baseline, we compare Knob against full re-encryption, in which all blocks of the file are re-encrypted. For fairness, we perform the full re-encryption within an SGX enclave. Figure 8a presents the latency of the re-encryption operation on a logarithmic scale, due to the large difference between cryptographic operations (executed within the SGX enclave) and the I/O operations. One can notice that for each block size, the time spent in the SGX enclave increases linearly with the number of super blocks (with increments on the 10^{-3} scale). One can also notice a relative independence of the cryptographic SGX time from the block size. On the contrary, I/O time shows a linear dependence on both block size and the number of super blocks.

Figure 8b shows the performance improvement of Knob compared to a full re-encryption. Our scheme is 100 times faster when considering a single super block of 4 MB and up to 1,263 times faster with blocks of 256 KB.

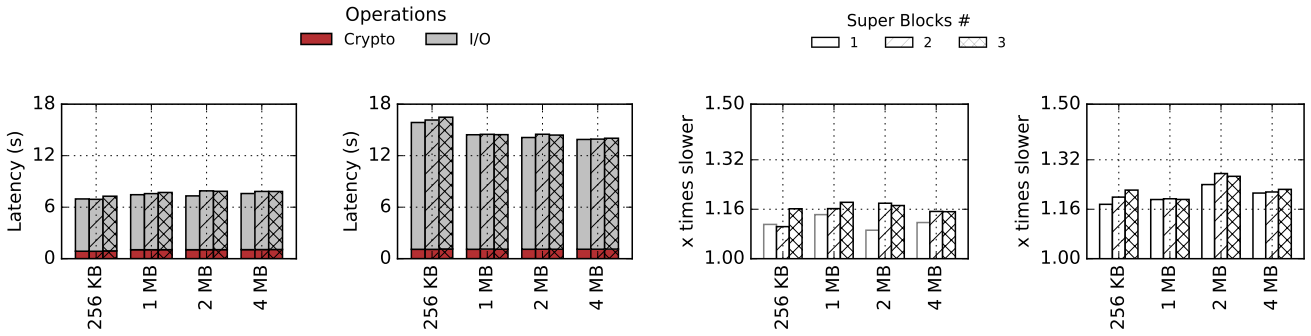
Revocation throughput. An important property of Knob is that the revocation operation is independent of the size of the file. Figure 9 shows the number of files that can be re-keyed per second using Knob. The throughput ranges from 3.3 files per seconds when using 3 super blocks of 4 MB to 40 files per second when using a single super block of 256 KB.

8.3 Macrobenchmarks

In this second part we focus on distributed experiments with representative workloads. We use 11 physical machines: one for the admin operations, 4 for the Cassandra cluster, one for the ZooKeeper master node and the remaining 5 for the re-encryption workers.

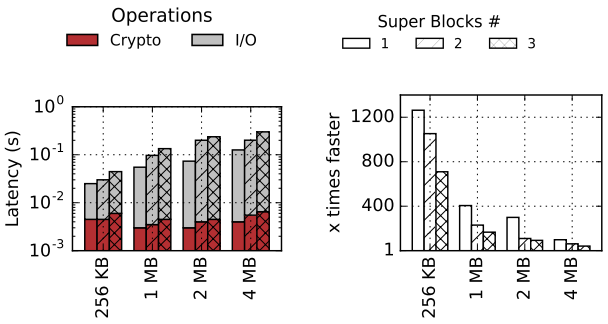
We first benchmark the scale-out capabilities of Knob when increasing the number of worker nodes. We re-encrypt a sample of 4.3 TB of the satellite data set. This corresponds to a total of 10,000 files. We use a block size of 256 KB, and a single super block per file. As shown in Figure 10, the total

[1] https://cloud.google.com/storage/docs/public-datasets/sentinel-2
[2] https://www.internationalgenome.org/



(a) Knob write **(b)** Knob read **(c)** Knob write vs. plain encryption **(d)** Knob read vs. plain decryption

Figure 7. Performance of Knob vs Encrypt schemes at the client side.



(a) Knob re-encryption **(b)** Knob vs. full re-encryption

Figure 8. Performance of Knob vs Encrypt schemes at storage provider side.

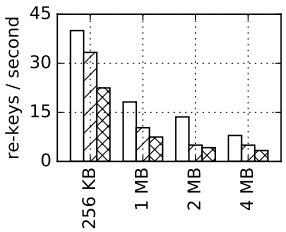


Figure 9. Knob revocation throughput

time for the re-encryption of the data set decreases from 231 seconds for one single worker to 63 seconds when using 12 workers.

We now report the total time for an active revocation when varying the number of files from 100 to 100,000. For both use cases, we generate sample (random) files that mimic the file size distribution of the corresponding datasets, as

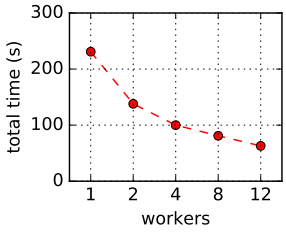


Figure 10. Knob re-encryption scale-out

described in Figure 6[3]. For Knob, we use blocks of 256 KB with one single super block and measure the total time for completing the revocation procedure. For full re-encryption, we use blocks of 256 KB and compute the total time by multiplying the time for re-encrypting one single block by the total number of blocks divided by the number of workers. This corresponds to an estimation since we do not take into account the overhead of I/O and coordination service, but it is sufficient to estimate the orders of magnitude. Figure 11 shows that Knob enables active revocation of 1,000 files in less than one minute whereas it would take hours using full re-encryption. When considering a large volume of data, Knob makes active revocation practicable in a few minutes while it would take several days otherwise.

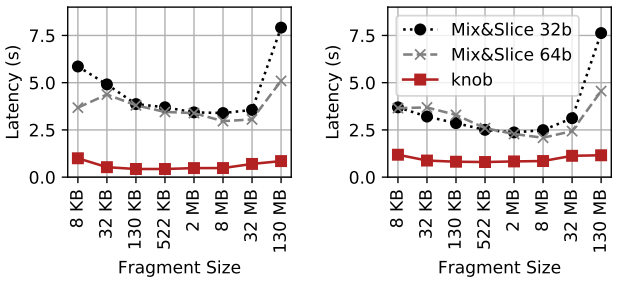
8.4 Comparison with *Mix&Slice*

The difference in computational cost between Knob and *Mix&Slice* can be visualized in Figure 12. We focus on the computational time and choose to omit cloud I/O latency, as the cost of transport of data to (write, Figure 12a) and from read, Figure 12b) the cloud is the same for the two schemes. The tests use a single file size of 522 MB (as in § 8.2), and

[3]We do not need to use the actual data and impose costs for downloading it to its providers, as cryptographic operations have a cost that is independent of the actual content. Only the file size matters in our case.

Files Count	Knob	ESA Sentinel 2 Size	ESA Sentinel 2 full re-encryption	Genomic Data Size	Genomic Data full re-encryption
1,000	7.5 s	450 GB	42.2 m	1.3 TB	2.2 h
10,000	59 s	4.3 TB	6.7 h	13 TB	19.2 h
100,000	11.1 m	43.3 TB	2.8 d	132 TB	8.6 d

Figure 11. Results of running Knob on real workloads, using a single 256 KB super block with 12 workers



(a) Write **(b)** Read

Figure 12. Knob vs. *Mix&Slice* latency (without I/O time).

vary *Mix&Slice* macro-blocks from 32 B to 512 KB. This produces, after slicing, fragments (displayed on the abscissa) ranging from 8.1 KB to 130.5 MB in size. One should notice that the larger the macro-block is, the smaller the fragment. *Mix&Slice* is benchmarked with *mini-blocks* of 32 and 64 bits. Knob was tested with blocks of the same size as *Mix&Slice* fragments.

For the write operation (Figure 12a), Knob is on average 7.6 times faster than *Mix&Slice* on 32-bit, and as much as 9.3 times faster for the largest tested fragment size of 130 MB. Compared to the 64-bit version, Knob write operation is on average 6.5 times faster. Knob latency is generally under one second, the highest value of 0.9 s is obtained for the lowest fragment size of 8 KB. For the read operation (Figure 12b), Knob is on average 3.5 and 3.2 times faster than *Mix&Slice* on 32 and 64 bits respectively. Knob has a reading time of less than a second for most fragment sizes (from 32 KB to 8 MB) incurring the highest latency for the smallest fragment size of 8 KB. For both read and write operations, Knob shows a rather constant performance, independent of fragment size. Differently, *Mix&Slice* shows a decreasing trend, handling faster increasing sizes of fragments, explained by the smaller size of macro-blocks and thus fewer rounds of encryption, while the increased performance of the largest fragment size (130 MB) is caused by the macro-block size becoming smaller than the cipher block size. When cumulating the I/O time to and from the remote storage (Fig. 7a and 7b) to the computational time (Fig. 12a and 12b), Knob outperforms *Mix&Slice* by at least 45% for writes and 10% for reads. The minimum difference is observed for a 2 MB block.

Finally, *Mix&Slice* requires the entirety of fragments to be loaded in memory to get the content of a single macro-block. Differently, Knob uses a linear block processing, a mechanism that requires only a single block in memory at a time.

9 Conclusion

We introduced Knob, a practical and efficient approach to active revocation. Knob targets large-scale data sets stored in untrusted clouds, where access is regulated by the provision of group keys under a subscription model. Active revocation prevents further access to both new and existing data by revoked users. Knob offers active replication at a fraction of the cost of full re-encryption. It splits files in fixed-size blocks and only needs to re-encrypt a fixed small subset of these (typically just one), called super blocks, upon a revocation. The use of All-or-Nothing Transforms effectively prevents pre-provisioning attacks. Knob supports partial re-encryption operations directly in the cloud, leveraging the availability of Trusted Execution Environments. Its performance greatly improves over *Mix&Slice*, a state-of-the-art system using partial re-encryption, also significantly reducing the number of cryptographic operations to perform at the clients. Our evaluation results show that Knob enables practical active revocation on representative industrial workloads: It can prevent further access to a dataset of 132 TB in only 11 minutes using 5 modest servers, compared to the 8.6 days required for full re-encryption, while providing the same security guarantees.

Acknowledgments

This research work was financially supported by *Bpifrance* under grant number DOS0092102/03. The authors extend thanks to Enrico Bacis for his kind support on *Mix&Slice* [4], to the Middleware anonymous reviewers, and to Kaveh Razavi for shepherding our paper.

References

[1] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. 2006. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)* 9, 1 (2006), 1–30.

[2] AutismSpeaks. [n. d.]. https://www.autismspeaks.org/

[3] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Daniele Guttadoro, Stefano Paraboschi, Marco Rosa, Pierangela Samarati, and Alessandro Saullo. 2016. Managing data sharing in OpenStack swift with over-encryption. In *ACM Workshop on Information Sharing and Collaborative Security (WISCS)*.

[4] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. 2016. Mix&Slice: Efficient access revocation in the cloud. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[5] Enrico Bacis, Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Marco Rosa, and Pierangela Samarati. 2016. Access control management for secure cloud storage. In *International Conference on Security and Privacy in Communication Systems (SecureComm)*. Springer.

[6] Michael Backes, Christian Cachin, and Alina Oprea. 2005. Lazy revocation in cryptographic file systems. In *Third IEEE International Security in Storage Workshop (SISW)*. IEEE.

[7] Marco Baldi, Nicola Maturo, Eugenio Montali, and Franco Chiaraluce. 2014. AONT-LT: A data protection scheme for cloud and cooperative storage systems. In *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE.

[8] Alysson Neves Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Ferreira Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. 2014. SCFS: A Shared Cloud-backed File System.. In *USENIX Annual Technical Conference*. 169–180.

[9] Dan Boneh, Craig Gentry, and Brent Waters. 2005. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Annual International Cryptology Conference*. Springer, 258–275.

[10] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. 2013. Key homomorphic PRFs and their applications. In *Advances in Cryptology–CRYPTO*. Springer, 410–428.

[11] Dan Boneh and Richard J Lipton. 1996. A Revocable Backup System.. In *USENIX Security Symposium*.

[12] Mariem Bouchaala, Cherif Ghazel, and Leila Azouz Saidane. 2019. Dual Revocation: Attribute and User Revocation Based On CPABE In Cloud Computing. In *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE.

[13] Victor Boyko. 1999. On the security properties of OAEP as an all-or-nothing transform. In *Annual International Cryptology Conference*. Springer, 503–518.

[14] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOTS)*.

[15] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. 2016. SecureKeeper: confidential ZooKeeper using Intel SGX. In *Proceedings of the 17th International Middleware Conference*. ACM.

[16] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *ACM Asia Conference on Computer and Communications Security (AsiaCrypt)*. ACM.

[17] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. 2009. Controlling data in the cloud: outsourcing computation without outsourcing control. In *ACM workshop on Cloud computing security (CCSW)*.

[18] Stefan Contiu, Rafael Pires, Sébastien Vaucher, Marcelo Pasin, Pascal Felber, and Laurent Réveillère. 2018. IBBE-SGX: Cryptographic Group Access Control using Trusted Execution Environments. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE.

[19] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[20] Adrian J. Duncan, Sadie Creese, and Michael Goldsmith. 2012. Insider Attacks in Cloud Computing. In *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*.

[21] William C Garrison, Adam Shull, Steven Myers, and Adam J Lee. 2016. On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In *IEEE Symposium on Security and Privacy (SP)*. IEEE.

[22] Google Genomics. 2018. *Genomic Data is Going Google.* Technical Report. 8 pages. https://cloud.google.com/genomics/resources/google-genomics-whitepaper.pdf.

[23] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *10th European Workshop on Systems Security (EuroSec)*. ACM.

[24] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security (CCS)*. Acm, 89–98.

[25] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th USENIX Security Symposium*.

[26] Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina, and Eduardo B Fernandez. 2013. An analysis of security issues for cloud computing. *Journal of internet services and applications* 4, 1 (2013), 5.

[27] James S Plank Jason Resch. 2011. AONT-RS: Blending Security and Performance in Dispersed Storage Systems. In *9th USENIX Conference on File and Storage Technologies (FAST)*.

[28] Flavio Junqueira and Benjamin Reed. 2013. *ZooKeeper: Distributed Process Coordination* (1st ed.). O'Reilly Media, Inc.

[29] B Kaliski and J Staddon. 1998. RFC2437: PKCS# 1: RSA Encryption.

[30] Katarzyna Kapusta and Gerard Memmi. 2018. Selective All-Or-Nothing Transform: Protecting Outsourced Data Against Key Exposure. In *10th International Symposium on Cyberspace Safety and Security (CSS)*. Springer.

[31] Katarzyna Kapusta, Han Qiu, and Gerard Memmi. 2019. Secure Data Sharing with Fast Access Revocation through Untrusted Clouds. In *10th IFIP International Conference on New Technologies, Mobility and Security (NTMS) (NTMS)*. IEEE, 1–5.

[32] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium, USENIX Security*. 16–18.

[33] Jingwei Li, Chuan Qin, Patrick PC Lee, and Jin Li. 2016. Rekeying for encrypted deduplication storage. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 618–629.

[34] Mingqiang Li, Chuan Qin, Jingwei Li, and Patrick PC Lee. 2016. CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. *IEEE Internet Computing* 20, 3 (2016), 45–53.

[35] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX enclaves from practical side-channel attacks. In *2018 USENIX Annual Technical Conference (ATC)*.

[36] Zachary NJ Peterson, Randal C Burns, Joseph Herring, Adam Stubblefield, and Aviel D Rubin. 2005. Secure Deletion for a Versioning File System.. In *4th USENIX Conference on File and Storage Technologies (FAST)*.

[37] Planet. [n. d.]. https://www.planet.com/

[38] Han Qiu, Katarzyna Kapusta, Zhihui Lu, Meikang Qiu, and Gerard Memmi. 2019. All-or-nothing data protection for ubiquitous communication: challenges and perspectives. *Information Sciences* 502 (2019).

[39] Noëlle Rakotondravony, Benjamin Taubmann, Waseem Mandarawi, Eva Weishäupl, Peng Xu, Bojan Kolosnjaji, Mykolai Protsenko, Hermann De Meer, and Hans P Reiser. 2017. Classifying malware attacks in IaaS cloud environments. *Journal of Cloud Computing* 6, 1 (2017),

26.

[40] Joel Reardon, Srdjan Capkun, and David Basin. 2012. Data node encrypted file system: Efficient secure deletion for flash memory. In *21st USENIX Security Symposium*. USENIX Association, 17–17.

[41] Ronald L Rivest. 1997. All-or-nothing encryption and the package transform. In *International Workshop on Fast Software Encryption*. Springer, 210–218.

[42] Francisco Rocha and Miguel Correia. 2011. Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *Workshops of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks (DSNW)*.

[43] Vinnie Scarlata, Simon Johnson, James Beaney, and Piotr Zmijewski. 2018. *Supporting third party attestation for Intel® SGX with Intel® data center attestation primitives*. Technical Report. Intel Corporation.

[44] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *IEEE Symposium on Security and Privacy (SP)*.

[45] Hui Tian, Fulin Nan, Hong Jiang, Chin-Chen Chang, Jianting Ning, and Yongfeng Huang. 2019. Public auditing for shared cloud data with efficient and secure group management. *Information Sciences* 472 (2019).

[46] Troy Toman. 2017. Our Data From Space Lives in Google Cloud. https://www.planet.com/pulse/planets-data-from-space-lives-in-google-cloud/.

[47] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association.

[48] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM.

[49] Saman Zarandioon, Danfeng Daphne Yao, and Vinod Ganapathy. 2011. K2C: Cryptographic cloud storage with lazy revocation and anonymous access. In *International Conference on Security and Privacy in Communication Systems*. Springer.