

ALASCA: Function-Driven Advanced Access Control for Big Cold Data

Karl Wolf ^a, Frank Pallas ^b and Sebastian Werner ^c

TU Berlin, Information Systems Engineering, Germany

Keywords: Big Data, Cold Data, Cloud Services, Access Control, Privacy, Function as a Service.

Abstract: Large datasets collected over a long time and only accessed on an infrequent basis – called Big Cold Data herein – play an important role in a broad variety of data-driven applications. In managing such data and the access to it, implementing advanced access control schemes beyond mere role-based yes/no-decisions becomes decisive, given the often sensitive or personal nature of the data as well as the multitude of regulatory requirements and other constraints applying to it. Current, mostly cloud-based technologies for storing and managing Big Cold Data, however, lack that advanced access control functionalities, such as consent-based transformations, while existing approaches for implementing such functionalities do not pay sufficient regard to the particularities of Big Cold Data to offer efficient access on an infrequent basis. We therefore propose an architecture and framework (ALASCA) following the function-as-a-service (FaaS) paradigm for implementing versatile access services on cloud-managed Big Cold Data. Towards that end, we offer a first characterization of *Big Cold Data* and raise challenges in access control, specifically in performing custom and infrequent transformations on large heterogeneous datasets. We demonstrate the applicability of ALASCA by implementing and evaluating it for AWS and Google Cloud. Our preliminary evaluation shows the promise and practical applicability of FaaS-based access control, especially for advanced access control schemes to be applied to Big Cold Data.


1 INTRODUCTION


Big Cold Data – large datasets accessed on a rather infrequent basis – play an increasingly important role in usecases such as interactive ad-hoc analysis or audits. At the same time, respective data are in many cases of personal or otherwise sensitive nature and must be effectively protected from illegitimate access. Moreover, such data are collected over many years and thus often subject to a wide variety of usage rights and access policies that require careful attention. In particular, access must often be restricted or provided depending on the pursued purpose, individual consent provided by the person(s) the data refers to, or other factors characterizing the accessing party. Thus, access to big cold data must deal with the longevity of stored data and related policies and the inherent uncertainty of using data legitimately.


Here, existing advanced access control schemes going beyond simple, established role-based models – such as (conditional) purpose-based (Agrawal et al.,

2002; Kabir and Wang, 2009), consent-based (Ulbricht and Pallas, 2018) or attribute-based access control (Hu et al., 2014) – become increasingly relevant for addressing regulatory and business requirements. In addition, prevalent usecases building upon Big Cold Data often call for access control models encompassing on-access transformations that generate individualized views on the same data, taking into account historic policies, original collection purposes, and stored user consent. Cloud platforms, being the dominant infrastructure for hosting and providing Big Cold Data, already come with rudimentary, typically role-based access control capabilities but do not provide these functionalities out of the box. Advanced access control schemes capable of ad-hoc transformation must therefore be implemented separately on top of existing cloud datastores such as AWS S3, Google Cloud storage, etc. This, in turn, raises a hard-to-resolve conflict:

On the one hand, advanced access control schemes, especially those providing capabilities for complex data transformations, require substantial computational power in order to not introduce inappropriate roadblocks. On the other hand, Big Cold

^a  <https://orcid.org/0000-0003-4607-7823>

^b  <https://orcid.org/0000-0002-5543-0265>

^c  <https://orcid.org/0000-0001-8051-7226>

Data are inherently characterized by rather infrequent (or “sparse”) access patterns, rendering computationally powerful and always-active access control systems economically inefficient in phases of non-usage. Cold standby models with such systems being started on-demand, in turn, would introduce overly long response times and thereby contradict the service-oriented and ad-hoc-focused nature of Big Cold Data. Established approaches of dedicated, high-throughput enforcement points, therefore, prove wasteful in the context of big cold data.

To resolve this tension and to thereby foster the practical adoption of usecases employing Big Cold Data in line with non-trivial privacy and security requirements, we propose a function-driven, extensible model and system architecture for implementing advanced access control schemes on Big Cold Data in a scale-to-zero fashion. This approach allows to rapidly scale computational power in the case of access actually happening, while reducing wasted costs in times without data access. We provide a prototypical, generic implementation – *ALASCA* – and two ready-to-use provider-specific instantiations for both S3 buckets on AWS and Google Cloud Storage, two of the most widely used stores for Big Cold Data. In particular, we herein propose and contribute:

- A foundational characterization of the concept of *Big Cold Data* and a set of specific challenges arising with regard to access control
- Our Adaptive, serverLess Architecture for Scalable, Content-level Access-control – *ALASCA* – a modular, advanced access control and data transformation *reference architecture* and *implementation framework* particularly tailored to the characteristics of FaaS platforms, thereby combining the capacity to handle bursty ad-hoc access with scale-to-zero capabilities
- *Prototypical instantiations* for two major cloud platforms, namely AWS and Google Cloud, and their *experimental evaluation* to assess the practical feasibility of FaaS-based advanced access control for Big Cold Data in a realistic usecase.

Our prototypes implement computationally heavy conditional consent-based access control and complex data transformations leveraging one of many possible policy languages – YaPPL (Ulbricht and Pallas, 2018). Our instantiations are thoroughly tailored to respective platform-specific particularities and design options following platform-standards for implementing big data access, i.e. using platform-provided identity management services. A preliminary evaluation for our prototype confirms the practical viability and cost-efficiency of our approach.

Section 2 briefly lays out and discusses related work. Section 3 characterizes the concept of Big Cold Data and identifies a set of specific challenges to be solved in this context with regard to implementing advanced access control schemes. Section 4, in turn, presents *ALASCA*, our FaaS-tailored reference architecture and implementation framework, which is transferred into the prototypical instantiations and their evaluation in section 5. Section 6 concludes.

2 RELATED WORK

Relevant related work particularly exists in the areas of access control for cloud-hosted object stores, of advanced access control schemes, and of FaaS-based data transformations. Regarding access control for cloud object stores, providers typically offer their own IAM, Access Control, and sometimes content-based querying solutions¹. These are by definition well-integrated and fast, but severely limited in the complexity and functional scope (e.g., regarding content-based access control) of their policies and result in lock-ins with respective cloud vendors. Pre-existing alternative approaches (Sampe et al., 2016; Saiz-Laudo and Sánchez-Artigas, 2022) allow the use of user-defined functions for more potent policy enforcement – including on the content level – but rely on users setting them up and scaling them manually rather than seamlessly supporting existing data in cloud storage.

In matters of advanced access control schemes, which are particularly necessary for Big Cold Data (see section 3.3), relevant paradigms include attribute-based / ABAC (Hu et al., 2014), purpose-based / PBAC (Byun et al., 2005; Agrawal et al., 2002), and consent-based access control / CBAC (Ulbricht and Pallas, 2018) as well as their conditional (Kabir and Wang, 2009) and content-based (Saiz-Laudo and Sánchez-Artigas, 2022) variants. Some of these have successfully been integrated into different system classes, from databases (Agrawal et al., 2002; Colombo and Ferrari, 2017) over publish-subscribe-systems (Wolf et al., 2021) to application-layer ORMs (Pallas et al., 2020). A dedicated focus on or a conscious, systematic, and coherent integration with cloud-hosted object stores – let alone an adaptation to the specifics of Big Cold Data – is, however, lacking.

Lastly, FaaS-based data transformations are increasingly proposed and used for conducting ad-hoc data analysis (Werner et al., 2018) and exploration (Müller et al., 2020), especially on rarely accessed

¹e.g. AWS IAM, Google Cloud IAM, S3 Select

data. To the best of our knowledge (and somewhat surprisingly), however, approaches for adopting the same concepts for implementing advanced access control schemes for Big Cold Data – particularly including data transformations – have not been made yet. The recently proposed *EGEON* (Saiz-Laudó and Sánchez-Artigas, 2022) employs user-defined policy functions, but does not leverage FaaS solutions. Instead, it is designed for self-hosted OpenStack Swift rather than utilizing public cloud services. Media-focussed transformations on the edge have been discussed as privacy measures, but target streamed data rather than object storage (Sedlak et al., 2023a).

3 BIG COLD DATA

This section defines the concept, characteristics, and challenges involved in managing access to Big Cold Data. To achieve this, we first present a usecase that exemplifies the characteristics of Big Cold Data before discussing the related access control challenges.

3.1 Motivational Usecase

Some of the most striving challenges for banks today relate to money laundering and fraud. In addition to mere know-your-customer rules, banks are obligated to identify and prevent respective activities. To meet these obligations, automatic detection systems process the vast amounts of transaction data generated every second (Carcillo et al., 2018), e.g., comparing new transactions to historical data to detect anomalies. However, due to the inherent limitations of automated systems, human operators must often manually re-analyze or investigate suspicious transactions.

This analysis requires access to historical transaction data, which may be stored in various outdated formats, due to the *longevity* of Big Cold Data. Moreover, investigations also include personal and sensitive payment data from customers, which is subject to *complex policies* dictating how and for what purpose it can be analyzed. Additional data, such as surveillance videos from ATMs, may be necessary in rare cases. The data to be processed is thus *heterogenous* and often provided through legacy interfaces alone (and possibly requiring legacy policy evaluation), calling for easily adaptable data (access) pipelines. At the same time, the inherent bigness of the data to be processed raises requirements in matters of performance and scalability for the access control mechanism to be used. This is particularly the case when transforming the data is a requisite of accessing it while at the same time, interactive analysis by an

operator requires a reasonable response time. Finally, using specialized tools should ideally work regardless of whether an object storage is accessed directly or through an additional access control layer, i.e. any additional access control solution should work with existing clients by preserving the original storage’s API.

For data stored in cloud object storage, access policies can basically be specified and enforced through the cloud providers’ individual IAM offerings. These are, however, severely limited with regard to legacy or complex policies and cannot perform transformations on the data. More powerful and versatile solutions, on the other hand, are subject to the above-mentioned conflict between economic efficiency and viable responsiveness. We thus need an access control layer that is both flexible and powerful and at the same time integrates seamlessly into existing heterogeneous cloud applications in a cost-efficient, yet responsive fashion.

3.2 Characteristics of Big Cold Data

Departing from the use case, we can grasp a set of characteristics of Big Cold Data particularly shaping the arena for respective access control mechanisms. First and foremost, Big Cold Data can be considered a subset of Big Data, which has traditionally been characterized by the *five Vs* (Kaisler et al., 2013). Beyond these, however, we identify the following additional characteristics of Big Cold Data that any proposed access control mechanism must take into account.

Heterogeneity. Big Cold Data comes from diverse and vast sources with varying data structures and formats, making data integration and analysis challenging. As a result, the data may not be in a suitable format for modern tooling or may require legacy systems to filter and process them. Capabilities for flexibly adapting to said legacy systems are thus of crucial importance.

Longevity. Big Cold Data is typically collected and stored for a long period, often for decades, with the intention of being used for historical analysis. However, due to this longevity, access and storage policies may vary based on the time of collection. This necessitates capabilities for data transformation and filtering due to the very heterogeneous data access policies that have grown over time.

Complexity. Big Cold Data may originate from a variety of sources, countries, time periods, modes of collection, and formats. Associated metadata may also be incomplete, inconsistent or outdated. Together, this introduces significant complexities to the actual use of such data.

Uncertainty. The type and purpose of later use of and access to Big Cold Data are often unpredictable in advance. This makes it intrinsically difficult to formulate conditions of use at the time of data collection.

In summary, Big Cold Data exhibits a unique set of characteristics and challenges arising thereof. Understanding these helps developers establish strategies for effectively analyzing and utilizing Big Cold Data in practice. This also applies to ensuring compliance with legal and regulatory standards and, hence, to the design and implementation of proper access control mechanisms:

3.3 Access Control Challenges of Big Cold Data

The challenges of access control in Big Cold Data are related to those present in Big Data, such as fine granularity, context management, and efficiency (Colombo and Ferrari, 2019). However, due to the longevity, heterogeneity, complexity, and uncertainty characteristics, additional challenges arise for access control, especially when considering the management of access to personal or otherwise sensitive data. In the following, we therefore elaborate on the unique set of traits that emerges for access control in Big Cold Data. Each of these is given a separate identifier (**C:XX**) for later reference.

Policy Variety (C:PV). Resulting from Big Cold Data's *heterogeneity* and *complexity* characteristics, the policies to be applied to these data are typically highly variant. Especially with unstructured data, they might also depend on the content of the data. In the example of fraud analysis, a log of payments may, for instance, contain data referring to many different parties with individual protection requirements, e.g., private citizens, international companies and federal institutions. In addition, combining data from different users (which, as data subjects, could express individual preferences under regulations such as the European General Data Protection Regulation / GDPR), different countries of origin (with their own legislations), and different times of collection (likewise resulting in different access policies) increases policy variety further. This, in turn, implies that the access policies to be applied are not only heterogeneous but also complex and dynamic, requiring developers to retrieve, attach and enact a broad variety of policies properly.

Diversity of Access Factors and Contexts (C:CtX).

Different parties accessing the same data with different purposes and in different contexts is one of the core implications of the above-mentioned *longevity* and *uncertainty* characteristics. In many scenarios, the mere role of the accessing party will here not be sufficient for proper access decisions. Instead, factors such as the *subject*, *purpose*, *object* (i.e., the file being accessed), *action* (e.g., reading, writing), and *environment* (e.g., time of access or type of device being used) must be considered in the access decision-making process. Moreover, access needs to be grantable or deniable in a granular fashion – e.g., only allowing access to specific columns or fields within the data – and transformations (see below) may be mandatory for certain combinations of subjects, purposes, and data content. For example, analyzing payment data may only be legitimate for very specific purposes and under specific protection settings, such as teams operating only from a dedicated office.

Computationally Heavy Policy Resolution (C:Res).

Big Cold Data stores typically handle large amounts of data. Combined with the *complexity* characteristic as well as the *policy variety* and *diversity of access factors*, computing ultimate access decisions can be very resource-intensive. This becomes particularly evident given that Big Cold Data often requires content-based access control strategies where the content of the data needs to be scanned to determine an access decision. In consequence, being able to provide sufficient resources for resolving (complex and content-based) policies in an ad-hoc fashion and on-access becomes critical.

Specifying and Applying Complex Transformation Policies (C:Tx).

In order to comply with requirements and constraints of *complex* data sets, non-trivial data transformations may be necessary on top of mere policy resolution. This could mean omitting some records (e.g., rows or columns of a CSV file), but more complex (and request-specific) transformations – such as redacting, anonymizing, or aggregating the data – can also be employed to make data usable that could otherwise not be utilized in a compliant way (Pallas et al., 2022). Depending on the transformations at hand, this requires the system to not only have the necessary execution performance, but also the foundational possibility to specify and program such policies including complex transformations. As for our usecase, anonymization may, for instance, allow data to be analyzed that would otherwise be unusable for fraud detection.

Evolving Policies (C:Ev). While our usecase is primarily aimed at cold data, requests may very well include “fresher” data as well, e.g., in the scenario of a bank operator comparing very old with current financial transactions. Additionally, even if the data itself stays unchanged, *uncertainty* regarding changes in regulations, environment factors, and access policies themselves may necessitate a continuous re-evaluation of the data in order to make correct access decisions. The system must thus evaluate access policies and data content dynamically and perform transformations or enforcement actions ad-hoc and on-access.

In addition to these general challenges, the basically unavoidable performance overheads resulting from any advanced access control system must stay within reasonable boundaries and the system to be proposed should integrate with existing cloud object stores as seamlessly as possible. Only then will many basically desirable usecases for Big Cold Data actually be practical to implement.

4 ALASCA

Based on the identified characteristics of Big Cold Data and the specific challenges in matters of implementing access control for such data, we see the need for an approach that is 1) adaptable to many access control needs (C:PV, C:Ev, C:Ctx), 2) highly scalable to react to different needs, particularly including bursty access patterns (C:Tr, C:Res); and 3) works well with off-the-shelf cloud resources. We thus propose an *Adaptive serverLess Architecture for Scalable Content-level Access control* (ALASCA), which is a set of policy-language agnostic, FaaS-optimized components explicitly designed for flexible integration into existing cloud architectures.

Basically, ALASCA consists of three main components, which are reflecting respective components from the XACML architecture (OASIS Standard, 2013): The policy enforcement point, a policy retrieval component, and a number of transformation components. The architecture does not specify how (or if) these components translate into and/or distribute across multiple functions, offering a high degree of implementational flexibility. Splitting functionality across different functions enables higher parallelization, but comes at the overhead of additional data transfers and cold starts. In section 5, we show a minimal example deployment that uses core ALASCA functionality in a single function, while keeping additional transformations (such as a dedicated anonymization tool) and policy needs (such as

connecting to existing policy infrastructures) in separate functions. Below and in fig. 1, the individual components and their interplay are illustrated in the context of a typical access request flow.

Client Request and Policy Enforcement Component. A client requests a file from an object storage (1), either through plain HTTP or through a dedicated client library, such as for S3. It can optionally be authenticated using the cloud providers existing IAM and forced to use the ALASCA access point instead of directly accessing the storage. The **access point** is the API endpoint the client interacts with. It may be implemented either through indirection (e.g., using AWS’s capabilities for having multiple access points to the same storage) or through a custom API implementation.

Object stores have their own API which can be used either via direct REST calls, through a provider-given client library, or using a plugin for another application or framework. Here, an important design decision regards how additional context as required for implementing advanced access control functionality, e.g., the access purpose, context-specific attributes – see **C:Ctx**, is to be provided. For clarity and easier setup, passing such information (as well as additional configuration options) directly per request would be beneficial. This would, however, render the API incompatible with the original ones, thus significantly hindering adoptions of clients that so far used the existing object storage APIs. Alternatively, respective information can be provided indirectly, e.g. through encoding it via different access points or file names. This would keep the API intact and make the system work as a *transparent proxy*, albeit at the expense of clarity and ease of maintenance.

The access point passes the request to the **policy enforcement service**, acting as the foremost point of orchestration and as policy enforcement point. It serves as both the input and output: it is called with the respective parameters (requested file, requesting entity, and its context) and at the end of the process serves either the requested file, a transformed version thereof, or an “access denied” response. If access is (at least partially) granted, the system pulls the requested object from the specified object storage (3). Here, it is a specificity of advanced access control paradigms that it may be necessary to request the file before making an access decision – e.g. when it must be determined which data subjects are affected and, consequently, which policies are to be applied (see **C:Res**).

In order to implement such content-aware access control efficiently, the data may be split (4), typically

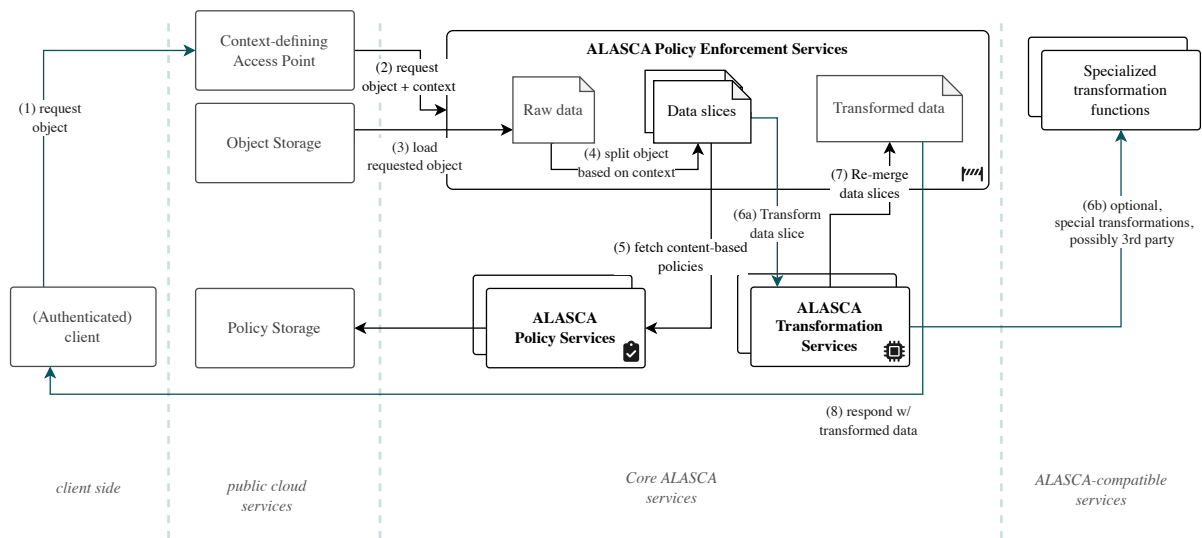


Figure 1: General architecture and request flow of ALASCA.

per data-subject as specified in the context. Assume, as a running example, that data subjects in a CSV file comprising financial transactions are denoted by an ID in a given column. For further content-aware treatment, data can then be split by this ID.² Each part of the data is now called a *data slice*. Depending on the chosen implementation and the configuration, all slices may then be handled within the core function or by additional functions called in parallel for each data slice.

Policy Retriever. For each slice, one or more policies are then fetched and evaluated (5). The system can have a multitude of **policy retrievers** that fetch policies on-the-fly (**C:Ev**) from different sources (**C:PV**). Policy retrievers may simply access a database or file, or perform more complex requests to external systems, potentially requiring some sort of authentication themselves. Each policy may contain one or more authorizing conditions based on which access is allowed. Depending on the access control paradigm to be implemented, these can refer to purposes, utilizers, context attributes, etc.

Transformation Service(s). In addition to mere yes/no-decisions, advanced access control schemes often also comprise the possibility to specify transformations to be applied on the data before releasing it (**C:Tx**). If present in an access policy, respective specifications and the data to be transformed are handed over to the **transformation function** (6).

²Other examples may include separating the data by regions or splitting a media file.

Whether a single function performs all transformations or transformations are distributed over multiple, e.g., per-slice functions is again subject to the overall ALASCA implementation.

The transformations themselves can serve various goals, ranging from confidentiality or privacy requirements to the provision of optimized views on existing data for the requesting party. In the most simple case, transformations might be mere selection or projection processes, i.e. only returning certain rows or columns. Further cases might require the redaction of individual fields (e.g., depending on individual consents provided by different data subjects covered in a dataset), more complex computations (such as for introducing differential privacy or other statistical operations) or to aggregate and generalize selected fields (for instance, to achieve guarantees such as k-anonymity). Other, non-privacy transformations might change data formats or offer additional data for a requested file, or filter the data according to a request in order to reduce the communication overhead. Finally, for non-text or even media objects, specific transformations of arbitrary computational complexity may be necessary, such as detecting and blurring all or specific faces in a photo or video. All transformations are usually called within the framework, but for highly specific operations (e.g. video-related changes or complex algorithms to achieve a given k-anonymity) or for performance reasons, individual transformations may also be located in a separate function (6b), potentially supplied by a 3rd party.

Completion. When all transformations are finished, the data slices are re-merged (7), preserving the initial

structure as much as possible. Finally, the data is then sent to the client (8), again mimicking the behavior of the original object storage as much as possible. In case of a completely denied request, the client is given an error response.

5 PROTOTYPICAL IMPLEMENTATION & EVALUATION

In order to demonstrate and evaluate the viability of the proposed architecture and its practical applicability in different cloud settings, we provide a twofold prototypical implementation: First, we implemented a Python-based general ALASCA framework that evaluates YaPPL policies (Ulbricht and Pallas, 2018) (other, even multiple, policy languages could easily be integrated, too) to provide Consent-Based Access Control and applies data transformations specified therein, using the pandas data analysis library. Second, we developed two vendor-specific instantiations consciously tailored to the specifics of different cloud platforms – Amazon AWS and Google Cloud Platform – that allow ALASCA to be used with minimal integration effort. The general ALASCA framework is implemented in line with the component / function structure laid out in section 4 and the code is publicly available for further refinement, extension and adoption.³ The most important details of the vendor-specific instantiations are delineated below, followed by a preliminary experimental assessment substantiating the practical viability of our approach.

5.1 Amazon Web Services (AWS)

AWS offers cloud object storage (S3) and a FaaS platform (lambda), but also allows to create access points that run the requested data through one or more specified functions: Object Lambda Access Points.⁴ This enables a fully transparent access model towards an S3 bucket where instead of the bucket, the function is called with the requested file as a parameter (as shown in fig. 2). In this model, the API stays identical to that of S3. Notably, all existing IAM settings of AWS are available for these access points as well, allowing to leave established authorization measures in place. The downside of such a fully transparent API is that additional context information relevant for

³ALASCA Code: <https://github.com/PolarFramework/alasca>

⁴<https://docs.aws.amazon.com/AmazonS3/latest/userguide/olap-create.html>

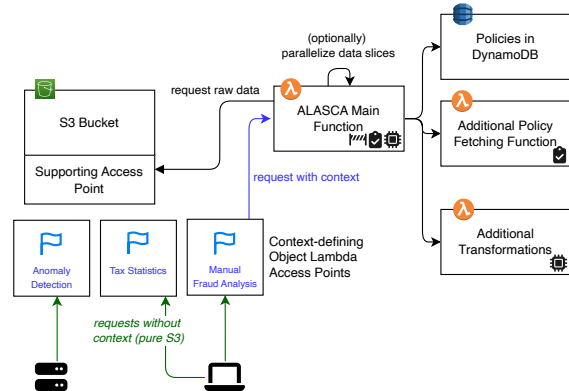


Figure 2: AWS Deployment: Function accessed through AWS’ built-in Object Lambda Access Points.

access control – such as the access purpose or utilizer – cannot be passed per request but must be reflected as combination-specific access points. For instance, an external vendor accessing the same data for multiple purposes, which in turn allow them access to different subsets or transformed variants of that data, requires an access point for each purpose. While creating an access point has no cost, when dealing with a high number of access-relevant parameters and combinations thereof, a high number of access points is required, giving rise to significant setup overhead. This requires the automation of access point creation, which can happen through an infrastructure as code approach such as AWS’ own CloudFormation or SAM, allowing the creation of access points through a configuration file. We employ the latter in our prototype: The given template supports the deployment of both the relevant functions and access points for each access scenario. The access context is – together with other configuration options to be provided to ALASCA’s core components (such as transformation and parallelization strategy) – given as access point *payload*.

5.2 Google Cloud Platform (GCP)

Google Cloud Platform provides object storage and functions, but currently has no feature comparable to Object Lambda access points. Therefore, we did in this case integrate ALASCA as a regular function that needs to be called directly with parameters being provided in the query string. As opposed to the Object Lambda approach used for AWS, this implies a loss of API compatibility and prevents a transparent drop-in use of ALASCA with pre-existing solutions expecting a Cloud Storage endpoint (see fig. 3), but can provide additional flexibility for, e.g., an analyst performing tasks in different contexts. For usecases requiring

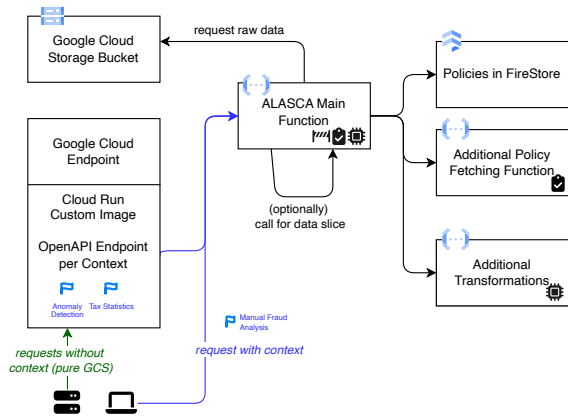


Figure 3: GCP Deployment: Function can be called directly or through a proxy endpoint.

seamless integration – e.g. for an automated service relying on the exact Google Cloud Storage API – we also implemented a separate REST endpoint with a compatible API. This endpoint is then again deployed once per parameter combination using Extensible Service Proxy (ESP)⁵ and redirects all traffic to the above-mentioned cloud function together with the necessary parameters. The wrapper, however, does currently not support GCP’s advanced authentication and authorization functions, making AWS the more potent deployment for cases that require both cloud IAM features and a transparent API.

For both deployment scenarios – with regards to the use case at hand – the core ALASCA components (policy enforcement, retrieval and transformation services, as in fig. 1) are placed in a single core function in order to minimize transferring overheads for basic cases. The function can access additional transformations and policy retrievers. In the prototype, the policies are retrieved from native cloud vendor offerings: AWS DynamoDB and GCP FireStore respectively. The function can call itself for parallel processing of data slices where appropriate, deliberately reusing the same function to mitigate cold starts.

5.3 Evaluation

As the imposed performance overhead is a decisive factor for the actual practical applicability of ALASCA, we conducted initial performance exper-

⁵In fact, the actual setup is even more complex and follows a well-established deployment pattern, comprising an API auto-generated from an OpenAPI schema, the respective API configuration then being retrieved back and then baked into a docker image, which is lastly deployed to Google’s image registry. For details, see <https://cloud.google.com/endpoints/docs/openapi/set-up-cloud-functions-espv2>

iments for both instantiations, also evaluating the impact of factors such as concurrency. To properly assess where observed overheads emanate from and how different factors impact the respective relations, we used two baselines in our experiments: first, a direct call to the object store (i.e. AWS S3 and Google Cloud Storage) and second, a modified version of our main guarding function that simply passes through the data without making any additional calls to fetch policies or perform any transformations. The second baseline thereby allows to assess the general integration overhead of our approach, while comparing actual enforcement functions to this second baseline provides insights about the impact arising within the ALASCA framework. Besides, all experiments were conducted in line with established practices of security- and privacy-related performance measurements (Pallas et al., 2020), comprising a warm-up phase before each experiment run, sufficiently sized client machines, etc.

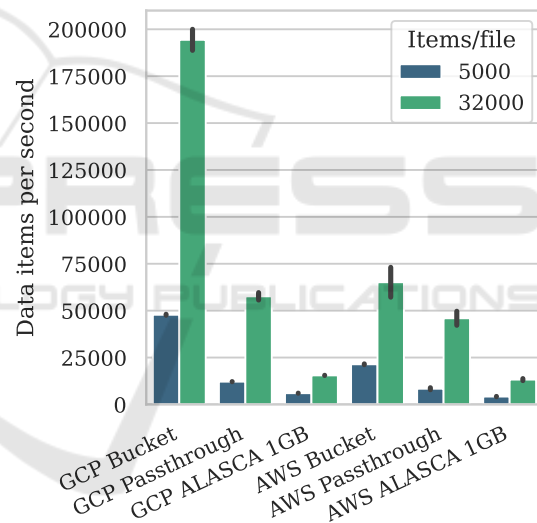


Figure 4: Throughput of ALASCA vs. object stores vs. passthrough.

As the goal was mainly to test the approach itself rather than the cost of individual transformations, we assume a basic scenario from our use case: A bank officer accesses old payment data for analysis, but is not allowed to see all details of said data (precise location and full IP as well as precise amount of the payment), resulting in some of the data being transformed before being returned. The data is provided in different CSV files containing between 5k and 64k payment data items referring to 10 different user IDs. As for the transformations, we used a realistic combination of redactions, including the individual manipulation of strings as well as the addition

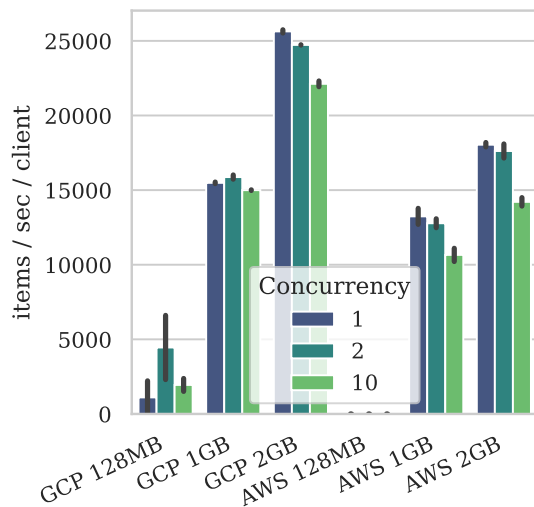


Figure 5: Effects of different function configurations and use of concurrent requests.

of noise based on statistical properties of the dataset that would be unfeasible with a simplistic row-by-row transformation strategy. Notably, the transformations do not significantly reduce the amount of data to be transferred. For further access scenarios, the transformations themselves could very well be more complex and thus lead to much higher performance degradation than in this setup. Still, we are mainly interested in the general overhead of ALASCA’s approach and not in that of different transformations, which are inherently application-specific. Throughput was measured using a custom measuring script (provided in the repository), which concurrently called a given target for a given period of time, similar to a tool like Apache Bench (ab).

Initial results are visualized in fig. 4: Directly accessing the object stores (S3 and GCS, respectively) obviously has the highest throughput, with the difference to accessing the same data through a passthrough function being significant (at 32k items/file: 70% throughput loss for GCP, 29% for AWS). Given the unavoidable computational efforts for de-and re-serialization, this was expectable. The actual transformation then adds additional overhead, reducing throughput by another $\sim 70\%$ compared to the passthrough case functions for both, AWS and GCP (with 1GB RAM). Figure 4 further shows that for larger objects with more items, the relative overhead becomes smaller, suggesting that the performance hit introduced through the functions is at least partially static.

Not surprisingly, function setups with more RAM (and in both cloud providers more vCPUs in consequence) performed significantly faster (see fig. 5);

the function deployed with 128mb RAM cannot process the 64k items on AWS, while on GCP the smallest function had an error rate of 50 percent for both 32 and 64k items. Functions provisioned with more memory did not show any errors within the experiment, although it is to be expected that they in turn have an inherent object size limit as well. Choosing the right function size is thus a trade-off between performance and cost, with a minimum requirement to ensure proper functioning and with the exact values foreseeably depending on the data and the planned transformations. The effect of parallelity, in turn, is also illustrated in fig. 5: performing concurrent requests only slightly slows down the achievable throughput per client (sometimes within measuring uncertainty), meaning that fetching multiple objects in parallel allows for scaling overall data acquisition speed. These observations show that the organization of data has a high impact on the performance of ALASCA and, most importantly, demonstrate ALASCA’s capability do actually handle bursty loads in the intended, scalable fashion.

Evaluating ALASCA’s cost as compared to an always-on access control and transformation service naturally requires consideration of the nature and number of requests. With requests as performed in the above experiment, a request uses between 1 and 2 GBs on Lambda, making the serverless approach cheaper than AWS’ cheapest EC2 VM for all cases with less than 100k requests per month (which would still amount to 138 requests per minute and, thus, more than sparse access).

5.4 Discussion

The evaluation demonstrates ALASCA’s general functionality and confirms that it scales elastically and provides – given that no particular optimizations were done so far – reasonable performance. We also find that ALASCA proves to address the aforementioned challenges of Big, Cold Data well: The prototype uses YaPPL, focussing on user consent, purpose and transformations, allowing for a broad set of policies (**C:PV**), which can be tailored to even individual parts of a dataset (**C:Ct \times**). The fact of different individuals being included and leading to different, content-dependent, and in themselves complex YaPPL policies being applied also illustrates ALASCA’s viability for cases requiring computationally heavy policy resolution (**C:Res**). Especially in Big Cold Data scenarios with long periods of non-usage, the performance overheads will in most cases be outweighed by ALASCA’s scale-to-zero capabilities and flexibility and, not to forget, the gained possibility to com-

pliantly use certain data at all. Through the use of FaaS, an unlimited variety of transformations, potentially including externally provided ones, can easily be applied to existing cloud data stores (**C:Tr**). On the other hand, ALASCA is also subject to some FaaS-inherent limitations such as overheads cumulating with a rising number of involved functions. Similarly, size or runtime limits may impact the main coordinator function when using synchronous client requests. This could partly be mitigated by caching some of the transformed data, even though this would impact ALASCA's capabilities for reacting to changing policy requirements (**C:Ev**) through purely ad-hoc request handling.

Some of these challenges are implementation-specific and could be mitigated through further optimization, while others are inherent to the chosen general approach and must be seen as unavoidable downsides of a highly flexible framework addressing the specific needs of Big Cold Data.

6 CONCLUSION AND FUTURE WORK

ALASCA proposes a flexible, FaaS-driven architecture that allows fine-grained advanced access control and transformations on general object storage and was shown to work with two major cloud providers. It allows an arbitrary amount of cold data to be accessed whilst undergoing complex transformations in accordance with user preferences, legal obligations or compliance guidelines through fine-grained, content-aware access policies at no up-front or standby costs. Insofar, ALASCA provides a highly flexible and practically viable approach for implementing advanced access control, especially for Big Cold Data.

Still, ALASCA's current, prototypical implementation also exhibits some framework and platform-driven limitations (Werner and Tai, 2021). Especially with regard to the performance, we feel further work is both necessary and promising, such as done in (Müller et al., 2020; Eismann et al., 2020). Trying to leverage vendor-specific functionalities such as S3 Select appears as one valuable path here. Similarly, exploring alternative implementation patterns (e.g., also allowing for an asynchronous use) as well as exploring (domain-specific) access patterns, necessary transformations, and other application requirements will also unlock further potentials for optimization and functional extension. Furthermore, adapting ALASCA for non-cloud FaaS environments like OpenWhisk could also yield further interesting insights. In matters of cost-efficiency, ALASCA's su-

periority over powerful and always-active access control systems remains to be further quantified. Like for any other FaaS-based practice, however, we expect this superiority to become particularly prevalent in scenarios and environments shaped by infrequent access and usage patterns as given the domain of Big Cold Data.

For the time being, however, we herein demonstrated that and how a function-driven approach can be used to flexibly implement advanced access control paradigms in a way particularly tailored to the specific characteristics of Big Cold Data. Given that such advanced access control mechanisms are increasingly required in real-world usecases (Sedlak et al., 2023b) but not yet supported by cloud providers, our work thereby paves the way for practically implementing a broad variety of usecases at all.

REFERENCES

- Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2002). Hippocratic Databases. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 143–154. Elsevier.
- Byun, J.-W., Bertino, E., and Li, N. (2005). Purpose based access control of complex data for privacy protection. In *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies - SACMAT '05*, page 102, Stockholm, Sweden. ACM Press.
- Carcillo, F., Pozzolo, A. D., Borgne, Y.-A. L., Caelen, O., Mazzer, Y., and Bontempi, G. (2018). SCARFF: A Scalable Framework for Streaming Credit Card Fraud Detection with Spark. *Information Fusion*, 41:182–194.
- Colombo, P. and Ferrari, E. (2017). Enhancing MongoDB with Purpose-Based Access Control. *IEEE Transactions on Dependable and Secure Computing*, 14(6):591–604.
- Colombo, P. and Ferrari, E. (2019). Access control technologies for Big Data management systems: Literature review and future trends. *Cybersecurity*, 2(1):3.
- Eismann, S., Grohmann, J., van Eyk, E., Herbst, N., and Kounev, S. (2020). Predicting the Costs of Serverless Workflows. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pages 265–276, Edmonton AB Canada. ACM.
- Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. (2014). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Technical Report NIST SP 800-162, National Institute of Standards and Technology.
- Kabir, M. E. and Wang, H. (2009). Conditional purpose based access control model for privacy protection. In *Proceedings of the Twentieth Australasian Conference on Australasian Database - Volume 92, ADC '09*, pages 135–142, AUS. Australian Computer Society, Inc.

- Kaisler, S., Armour, F., Espinosa, J. A., and Money, W. (2013). Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004.
- Müller, I., Marroquín, R., and Alonso, G. (2020). Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 115–130, Portland OR USA. ACM.
- OASIS Standard (2013). eXtensible Access Control Markup Language (XACML) Version 3.0.
- Pallas, F., Hartmann, D., Heinrich, P., Kipke, J., and Grünewald, E. (2022). Configurable Per-Query Data Minimization for Privacy-Compliant Web APIs.
- Pallas, F., Ulbricht, M.-R., Tai, S., Peikert, T., Reppenhagen, M., Wenzel, D., Wille, P., and Wolf, K. (2020). Towards application-layer purpose-based access control. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1288–1296, Brno Czech Republic. ACM.
- Saiz-Laudó, R. and Sánchez-Artigas, M. (2022). Egeon: Software-Defined Data Protection for Object Storage. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 99–108.
- Sampe, J., Garcia-Lopez, P., and Sanchez-Artigas, M. (2016). Vertigo: Programmable Micro-controllers for Software-Defined Object Storage. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 180–187, San Francisco, CA, USA. IEEE.
- Sedlak, B., Murturi, I., Donta, P. K., and Dustdar, S. (2023a). A Privacy Enforcing Framework for Data Streams on the Edge. *IEEE Transactions on Emerging Topics in Computing*, pages 1–12.
- Sedlak, B., Pujol, V. C., Donta, P. K., Werner, S., Wolf, K., Falconi, M., Pallas, F., Dustdar, S., Tai, S., and Plebani, P. (2023b). Towards Serverless Data Exchange Within Federations. In Aiello, M., Barzen, J., Dustdar, S., and Leymann, F., editors, *Service-Oriented Computing*, Communications in Computer and Information Science, pages 144–153, Cham. Springer Nature Switzerland.
- Ulbricht, M.-R. and Pallas, F. (2018). YaPPL - A Lightweight Privacy Preference Language for Legally Sufficient and Automated Consent Provision in IoT Scenarios. In Garcia-Alfaro, J., Herrera-Joancomartí, J., Livraga, G., and Rios, R., editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, volume 11025, pages 329–344. Springer International Publishing, Cham.
- Werner, S., Kuhlenkamp, J., Klems, M., Müller, J., and Tai, S. (2018). Serverless Big Data Processing using Matrix Multiplication as Example. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 358–365, Seattle, WA, USA. IEEE.
- Werner, S. and Tai, S. (2021). Application-platform co-design for serverless data processing. In Hacid, H., Kao, O., Mecella, M., Moha, N., and Paik, H.-y., editors, *Service-Oriented Computing*, pages 627–640, Cham. Springer International Publishing.
- Wolf, K., Pallas, F., and Tai, S. (2021). Messaging with Purpose Limitation – Privacy-Compliant Publish-Subscribe Systems. In *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 162–172. IEEE Computer Society.