Software-Defined Data Protection: Low Overhead Policy Compliance at the Storage Layer is Within Reach!

Zsolt István
IT University of Copenhagen
zsis@itu.dk

Soujanya Ponnapalli
University of Texas at Austin
soujanya.ponnapalli@utexas.edu

Vijay Chidambaram
University of Texas at Austin and
VMware Research
vijay@cs.utexas.edu

ABSTRACT

Most modern data processing pipelines run on top of a distributed storage layer, and securing the whole system, and the storage layer in particular, against accidental or malicious misuse is crucial to ensuring compliance to rules and regulations. Enforcing data protection and privacy rules, however, stands at odds with the requirement to achieve higher and higher access bandwidths and processing rates in large data processing pipelines.

In this work we describe our proposal for the path forward that reconciles the two goals. We call our approach "Software-Defined Data Protection" (SDP). Its premise is simple, yet powerful: decoupling often changing policies from request-level enforcement allows distributed smart storage nodes to implement the latter at line-rate. Existing and future data protection frameworks can be translated to the same hardware interface which allows storage nodes to offload enforcement efficiently both for company-specific rules and regulations, such as GDPR or CCPA.

While SDP is a promising approach, there are several remaining challenges to making this vision reality. As we explain in the paper, overcoming these will require collaboration across several domains, including security, databases and specialized hardware design.

PVLDB Reference Format:
Zsolt István, Soujanya Ponnapalli, and Vijay Chidambaram.
Software-Defined Data Protection: Low Overhead Policy Compliance at the Storage Layer is Within Reach!. PVLDB, 14(7): 1167-1174, 2021.
doi:10.14778/3450980.3450986

1 INTRODUCTION

Our online presence has been generating unprecedented amounts of data, a large portion of which are personally identifiable and hence prone to misuse. Even though companies have long used different access control and encryption techniques to secure the information they collect, with the emergence of regulatory frameworks such as GDPR in the EU [12] and CCPA in California [8], there is a need to homogenize and perhaps standardize these techniques to enforce rules at all levels of application stacks [37, 41]. Enforcing privacy rules through all the processing steps of large-scale applications is important but the storage layer plays a disproportionately important role. With it being the layer that persists the data and acts as its source and destination, it is important to ensure efficient compliance with rules and regulation. Matter of fact, analysis of GDPR finds that more than 30% of the articles enforcing data protection are directly related to storage [39]. However, maintaining strict compliance, as done today, is challenging because it requires computational resources beyond the capacity of storage nodes and slows down data access.

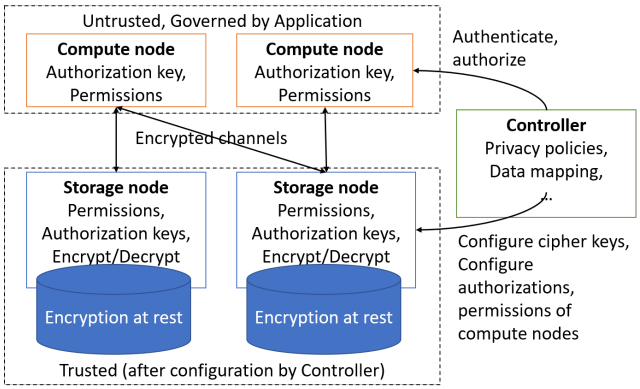

Figure 1: We propose Software-Defined Data Protection as a design approach to decouple policies from storage node implementation.

In this work we make the case that there is an optimistic outlook to enforcing privacy rules directly in the storage layer – mainly driven by two recent trends. First, disaggregated architectures with query pushdown to the storage layer, such as Amazon Aqua [1], are becoming a commodity in the cloud. There are also several SSDs on the market with in-storage processing [10, 36] and high bandwidth connectivity to the network. Second, as an effort to keep the management, configuration and monitoring of a large number of storage nodes scalable, Software-Defined Storage (SDS) has been proposed [42]. Our work relies on the realization that even though the goal of existing SDS systems is to guarantee performance isolation and service levels in distributed multi-tenant settings [30], the underlying ideas can be translated to the data protection domain.

The growing number of specialized-hardware-based smart storage solutions show that it is already challenging to keep up with increasing network speeds and workload complexities. When factoring in additional security and privacy operations, it is important to establish how we can rely on offloading functionality to avoid slowdown. Our goal is to make it feasible to implement rich policies, such as GDPR or CCPA, with smart storage devices at the network line-rate. This, however, will only happen if we use them to do what they do best: pipelined data processing. To deal with complex

Proceedings of the VLDB Endowment, Vol. 14, No. 7 ISSN 2150-8097.
doi:10.14778/3450980.3450986

decision-making code required for policy interpretation, we propose the use of a software-based logically centralized controller. We call the approach *Software-Defined Data Protection (SDP)*. Figure 1 shows how policy interpretation and decision making (i.e., the control plane) is separated from applying rules to request processing (data plane).

We validate our SDP vision by sketching how GDPR could map to it. We chose GDPR because it is relatively restrictive and therefore it can be seen as a superset of possible company-wide policies. It is also a precursor to other, similar, regulatory frameworks, such as CCPA in California. As a result, if our proposal can handle GDPR, it will work with other rule-sets as well.

To summarize, **our vision is that using in-storage computation together with a control-plane/data-plane separation will allow enforcing data protection rules at line-rate**, without requiring storage nodes to be implemented with large, power-hungry, servers. Somewhat surprisingly, much of the required in-storage functionality can be provided by re-purposing existing hardware building blocks, opening the path to ensuring regulation compliance without negatively affecting performance.

There, are, however, two remaining challenges to making SDP reality: One is related to bootstrapping storage nodes and will require adding Trusted Execution Environments (TEEs) [18] inside the storage nodes to enable remote attestation of the firmware by the controller. The second is the translation of complex, high level, rules and policies at the controller level to the common SDP substrate. Today's solutions in this space, e.g., [5, 31, 38, 44], already use secure enclaves and forms of offloading but all implement a different interface and low-level enforcement logic.

2 BACKGROUND AND RELATED WORK

2.1 Implications of GDPR on Storage

The General Data Protection Regulation (GDPR) outlines the rights and responsibilities of the companies handling the personal data of EU citizens. GDPR regulates the entire life cycle of personal data, from its collection to its deletion. A vast majority of companies today rely on cloud services providers, for their infrastructural needs, making GDPR compliance in cloud environments a necessity. Therefore, rich related work proposes frameworks enabling cloud users to develop GDPR-compliant applications [35], GDPR-compliant databases [25, 37], along with studies that outline the compliance challenges faced by cloud users [11, 21, 41]. Other works propose benchmarks [40] and tools [29] that test GDPR-compliance, and explore the benefits of trusted hardware to prove compliance [31].

Below we summarize the features identified by related work [39] and required for a storage system to be GDPR-compliant [39]. Our proposal uses GDPR as its main use-case because it is a framework with strict requirements and representative for other state-wide regulations, such as CCPA. The six features are:

1. User-Specific Deletion. GDPR introduces the right to be forgotten, allowing users to demand the deletion of personal data in a timely manner. GDPR also states that personal data cannot be stored beyond its purpose in its storage limitation clause.

2. Logging and Monitoring. With GDPR, companies are vested with the responsibility of detecting potential data breaches, informing users about those data breaches, and proving their compliance.

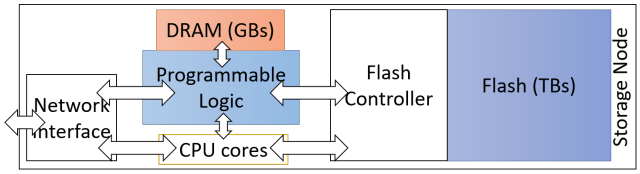

Figure 2: Emerging Smart Storage nodes incorporate both low-power CPU cores and Programmable Logic (FPGAs) offering both flexibility and predictable performance.

GDPR also provides users with the right to access, allowing them to request with whom and why their personal data is shared.

3. Metadata and Secondary Indexes. As GDPR requires personal data to be associated with a specific purpose, storage has to accommodate for additional metadata. Even though not a requirement, secondary indexes that categorize data as per purposes can be useful as an optimization.

4. Fine-grained Permissions. With GDPR's right to object, users can object to using their personal data for specific purposes. Further, with the purpose limitation clause, GDPR disallows companies to process user's data beyond its purpose and without appropriate security measures. To comply with these clauses, fine-grained permission checks and access control are required at the storage layer.

5. Encryption. GDPR mandates that personal data should be protected against accidental loss or damage and should be incomprehensible to any person unauthorized to access it. Encrypting personal data is a suitable measure to comply with this clause.

6. Location Control. GDPR requires companies to adhere to its standards, independent of the geographical location of where personal data is stored.

7. Data Integrity. GDPR requires storage systems to resist or detect malicious activities that compromise the integrity and confidentiality of personal data. Using checksums or Merkle tree-based data integrity checks have been proposed to comply with the integrity and confidentiality clause of GDPR.

2.2 Emerging Smart Storage Devices

The database community has long demonstrated the potential of "smart storage" [4, 16, 20] and now it is becoming mainstream. Amazon Aqua [1] provides query offloading to distributed storage using specialized hardware and there are commercially available Samsung SmartSSDs [36] appearing in the cloud as well.

As several studies showed, building smart storage only relying on small CPU cores (often by borrowing cores from the controller of the flash device itself) is very sensitive to the types of offloaded processing and can often become a severe bottleneck, instead of alleviating one [19, 23]. As a result, the architecture of smart storage nodes typically incorporates some form of specialized ASIC [1, 19] or a more general purpose Field Programmable Gate Array (FPGA) [4, 16, 20, 36]. Figure 2 depicts a high level overview of a modern smart storage node, attached to fast networking to be used in disaggregated architectures.

Related work in the space of FPGA-based smart storage has shown that it is possible to implement line-rate hash-tables [16],

deduplication schemes [26], in-storage filtering with regular expressions [15] on FPGA-based distributed storage. These operations are similar in nature to those required for building an SDP system (discussed in Section 3.3) and can be re-purposed (re-implemented) to this domain.

We sketch our proposal targeting today's smart storage platforms, that is, with a combination of low-power CPU cores and FPGA logic, such as the Fidus Sidewinder-100 [4] or the Samsung SmartSSD [9, 36]. The presence of both a general-purpose, albeit low power, CPU, and a re-programmable hardware element ensures that the devices can be managed remotely by software and, at the same time, can deliver high-performance behavior for privacy-related processing.

3 SOFTWARE-DEFINED DATA PROTECTION

3.1 Assumptions and Threat Model

Software-Defined Data Protection (SDP) targets cloud and datacenter use-cases within the context of a large corporation. Data is being stored in the distributed storage layer and is processed by several applications governed by a single set of privacy rules. We assume that user/client data can be identified explicitly through a universal *user identifier* (e.g., corresponding to a real-life person) and, depending on the purpose for which the data has been collected for, a *purpose identifier* (e.g., billing purposes, recommendation service purposes). Some applications will represent a single purpose (e.g., notifying users of unpaid bills) and some will map to several purposes (e.g., recommendations based on previous purchases), hence identifiers are valid across applications of the company, and the trusted SDP controller has a global view of which application can access what *purpose*s. All *user*s within a *purpose* are accessible to the application, unless consent has been revoked (e.g., user opting out from personalized recommendations).

We assume that attackers could tamper with the storage nodes while powered down (devices can be stolen or lost) but not when powered on and having been bootstrapped by the SDP controller. Therefore, it is important that the storage nodes provide encryption to safeguard persisted data and mechanisms for detecting when data has been modified or corrupted off-line. Furthermore, applications can be temporarily faulty, or even malicious, so SDP has to protect against data loss and corruption. This is achieved by checking permissions for all accesses to *user*s or *purpose*s. SDP allows several applications belonging to the same company (controller) sharing storage nodes, but, in the current design, using more than one controller per storage node is not envisioned.

3.2 The Anatomy of an SDP Deployment

Figure 1 shows the three types of nodes in SDP: *storage*, *processing*, and *controller*. **Storage nodes** implement a general-purpose key-value store (KVS) interface on binary data, can be shared across applications, and all communication with them happens over an encrypted channel (e.g., TLS). Applications run on one or several **processing nodes** and are managed under the governance of the developers. Applications have to register with the controller before accessing data but once granted access, can carry out most of their operations directly with the storage nodes.

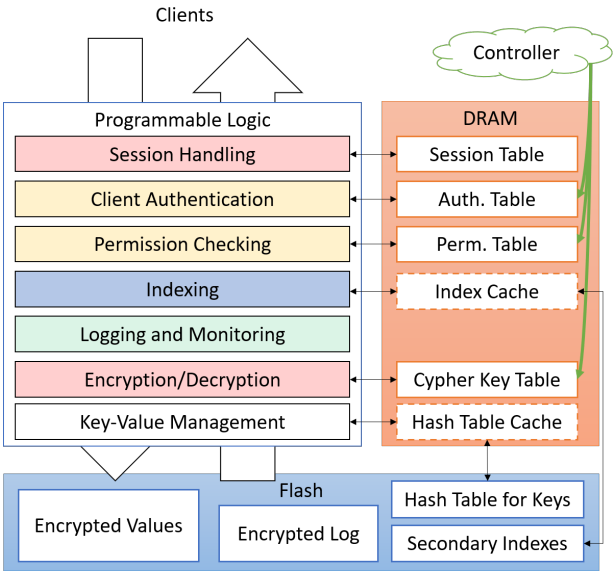

Figure 3: SDP enables separation of concerns, and as a result, policy enforcement can be laid out as pipeline within the storage nodes. The SDP controller configures and manages the nodes from the outside.

The **controller node** is logically centralized and is trusted by both the storage and processing layers and is controlled by the company's data officers. It interprets policies and maintains data mappings and is used to bootstrap storage nodes, authenticate application nodes, and manage their permissions. Storage nodes are trusted once configured by the controller. As we explain in the following subsections, once authenticated and configured, in common case neither the application nor the storage nodes have to communicate with the controller. For this reason, we do not consider the controller as a performance bottleneck.

Based on the assumptions and threat model, we depict the hardware pipeline structure for SDP storage nodes in Figure 3. It handles authenticated connections and encryption at the transport layer, as well as inside the storage. It also performs permission checks and detailed logging and monitoring. Furthermore, to enable point lookups and queries based, for instance on purpose-identifiers, it also includes a secondary indexing element. The controller can configure the behavior of SDP pipeline stages through the in-memory tables, depicted on the right-hand side in Figure 3. This means that the physical implementation of the stages can change without having to modify the controller or the SDP interface.

Given the threat model, SDP will only be effective if all permissions are verified and honored correctly by the storage node, even if it has been tampered with while powered down. Therefore, it is crucial to ensure it is running the expected software/firmware so that the Controller can trust the storage node once it has been powered on and "booted". For this, we propose expanding the storage node with a Trusted Execution Environment that can attest the correctness of the software and hardware contents to the Controller

– this is, however, a challenge that requires collaboration across research domains and is explained in more detail in Section 5.

3.3 Pipeline Functionality

The pipeline in Figure 3 assumes all in-storage processing to take place on an FPGA. In real-world implementations it can be beneficial to only keep performance-critical steps in hardware and move others to more traditional CPU cores within the storage nodes, reducing this way the development effort. The SDP approach makes this possible because the pipeline stages are loosely coupled. As an example, non-performance-critical steps such as Authentication that is only carried out once per session could be handled by a CPU core that shares in-memory data structures with the hardware modules, achieving this way heterogeneous processing.

In the following, we explain how the SDP pipeline provides the salient features required for enforcing company-wide policies or higher level regulations in an efficient way:

Encryption. Encryption can be implemented inside the storage layer in a way that *a)* relies on existing schemes and best practices and *b)* makes it possible to map the key-management operations to the SDP scheme we propose.

Data needs to be encrypted both at rest and on the move. We envision a system where clients (processing nodes) receive plain-text tuples over encrypted channels. Block ciphers underlying TLS have been shown to work well on FPGAs [7, 13] reaching throughputs high enough to saturate even 40Gbps links.

Persistent data on flash is always encrypted, assuming industry-standard, symmetric-key cryptography (e.g., AES). The storage nodes must not persist cipher keys. Instead, they have to remain in memory and need to be configured and managed by the SDP controller at run-time. This forbids unauthorized access to the drives and prevents leaks. One difference to traditional encrypted storage is that in the context of fine-grained rules and regulations, such as GDPR, it can be beneficial if each tuple is encrypted with a key specific to the *user* whose data it represents (or even the *user-purpose* combination) because, as later explained in the *Deletion* subsection, this enables quick logical deletion of data.

A side-effect of having multiple (de)encryption keys is that each client request will have to retrieve a different one. Since requests encode *user* and *purpose* explicitly, the storage node can use this to lookup the cipher key in an internal ephemeral cipher key table (KT). It is important that this table can sustain high access rates because, in contrast to other meta-data structures described later, this one has to be accessed on every incoming client request.

One challenge of creating fast hash tables on specialized hardware is that it is unlikely that cipher keys will fit on on-chip SRAM memory and will have to spill into off-chip DRAM (there is recent work in the context of high-performance key-value stores built on FPGAs [16, 28] using DRAM, showing feasibility).

Fine-grained Permissions. Beyond the question of how clients can reach storage devices (solved by SDS), an authentication step is necessary. Authentication matches a client's identity to a set of permissions (read/write/insert rights per *purpose*). To carry out authentication, an Authentication Table (AT) holds the public keys belonging to applications. A Permission Table (PT) stores the mapping of identities to permissions. Both tables are ephemeral and are populated and managed by the controller. In most workloads, the number of *purposes* (e.g., number of internal applications) will be orders of magnitude smaller than that of individual *users*, hence, the PT can be represented compactly, perhaps even on on-chip caches.

Permissions in the permission table (PT) are orthogonal to the presence/absence of cipher keys in the key table (KT): even if a client has the right to read all key-value pairs belonging to a *purpose*, only those for which the storage device holds a cipher key in the KT can be successfully read. The same holds for inserting tuple belonging to a new *user* or *purpose*. The SDP controller has to first insert the corresponding entries in the KT. It is important to note that permissions in themselves do not forbid applications to, for instance, make copies of data under bogus *user* keys. For this reason, end-to-end information tracking is required, as we highlight this in Section 5.

Metadata and Indexing. We expect that, in company-specific rules and national regulations alike, auditing (internal or external) will require an efficient way of retrieving all tuples belonging to a specific *user* or *purpose*. This can be achieved by relying on meta-data stored with the key-value pairs and by enforcing a tuple naming scheme.

While not strictly necessary, if such reads will occur often, it can be beneficial to maintain secondary indexes for performance reasons. Even though write operations will become more expensive, these data structures can be used to avoid scanning TBs of data to find a specific *user*'s entries. For this purpose, there are already FPGA-based key-value stores with low cardinality secondary index [16] and by using the same hash table approach as for permissions, etc., higher cardinality cases could be also handled.

Logging and Monitoring. Logging is important for ensuring auditability of the storage layer and can be implemented at various granularities. The storage node will persist an encrypted log (the key for the log is configured at run-time by the SDP controller) and implement some form of integrity check at the tuple level. While we foresee no challenges with the former task, the latter might require further investigation. Increasing the efficiency of data structures that ensure the integrity storage are still a topic in exploration [3, 33, 34].

For monitoring, the storage node has to notify the SDP controller of any request that failed any of the validation steps outlined above or has retrieved a key-value pair whose decryption key is missing. These events allow the controller to take adequate action, rectifying mis-configuration, revoking permissions, etc.

Efficient Deletion. As explained above, in many cases it can be a requirement that the storage can remove all data belonging to a specific *user*, e.g., once they revoked consent. In this work we focus on logical deletion of *user* data, since physically destroying all copies, and proving this to a third party, is orthogonal to our goals and a challenge in itself [22, 32]. If using an encryption scheme with a single key, the SDP controller can delete tuples belonging to a *user* and *purpose* by delegating this task to the storage device that will either scan the tuple space or use a secondary index. But in case an encryption scheme with multiple keys is used, deletion can be performed more efficiently by removing the corresponding cypher keys from the KT of the storage nodes. As a result, deleted tuples will not be accessible any more as plain-text, and without

Table 1: This table summarizes the GDPR articles relevant to storage and the high level functionality that the storage nodes and SDP controller need to fulfill those. Functionality outside of the scope of this paper is marked with †.

No.	GDPR article	Required functionality	Impacts mostly
5.1	Purpose limitation (data collected for specific purpose)	Fine-grained permissions	Storage, Controller
21	Right to object (data not used for objected reason)	Fine-grained permissions	Storage, Controller
5.1	Storage limitation (data not stored beyond purpose)	Deletion	Controller
17	Right to be forgotten	Deletion	Controller
15	Right of access by users	Metadata (and Secondary indexes)	Storage
20	Right to portability (transfer data on request)	Metadata (and Secondary indexes)	Storage
5.2	Accountability (ability to demonstrate compliance)	Logging and Monitoring	Storage, Controller
30	Records of processing activity	Logging	Storage
33, 34	Notify data breaches	Logging and Monitoring	Storage, Controller
25	Protection by design and by default	Encryption	Storage
32	Security of data	Encryption and Access control	Storage
13	Obtain user consent on data management	High level policy†	Controller
46	Transfers subject to safeguards	Location control†	Controller

the cipher keys, cannot be decrypted. This approach is in-line with existing practices [6, 46].

Bootstrapping and Interfaces. When powered on, the storage nodes will have to load the correct firmware/software (attested by the TEE) before they can operate on the persisted data. The controller is responsible for performing these steps and then populating the in-memory data structures of the SDP pipeline. These tables represent the interface to the SDP pipeline and decouple the physical implementation from the meta-data required to perform them.

Integrity checks after startup can be performed once the controller bootstraps the nodes and loads the crypto keys (until this happens, the key-value store of the storage node cannot decrypt the values). Data that cannot be decrypted with its corresponding key can be considered corrupted or tampered with.

3.4 Features and Questions out of Scope

Location. The mapping of *user* and *purpose* to the actual storage devices is carried out by the controller following either general purpose sharding strategies or depending on the privacy policies at the high level. Company-wide rules and regulations, such as GDPR, however, can mandate that regardless of the physical location of data belonging to a *user*, the same rules apply to it. The logically centralized nature of the SDP controller is essential for ensuring rule consistency at scale.

Fault Tolerance through Replication. For simplicity, in our discussion we assumed that each piece of data resides inside a single storage node. In a real system, however, replication will be required to ensure fault tolerance. As highlighted by the above two points, the task of setting up and controlling replication is external to our proposal and can be tackled by numerous existing schemes. Nonetheless, there is work on performing transparent, line-rate, replication of the KVS running on the FPGAs [17] that could be easily adapted to be managed by the Controller.

Performance of the Controller.C In our SDP vision, the Controller is required to actively participate only in a subset of operations. After carrying out initial authentications and configuring encryption keys and permissions, it is seldom accessed by either the processing nodes (application) or the storage nodes. Once exception is when data belonging to a new user or purpose is inserted for the first time. Such operations, however, are less common than regular reads and updates of existing data. Nonetheless, it is likely that the Controller will have to be implemented as a logically centralized by physically decentralized solution, to be able to keep up with the workloads of large enterprises.

4 FEASIBILITY STUDY

4.1 Does GDPR Map to SDP?

In Table 1, we summarize the GDPR articles relevant to storage and what type of functionality is required for their fulfillment. We also indicate for each aspect, whether the SDP storage nodes (data plane) or the controller (control plane) would bear most of the required complexity. Naturally, even if the storage node performs almost all the computation, for instance, as is the case with Encryption, the controller will still have to bootstrap the nodes. As visible from the table, the required functionality maps well to the SDP model with the storage node being able to utilize the specialized hardware pipeline to offload all data-intensive tasks. This means that, with SDP, GDPR-compliance is possible without negatively impacting the performance of the applications running on top.

The main remaining challenges for providing GDPR-compliance in practice, namely the mechanisms for obtaining user consent and controlling the location of data, are pertinent to the controller implementation and the way this interprets the regulation. We talk more about these aspects as part of the future work outlined in Section 5.

4.2 Are FPGAs the Right Platform?

We "approximate" the proposed SDP functionality to verify whether it can be implemented on today's SmartSSD solutions while guaranteeing 10Gbps and faster line-rate behavior. As a stand-in for the SDP functionality, we rely on the multi-tenant version of Caribou [14], an open-source FPGA-based key-value store. This version

Table 2: Resource consumption of single encryption and decryption cores (AES128-CBC) and the estimated size of the crypto engines necessary for sustaining 10Gbps network line-rate on a Virtex Ultrascale+ VU9P device.

Module	Logic Blocks (%)
AES128 Decryption core (1xD)	4.4k (2.9%)
AES128 Encryption core (1xE)	1.8k (1.2%)
Key-Value Store [14]	28.4k (19%)
+ AES128 Engine (1xD + 4xE) for TCP	±12k (±8%)
+ AES128 Engine (3xD + 4xE) for Storage (>256B)	±20.5k (±13.5%)
Estimated total with AES	±61k (±41.5%)

of Caribou already handles TCP connections from multiple tenants and is able to allocate bandwidth to them independently, it implements logic roughly equivalent to the Session and Authentication tables in SDP. It also has secondary bit-map indexes, which require similar logic as needed for Indexing in SDP. The main type of functionality that is missing from this "approximation" is encryption, decryption and authentication. Since authentication is not a performance-critical task, we focus in the following on quantifying the cost of providing encrypted TCP connections to clients and the encryption of the data at rest.

We investigate the cost of a commonly used encryption method that uses symmetric keys, namely AES-128 in CBC mode. We deploy a single encryption and decryption module (core) on a Xilinx VCU1525 board with an UltraScale+ VU9P FPGA (a chip similar to the one in the Amazon F1 machine instances). The cores are implemented relying on the Xilinx Vitis 2020.1 library. Table 2 shows that the single cores occupy only a small portion of the logic resources on the FPGA and do not utilize DSPs (small arithmetic units on the FPGA fabric) or BRAM (on-chip memory distributed over the FPGA fabric), except for buffering input. To provide a realistic SDP implementation, two AES engines, composed of several AES cores, are needed. One engine handles the network traffic and the other one secures data on the storage medium. As an estimate of the logic resources required in a real implementation, we estimate the size of a 10Gbps line-rate capable AES engine for the TCP traffic and the storage based on the single-core performance. Figure 4 shows how the AES cores behave with increasing packet size. Encryption has less overhead and the performance of a single core is less impacted by the size of the data that needs to be encrypted. Decryption, on the other hand, has a high overhead per data packet and its performance with small data sizes (64-128 Bs, representative of Hash Table entries) is significantly lower than with larger ones (1-2 KBs, representative of values stored in the KVS). The resource consumption numbers in Table 2 summarize our findings: on a device similar to the one we used, an SDP solution is likely to consume at least around 40% of the logic resources, half of which is used for the AES engines.

The above findings show that the proposed functionality can be fitted on the type of FPGA device we used but many FPGAs deployed in smart SSD solutions contain much less programmable logic. This consideration and the fact that the estimates in this section do not include the cost of RSA or ECDSA cores required for authentication purposes, we conclude that while feasible to

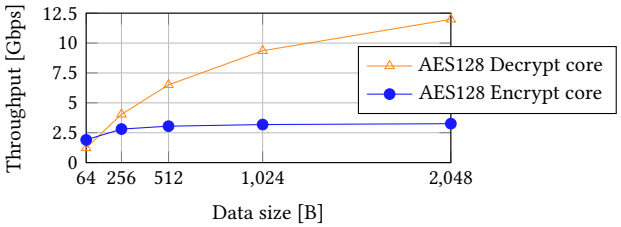

Figure 4: Performance of a AES128 cores running at 250MHz on a Xilinx Ultrascale+ device with various data sizes.

implement SDP on large FPGAs, moving forward, it will be crucial to design SDP solutions around heterogeneous platforms, combining several small CPU cores and programmable hardware in an efficient manner. Authentication, for instance, is a non-performance-critical task that can be delegated to software, freeing up FPGA resources; whereas streaming encryption is a better match for a hardware-only implementation.

5 FUTURE CHALLENGES

Trusted Execution Environments. There are several trust-related challenges in implementing SDP. First of all, the security of the encrypted data at rest hinges on the assumption that the storage device cannot and will not leak cipher keys from DDR memory. Also, the Controller has to be able to trust the storage node once it has been powered on and "booted". This could be ensured by using a Trusted Execution Environment inside the storage node – something that, for the moment, is missing from smart storage targeting the cloud! However, emerging research projects, such as Keystone [27], can implement TEEs with custom hardware components. This approach fits the use-case of SDP well, because a small RISC-V or ARM core could be used to load the firmware on the storage node that the controller provides. This firmware, in turn, can be verified not to be able to read out cipher keys to clients, etc. FPGAs have been also proposed to be used as TEEs in project examples such as Cipherbase [2] where they perform transaction processing in an always-encrypted database management system.

Translating Policies to SDP The question of how company- and application-wide policies are written, managed and translated to SDP rules is future work. There is rich related work [24, 43, 45] which demonstrates how to translate high level policies to compliant queries in databases (or compliant accesses in data storage layers) and there is also recent work that uses SGX enclaves for similar goals [5, 38]. These implementations, however, all differ and we believe that there is value in unifying them on top of SDP. In parallel to implementing a prototype of SDP we are also encouraging collaborations across domains to investigate how to convert rules to an SDP interface.

Data Tracking Beyond Storage. While making sure that the storage layer protects privacy and respects all rules, data misuse can happen at other layers of the application stack as well. There is no guarantee that a buggy or malicious application does not store, for instance, data belonging to one *user* under some other, potentially non-existent, one's data; or that results of processing do not leak

personally identifiable information to the outside world. Countering such behavior has been the subject of numerous studies in the context of Information Flow Control, but practical, general purpose, solutions are still not widely available. Even though the SDP approach does not solve this challenge, we believe that at least it makes it easier: On the one hand, removing all decision making and policy interpretation from the storage nodes and moving it into a logically centralized controller allows for better overview of the system. Other monitoring tools might be used to augment the monitoring capability of the controller, achieving this way better coverage. Furthermore, by not allowing storing new tuples into the storage unless they belong to a *user* and *purpose* known to the controller, some misuse scenarios can be limited and be audited after the fact (identifying, for instance, the application that created non-existent user IDs).

6 CONCLUSION

In this paper we painted our vision *Software-Defined Data Protection (SDP)* for smart storage that separates control path and data path in order to simplify the complexity of in-storage processing necessary for enforcing data protection and privacy rules. This makes high bandwidth, network line-rate, implementation of complex rules and regulations possible without increasing the energy consumption of storage nodes significantly. We describe a high level pipeline model that exposes a simple interface to the controller and can handle even as strict and complex regulations as GDPR. Based on recent advances in FPGA-based key-value stores and near-storage processing, the functionality needed for making SDP a reality could almost be provided by re-purposing existing building blocks. There are, however, open challenges to be tackled: most importantly, SDP requires secure boot and attestation of the software/firmware running on smart storage nodes and this will be only possible if we incorporate TEEs in them. Hence, this paper is a call to arms for security, database and systems researchers to join forces in making privacy protecting distributed smart storage a reality.

ACKNOWLEDGMENTS

Vijay Chidambaram and Soujanya Ponnapalli were partially supported by a grant from VMware. Zsolt István was partially supported by a grant from the Novo Nordisk Foundation. We thank Xilinx for their generous donation of hardware and software used in this work.

REFERENCES

[1] Amazon. 2020. AWS announces AQUA for Amazon Redshift (preview). https://aws.amazon.com/about-aws/whats-new/2020/12/aws-announces-aqua-for-amazon-redshift-preview/.

[2] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. 2015. Transaction processing on confidential data using cipherbase. In *31st International Conference on Data Engineering (ICDE)*. IEEE, 435–446.

[3] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. 173–190.

[4] Wolfgang Bauer, Philipp Holzinger, Marc Reichenbach, Steffen Vaas, Paul Hartke, and Dietmar Fey. 2018. Programmable HSA Accelerators for Zynq UltraScale+ MPSoC Systems. In *European Conference on Parallel Processing*. Springer, 733–744.

[5] Eleanor Birrell, Anders Gjerdrum, Robbert van Renesse, Håvard Johansen, Dag Johansen, and Fred B Schneider. 2018. SGX enforcement of use-based privacy. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. 155–167.

[6] Dan Boneh and Richard J Lipton. 1996. A Revocable Backup System.. In *USENIX Security Symposium*. 91–96.

[7] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. 2016. A cloud-scale acceleration architecture. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[8] CCPA. 2018. California Consumer Privacy Act. *California Civil Code, Section 1798.100* (Jun 28 2018).

[9] Keith Chapman, Mehdi Nik, Behnam Robatmili, Shahrzad Mirkhani, and Maysam Lavasani. 2019. Computational Storage For Big Data Analytics. In *Proceedings of 10th International Workshop on Accelerating Analytics and Data Management Systems (ADMS)*.

[10] Jaeyoung Do, Sudipta Sengupta, and Steven Swanson. 2019. Programmable solid-state storage in future cloud datacenters. *Commun. ACM* 62, 6 (2019), 54–62.

[11] Bob Duncan. 2019. EU General Data Protection Regulation Compliance Challenges for Cloud Users.

[12] GDPR. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46. *Official Journal of the European Union* 59, 1-88 (2016).

[13] Alireza Hodjat and Ingrid Verbauwhede. 2004. A 21.54 Gbits/s fully pipelined AES processor on FPGA. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 308–309.

[14] Zsolt István, Gustavo Alonso, and Ankit Singla. 2018. Providing multi-tenant services with FPGAs: Case study on a key-value store. In *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 119–1195.

[15] Zsolt István, David Sidler, and Gustavo Alonso. 2016. Runtime parameterizable regular expression operators for databases. In *24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 204–211.

[16] Zsolt István, David Sidler, and Gustavo Alonso. 2017. Caribou: intelligent distributed storage. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1202–1213.

[17] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. 2016. Consensus in a box: Inexpensive coordination in hardware. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 425–438.

[18] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Trusted Execution Environments: Properties, Applications, and Challenges. *IEEE Security & Privacy* 18, 2 (2020), 56–60.

[19] Insoon Jo, Duck-Ho Bae, Andre S Yoon, Jeong-Uk Kang, Sangyeun Cho, Daniel DG Lee, and Jaeheon Jeong. 2016. YourSQL: a high-performance database system leveraging in-storage computing. *Proceedings of the VLDB Endowment* 9, 12 (2016), 924–935.

[20] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, and Shuotao Xu. 2016. Bluedbm: Distributed flash storage for big data analytics. *ACM Transactions on Computer Systems (TOCS)* 34, 3 (2016), 1–31.

[21] Miriam Kelly, Eoghan Furey, and Kevin Curran. 2020. How to Achieve Compliance with GDPR Article 17 in a Hybrid Cloud Environment. *Sci* 2, 2 (2020), 22.

[22] Myungsuk Kim, Jisung Park, Genhee Cho, Yoona Kim, Lois Orosa, Onur Mutlu, and Jihong Kim. 2020. Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1311–1326.

[23] Gunjae Koo, Kiran Kumar Matam, I Te, HV Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: trading communication with computing near storage. In *50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 219–231.

[24] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. 2018. Pesos: Policy enhanced secure object store. In *Proceedings of the Thirteenth European Conference on Computer Systems (EuroSys)*. 1–17.

[25] Tim Kraska, Michael Stonebraker, L. Michael Brodie, Sacha Servan-Schreiber, and J. Daniel Weitzner. 2019. SchengenDB - A Data Protection Database Proposal. *Poly/DMAH@VLDB* (2019), 24–38.

[26] Lucas Kuhring, Eva Garcia, and Zsolt István. 2019. Specialize in moderation—building application-aware storage services using FPGAs in the datacenter. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

[27] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: an open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*. 1–16.

[28] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*. 137–152.

[29] Z. S. Li, C. Werner, and N. Ernst. 2019. Continuous Requirements: An Example Using GDPR. In *27th International Requirements Engineering Conference Workshops (REW)*. 144–149.

[30] Ricardo Macedo, João Paulo, José Pereira, and Alysson Bessani. 2020. A Survey and Classification of Software-Defined Storage Systems. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–38.

[31] Miti Mazmudar. 2019. Mitigator: Privacy policy compliance using Intel SGX.

[32] Soumyadeb Mitra and Marianne Winslett. 2006. Secure deletion from inverted indexes on compliance storage. In *Proceedings of the second ACM workshop on Storage security and survivability*. 67–72.

[33] Soujanya Ponnapalli, Aashaka Shah, Amy Tai, Souvik Banerjee, Vijay Chidambaram, Dahlia Malkhi, and Michael Wei. 2019. Scalable and Efficient Data Authentication for Decentralized Systems. *arXiv preprint arXiv:1909.11590* (2019).

[34] Pandian Raju, Soujanya Ponnapalli, Evan Kaminsky, Gilad Oved, Zachary Keener, Vijay Chidambaram, and Ittai Abraham. 2018. mlsm: Making authenticated storage faster in ethereum. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

[35] Erkuden Rios, Eider Iturbe, Xabier Larrucea, Massimiliano Rak, Wissam Mallouli, Jacek Dominiak, Victor Muntés, Peter Matthews, and Luis Gonzalez. 2019. Service level agreement-based GDPR compliance and security assurance in (multi)Cloud-based systems. *IET Software* 13 (2019), 213–222.

[36] Samsung. 2020. Samsung SmartSSD Product Brief. https://www.nimbix.net/wp-content/uploads/2020/02/SmartSSD_ProductBrief_12.pdf.

[37] Malte Schwarzkopf, Eddie Kohler, M Frans Kaashoek, and Robert Morris. 2019. Position: Gdpr compliance by construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 39–53.

[38] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K Rajamani, Janice Tsai, and Jeannette M Wing. 2014. Bootstrapping privacy compliance in big data systems. In *IEEE Symposium on Security and Privacy*. IEEE, 327–342.

[39] Aashaka Shah, Vinay Banakar, Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. 2019. Analyzing the Impact of GDPR on Storage Systems. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

[40] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2020. Understanding and Benchmarking the Impact of GDPR on Database Systems. *Proc. VLDB Endow.* 13, 7 (March 2020), 1064–1077. https://doi.org/10.14778/3384345.3384354

[41] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. 2019. The Seven Sins of Personal-Data Processing Systems under GDPR. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.

[42] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron, Tom Talpey, Richard Black, and Timothy Zhu. 2013. IOFlow: a software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*. 182–196.

[43] Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. 2015. Automatic enforcement of data use policies with datalawyer. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 213–225.

[44] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Rodrigo Rodrigues, Johannes Gehrke, and Ansley Post. 2015. Guardat: Enforcing data policies at the storage layer. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*. 1–16.

[45] Frank Wang, Ronny Ko, and James Mickens. 2019. Riverbed: enforcing user-defined privacy constraints in distributed web services. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 615–630.

[46] Qingbo Zhu and Windsor W Hsu. 2005. Fossilized index: The linchpin of trustworthy non-alterable electronic records. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (Sigmod)*. 395–406.