Big Data SAVE: Secure Anonymous Vault Environment

Omar H Soliman
New Mexico Institute of Mining and Technology
oms@cs.nmt.edu

Abstract

There has been great progress in taming the volume, velocity and variation of Big Data. Its volume creates need for increased storage space and improved data handling. Its velocity is concern for the speed and efficiency of applied algorithms and processes. Its variation requires flexibility to handle assorted data-types. However, as with many emerging fields, security has taken a backseat to benchmarks. This has led to retrofitting traditional security techniques ill-suited for Big Data protection, or high-performance setups exposed to data breach. Proposed is an innovative storage system that can provide large-scale, low-overhead data security, akin to safe-deposit boxes. This approach allows for anonymously-shared storage space, discrete levels of access, plausible deniability, and customizable degrees of overall protection (including warrant-proof). A promising factor of this new model is the use of a simple encryption algorithm (proven faster than industry-standard ciphers), that provides inherent attack resiliency and strong backward secrecy.

1. Introduction

Big Data is characterized as data that exceeds the processing capacity of conventional database systems [12]. The amount of data being produced and collected in the world has reached the order of exabytes; the New York Stock Exchange itself generates more than 1 terabyte of information each session [16]. This growing paradigm of computer science is a challenge to IT administrators tasked with its handling. They face unique situations for managing it, either due to its large volume, high velocity, or pronounced variety [12][17]. The sheer size of Big Data-sets makes it difficult to store and manipulate, since classic data stores were not designed with extreme scalability in mind, nor perform efficiently at extreme volume. Additionally, many algorithms become impractical when dealing with tera/petabyte input. The speed at which these datasets are generated can be overwhelming, and the variation of data types found in Big Data causes many inconveniences when utilizing storage systems designed with a specific data set/type in mind, or those expecting a certain schema when none may exist.

The combined benefits of Big Data continue to provide multiple benefits to consumers [28]. Engineers receive data back from deployed product sensors, and can improve the design of failing parts or preempt future disasters based on that information. Healthcare companies can reduce service cost by analyzing patient data and engaging in preventative care or focusing on areas of greatest need. Big Data gathered by insurance and credit card companies is utilized to reduce incidence of fraud and costs. Personalized service on e-commerce sites is the result of Big Data analysis of consumer behavior. Climate patterns are based off weather models contained in Big Data-sets. The end outcome of Big Data is increased knowledge available to decision-makers, which in turn benefits society through better-informed decisions, resulting in greater efficiency, less cost, and accurate personalization of products and services [6]. By the end of 2018, companies will have spent more than $150 billion on Big Data technologies, to reach almost a quarter trillion dollars by 2020 [18]. Most of these technologies focus on increasing the technical capabilities of conducting analysis on (and deriving knowledge from) Big Data collections.

1.1. Cybersecurity Concerns

Meanwhile, relatively less thought seems to be placed in securing Big Data [35]. In many cases, this data is personal consumer data, and often can be used to identify the consumer or affect their privacy [21]. In regards to companies that do not encrypt data, or do so poorly, customers are faced with possible theft or disclosure of their private details and behavior, resulting in anything from embarrassment to monetary loss. An average cyber breach can cost an organization $15 million per year, with 42% of external costs being

URI: https://hdl.handle.net/10125/60146
ISBN: 978-0-9981331-2-6

directly related to the information stolen [15]. Recent examples include Equifax's disclosure of 143 million customer records, Anthem's loss of 79 million patient records, and T.J.Maxx's exposure of 94 million credit cards [19]. For companies that attempt to secure Big Data, but suffer the consequences of poor performance and re-use of old methodology not intended for this paradigm, Big Data-oriented security solutions are needed, whose cornerstones are fast encryption and key management [33]. Such an efficient solution could help increase the adoption and performance of Big Data security, as well as ensure compliance with emerging privacy laws such as the European Union's General Data Protection Regulation (while avoiding their costly legal penalties for non-compliance). Additionally, swift and tailored Big Data security solutions may grant access to datasets deemed too risky to handle otherwise, allowing further scientific analysis and discovery.

Thus, the problem present in Big Data is a lack of relevant security solutions (fast, scalable, and customizable) that can afford sensitive data privacy for consumers and operational efficiency for companies.

1.2. Cybersecurity Considerations

In cases of multiple entities simultaneously utilizing a single storage platform, their data may share the same physical or virtual storage space. This gives rise to the risk that data may be leaked or intercepted by another user or third party [26][34]. Although encryption can protect data contents, obfuscating the data location/owner relationship can add defense-in-depth to the system. This can be accomplished by distributing data non-contiguously in the store and providing a discrete access mechanism for users.

In addition, storage of data in a third-party environment exposes it to elements beyond the originator's control [26][34]. Data can be shared with unauthorized parties or even mined by the service itself. Furthermore, the vendor may be compelled to produce customer data in the case of a warrant or subpoena, leading to a loss of privacy [29]. Thus, plausible deniability is a requirement for ensuring privacy in such an environment, which can be accomplished via careful key management and a warrant-proof encryption scheme. Here, warrant-proof refers to the trait that even if presented with a valid warrant, the system is inherently unable to usefully comply, and can only, at most, provide a complete copy of the entire datastore (in its encrypted form).

Finally, not every use-case requires the same level of access control. There is a tradeoff between certain security aspects (e.g. authentication/auditing vs. anonymity), which users may prioritize differently. This makes customizability of a secure storage scheme an important trait for encouraging its adoption among a broader userbase.

1.3. Remaining Sections

In this paper, we describe our implementation of a novel and secure Big Data storage-architecture. Section 2 reviews relevant work in securing Big Data. Section 3 describes our solution approach. Section 4 details our solution architecture. Section 5 provides core implementation. Section 6 discusses applications. Section 7 concludes with final remarks.

2. Related Work

Reference [20] introduces the use of survivable routing and Redundant Array of Inexpensive Disks (RAID) for providing integrity and confidential Big Data storage in network file systems (when combined with broader schemes such as Hadoop). The authors argue that the common security tactics of preventative measures, attack detection, and then recovery are inadequate to secure such large cyber assets. Instead, they focused on being intrinsically attack tolerant by eliminating single points of failure, through the use of fragmentation and eventual reassembly of stored data. They also argue that incorporating intrusion tolerance within a Big Data security architecture is preferable to attempting complete prevention. Compelling points of the paper are the need to inherently expect and tolerate attacks against a Big Data storage system. Also of note is the idea of data dispersal (wherein their data is not stored contiguously in the system, thereby increasing the difficulty of its malicious retrieval), as well as reducing the consequence of a single failure against the system.

Reference [9] presents a distributed anonymous storage service ("Free Haven"), which focuses on data availability for a guaranteed period of time (versus absolute persistence). It is based on a trust network which involves file sharing. A community of servers (nodes) compromises a "servnet". Each node holds portions of documents; these fragments are called "shares". Each node is assigned a "pseudonym", which enables the tracking of server reputation. Reputation is used as incentive for nodes to store shares during a contractual length of time; fulfilling storage contracts raises a node's reputation, and allows it to store more of its own data on other nodes. Shares can also be traded, which allows nodes to fluidly move in and out of the servnet. Additionally, shares have a hard expiration date, and can be dropped at the end of their holding contracts. The greatest feature of this service in

a Big Data context is the provision of multiple kinds of anonymity (that of the authors, publishers, readers, servers, documents, and queries). Drawbacks include the use of relatively slow public/private encryption, reliance on anonymous networking and volunteer computing, and large overhead per transaction. In regards to the Big Data paradigm, data velocity is of concern in such a system, which relies on broadcast networking to store and retrieve data.

Reference [8] describes "Freenet", implemented as a peer-to-peer network that pools drive space anonymously, acting as a location-independent distributed file system. Nodes query each other to publish and retrieve data files (referred to by keys/hashes). Each node maintains a portion of local data storage for the network, as well as a dynamic routing table of other nodes and which files they likely hold. Users of the system are encouraged to run nodes, both to decrease odds against unintentionally accessing a hostile node, and to increase the overall storage capacity available to the network. Freenet grants anonymity to both producers and consumers of information, plausible deniability for holders of information, and is resistant to denial-of-service attacks. However, it assumes a guaranteed secure transport layer, and does not guarantee data permanence (a major issue when the goal is to securely store data for long periods of time). Additionally, it incurs copious overhead during hashing, public/private encryption, and decentralized routing.

Reference [3] deliberates the importance of Big Data, specifically in regards to reconciling security with privacy. Aside from traditional privacy techniques that are not all suitable for Big Data, the authors discuss progress in privacy-preserving data analytics, such as data matching, mining, and biometric authentication. A big issue with these techniques is their lack of scaling at the Big Data level, inefficiency, or immaturity. Specifically, the common theme of these techniques is the secure sharing of Big Data-sets, without compromising information to malicious actors, while also limiting specific details between collaborators. In order to gauge the suitability of solutions to varying Big Data scenarios, they propose a multi-objective optimization framework for data privacy (using cost/utility/risk). However, it is limited by the need to objectively deduce acceptable levels for those parameters. Lastly, they detail issues surrounding Big Data confidentiality (in terms of access control and encryption) as well as possible privacy-violating correlation between datasets.

Reference [36] discusses Big Data security in light of cloud computing. The authors present a computational cloud model in order to evaluate Big Data use cases. They survey three specific security techniques (homomorphic encryption, verifiable computation, and multi-party computation), in specific cloud-trust contexts (un/semi/trusted), using their model to evaluate confidentiality and integrity of simulated data. Because of Big Data's inherent business-related nature and value, a recurring theme in this Big Data paper and many others focuses on maintaining confidentiality throughout computation, through the use of homomorphic and/or functional encryption schemes [7][32][25]. The above approaches emphasize manipulating data-at-rest in their encrypted state, without focusing on the performance penalty of achieving said encryption. Faster encryption designs could negate some of the need for homomorphic operations (since the cost to decrypt/operate/re-encrypt would be relatively negligible).

Many companies also offer closed-source Big Data security products for a fee. CertainSafe offers a Digital Safety Deposit Box [27]. A two-part key is used to encrypt consumer data (one held by the consumer, the other by the company). "MicroEncryption" takes form in fragmenting stored data across multiple drives and servers, in order to minimize the effects of a single-server compromise. However, neither the encryption algorithms nor their throughput was mentioned. Also, the server is not truly keyless since not only does it possess a key, it handles the consumer's key after authenticating and identifying them. Dropbox stores data using 256-bit Advanced Encryption Standard (AES) and Secure Sockets Layer/Transport Layer Security (SSL/TLS) tunnels for transfer (no keys are held by the user) [11]. Box stores data using 256-bit AES and TLS tunnels for transfer [5]. Additionally, they use an "encryption key wrapping strategy" that also utilizes 256-bit AES. No keys are held by the user. Box KeySafe encrypts data with a BoxKey [4]. The BoxKey is then encrypted using the user-defined key. However, a permanent audit log is kept to record key usage, and no explanation of how the BoxKey is generated was given, nor any guarantee it cannot be recreated. SpiderOakONE claims zero knowledge of stored data [31]. Data is encrypted before reaching their servers, where it is stored in generalized blocks. However, data is encrypted using layered 2048-bit Rivest–Shamir–Adleman (RSA) and 256-bit AES keys. Those keys are encrypted using a 256-bit AES key generated through salted Password-Based Key Derivation Function (PBKDF)/Secure Hash Algorithm (SHA)-256. The overhead from those operations is not conducive at the Big Data scale. All these closed-source solutions fail to simultaneously provide broad anonymity, low-overhead, high-performance, tamper detection, shared storage, zero-server-knowledge, and customized access control.

3. Approach

Many large technology companies currently offer cloud services, including data storage solutions. The biggest of these service providers include Google, Microsoft, and Amazon [10]. Reference [14] details how security is designed into Google's technical infrastructure. Reference [1] discusses Amazon's approach to security in their web services. Reference [22] provides a comprehensive look at the security features of Microsoft's cloud system. Common security features implemented in these aforementioned services include access control, encryption, and logging.

To provide access control and encryption in a high-performance security solution, an appropriate encryption method needs to be selected. Synchronous Dynamic Encryption System (SDES) [30] is a lightweight encryption protocol which utilizes symmetric keying and provides a streaming block cipher mechanism. Its main advantages over a standard such as AES are its very high speed (approximately 4 times faster software benchmark vs. hardware-accelerated AES) and the involvement of the plaintext in the one-way rekeying process. This allows for fault tolerance from a mid-stream record compromise – previous records do not become any easier to decrypt as a result (i.e. backward secrecy). Additionally, this provides an inherent integrity mechanism, since key generation will falter if any records are improperly altered.

SDES's low memory footprint and overhead is attributable to its simple scheme. A vector of {0-255} is shuffled using an initial management key. The shuffled vector is now the current encryption key. The plaintext record is applied bitwise exclusive-or (XOR) with the encryption key to generate ciphertext. The plaintext record, management key, and encryption key are then combined to form the new management key. The new management key is used to shuffle the old encryption key. The shuffled, old encryption key is now the next encryption key. In total, each byte per round of encryption utilizes 2 additions, 3 assignments, and 1 XOR operation. The runtime of SDES is O(m*n) (the number of records and record size in bytes). The cryptanalysis complexity for guessing the key is $\Omega(2^n)$ (regarding the bit-length of the key). These attributes give SDES fast execution, and therefore better scaling with larger datasets.

Augmenting the use of encryption is anonymity. When utilizing shared data space with others, anonymity affords multiple protections. Three types of anonymity are of concern. User anonymity refers to the concealment of user identities from others (including the server). This allows organizations utilizing co-located systems a lower profile and more protection from targeted attacks. Data anonymity refers to the concealment of data from everyone but the originating user. This allows for plausible deniability in regards to hosting specific content, as well as prevention of content identification. Crowd anonymity refers to the veiling of data amongst other data such that the locations of related data are known only to the originating user. This is especially useful in Big Data since there is a large quantity of data intrinsically present in the environment, which can inherently aid in crowd anonymity.

Combining these two components, a parallel can be drawn between safe deposit boxes and encryption with anonymity. Banks provide a secure location for the storage of valuables in an anonymous manner. Generally, a client can rent a safe deposit box with the bank, bring in some assets, safely store them inside an anonymous box, receive a key and box number, and can then retrieve the valuables at a later time. The bank's vault contains many of these boxes that serve many customers, with no indication of their contents nor owners. The bank (in theory) cannot access the secured contents alone, without the help of the owner. In some cases, the bank can provide monitoring service of box openings, or provide privacy when they are being accessed.

This approach to securing Big Data is to emulate the workings of a bank's safe deposit box system for use as an electronic data store. In classic storage models, data and keys belong to the server, which decrypts and streams data to the user after successful authentication. However, this requires data segregation, user-to-data mapping, and trust in the server. Implementing user anonymity (limit authentication), data anonymity (prevent server from reading data in a compromise by not storing keys), and crowd anonymity (no mapping of user to data) equalizes external and internal threats to the system. By utilizing SDES, fallout from a successful compromise is limited to a fraction of what it otherwise would be. Hence, the Secure Anonymous Vault Environment (SAVE) is proposed.

An implementation of SAVE is hosted on a storage system which stows client data by splitting it into chunks, encrypting each in series using SDES, and fragmenting them throughout a series of safe boxes (records). These are simply indexed locations in the storage space, which the system cannot reassemble without the initial key and location, known only to the originating user. Data is reconstructed by unlocking the first record and using its contents to locate the next, and so on. In effect, users can securely store data anonymously in an attack tolerant manner.

4. SAVE Architecture

Reference [2] is a security framework for well-architected design according to Amazon's Web Services best practices. The two most applicable pillars from that framework in the context of SAVE are security and performance efficiency. Design principles listed for strengthening security include: strong identities, traceability, defense-in-depth, automation, protecting data in-transit/rest, segregating people from data, and anticipating security events.

In order to implement these principles, SAVE consists of a few basic components: the user, initial key, incoming authentication, audit logs, front-end server, and storage system (Fig. 1). In context of the prior design principles: data protection and strong identities are enforced through encryption; traceability is provided through logging; defense-in-depth is present through multi-level security options; and automation, data/person segregation, and security event anticipation are afforded by the system design itself.

The nature of SAVE leads to many scheme variations and a high level of customizability for meeting situational requirements. Detailed variations and components of prospective SAVE setups are discussed and summarized in Table 1.

4.1. Authentication

Authentication is performed when clients initially contact the SAVE host. Its purpose is to potentially identify incoming connections and reduce malicious traffic. Three authentication levels are presented. Free Authentication: no authentication is performed on incoming connections to the SAVE host. Clients are free to send and receive read-transactions. This allows for fast, unsecured communication, but does nothing to prevent malicious requests. In this case, user anonymity is fully maintained as clients are not identified to any server. However, brute force attacks may be carried out against encrypted records since attackers may freely access the same record repeatedly. This could also lead to high incoming traffic from such requests. Host Authentication: incoming connections must first authenticate with the SAVE host, which then establishes a long-term session with the client through some mechanisms (e.g. cookies). This method is the least anonymous since the server must be aware of the client's identity. Although it provides a standard level of security, performance will be impacted from the overhead. Token Authentication: incoming connections must first authenticate with a token server, which issues the client a token upon success. It also makes available to the SAVE host the token information through a secure channel. When establishing communication, the client can then use the token as a shared secret with the host (e.g. to utilize SDES as a lightweight protocol). However, user anonymity is reduced, since a token server is aware of their identity (although not the SAVE host). Communication speed is also impacted depending on protocol choice.

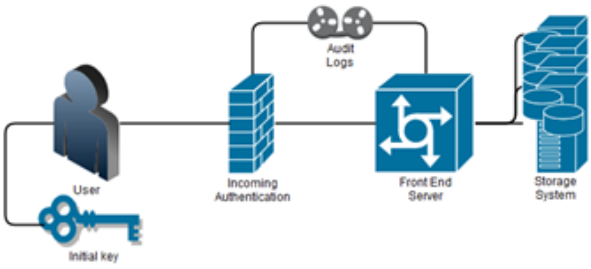

Figure 1. Diagram of SAVE architecture

In addition to authentication, the system must also decide how to deal with access control violations. Possible actions corresponding to the given authentication methods are mentioned below. Free Response: clients which repeatedly request the same record with different keys (indicative of brute force) are blocked. Depending on the service scheme, clients which access numerous or certain records may also be blocked. Host Response: clients which repeatedly fail authentication are blocked. Depending on the other components, they may also be warned of suspicious actions rather than immediately be blocked. Token Response: clients which repeatedly fail authentication are blocked. Clients which spoof tokens are also blocked. Depending on the other components, they may additionally be warned of suspicious actions rather than be immediately blocked.

4.2. Logging

Logging is performed whenever the client communicates with the SAVE system. Its purpose is to provide notation of record accesses. No Logging: no client details are logged by any hosts. This provides high user anonymity, but at the cost of no auditing capability. Basic Logging: request timestamp and desired record index are logged. User anonymity is still maintained, but record accesses can be referenced later. Full Logging: request timestamp, IP address, and desired record index are logged. User anonymity is severely reduced, but record accesses can be referenced in greater detail later. Blockchains could also be used to maintain a ledger of record accesses.

In addition to logging, the system can also include active auditing processes that can check for suspicious behaviors. Possibilities are listed below. No Auditing: audits are not performed on any logged data. This is useful in instances where logs are not kept. Temporal

Auditing: audits compare record indices with timestamps in attempt to detect irregular timed accesses. This is suitable for use with basic logging. Full Auditing: audits compare IP address and timestamp vs. record index in order to better detect irregular record access. This is suitable for use with full logging. Moreover, artificial intelligence (e.g. Markov chains) can be used to aid in the detection of irregular behavior.

If an audit discovers suspicious activity, the following are potential action items. Basic Detection: the incident is logged and administrator notified. Since a goal of SAVE is anonymity, it may not be possible to notify the record owner. IP Reaction: if IP addresses are being logged, then the offending client can be blocked via firewall. Record Panic: an extreme reaction by the SAVE system could be to wipe records being accessed maliciously. However, this may not be very effective if the data had already been accessed (unless it was to prevent multi-client brute force attacks in progress). If the SAVE host can derive related records to the ones being attacked, some of those can be wiped to limit a successful compromise in its tracks (although this would reduce crowd anonymity in the system).

4.3. Client Request

When clients request records from the SAVE host, they must supply certain data for a successful response. Varying request structures are described. Key & Initial Record Index: clients supply an alleged management key for decrypting the requested record and expect the SAVE host to decrypt the related records in series. If the request is legitimate, the SAVE host can do so without problem. However, if it is a malicious request, then locating related records from the incorrectly decrypted content and choosing what to return depend on other system settings. This method is also relatively inefficient if a specific record is ultimately desired from the middle of larger dataset (since all prior related records must first be decrypted). Key & Record Index: clients supply an alleged management key for decrypting the requested record. This differs from the previous scheme in that a record from the middle of a dataset can be accessed directly (although the client must store additional keys for every such access point). Key & Record Index & Total: clients supply an alleged management key for decrypting the requested record,

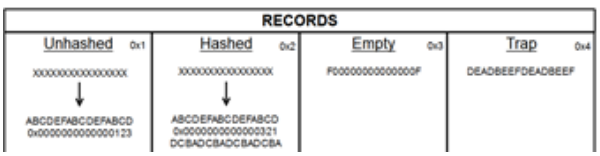

Figure 2. Example of record contents

along with the total number of records they wish to retrieve from the dataset. This allows the SAVE server to better deal with malicious requests (since it can return an appropriate amount of spoofed data matching the requested record count) and potentially improve its resource allocation when prioritizing responses (many small taskings vs. one sizeable request).

4.4. Server Response

When a client presents the SAVE host a request, an appropriate response will be returned depending on the chosen system scheme. Full Response: when using the client request scheme with total records requested, the SAVE host responds by accessing and decrypting the initial record, then attempts to decrypt the given number of records from the series. If the decryption fails due to malicious request, the server returns an appropriate amount of noise to mimic a successful access (in an attempt to not aid the attacker). Coded Response: the SAVE host responds by accessing and decrypting the first record and any that are linked if the given key is correct. Otherwise, it stops when a decryption fails or an inaccurate total of records is requested (dependent on record content structure). Single Response: the SAVE server responds with only a single record access and attempted decryption. This is useful for high volumes of granular accesses (since it is quickly locating and decrypting a specific record) or parallel request responsiveness (multiple-users would each get a small amount of data in succession, rather than waiting to reach the front of a queue).

4.5. Record Structure

There are many possible schemes (Fig. 2) for record contents (which can support other system components). Un-hashed Content: records contain data and the index of the next record in the dataset (in encrypted form). After decryption, the SAVE host attempts to update the current key and access the next record. This maintains crowd anonymity (since the location of related records is not known until decryption), but the host has no reliable way of determining if an attempted decryption was successful or not. Hashed Content: records contain data and the index of the next record in the dataset along with a hash (in encrypted form). After decryption, the SAVE server attempts to generate the sequential encryption key and access the next record after ensuring the hash is valid. This maintains crowd anonymity (since the location of related records is not known until decryption), but the server has a method of determining if an attempted decryption was successful or not.

Empty: if the SAVE server maintains a list of record indices currently in use, then the ones not in use are essentially empty. This reduces crowd anonymity since the server is effectively aware of which records are currently occupied. This is, however, useful when assigning new data to record locations, and detecting malicious requests (since empty records should naturally go unrequested). Traps: the SAVE server can strategically place decoy records within the vault in order to better detect malicious requests or act as honeypots for attackers. They can range from plaintext that is returned as if successfully decrypted or used to immediately trigger an audit response.

4.6. Record Allocation

There are many possibilities for sizing the SAVE storage space. Single Uniform Records: the storage space is uniformly divided into records of equal size and indexed sequentially. This maintains crowd anonymity since the records are all identical, and makes addressing them relatively simple. Multi Uniform Records: the storage space is uniformly divided into records of equal size which are grouped into uniform size boxes. This somewhat reduces crowd anonymity, but also reduces the overhead of successive record lookups (depending on the scaling of the box sizes). Dynamic Sizing: rather than pre-allocating the storage space, the host provides on-demand, custom-size storage for datasets. This is detrimental to crowd anonymity since records are of varying features (size) and therefore differentiable (to an insider). A scheme would also need to be implemented in order to address the data structures.

4.7. Submission

Storing data in the SAVE system is dependent on many of the factors discussed above. Generally, this is a stricter operation than reading since the SAVE host may be subject to denial of service by resource-consuming writes. The SAVE host must also be able to track record usage (either by an occupied list or dynamic storage) in order to safely locate unused records for depositing data into storage. Free Submission: in high-trust situations, clients may submit a key and dataset to be encrypted and stored in the vault, and are returned an initial record number by the SAVE host. If the user wishes to edit already stored data, a record number must also be provided (which may require decryption and re-encryption using the newly supplied data for all downstream records). This presents a high risk of malicious writes; however, it maintains high user anonymity. Additionally, an insecure line leaves the data and key vulnerable to interception (public keying may be useful). Registered Submission: storing data requires first authenticating in some manner with the system (in order to alleviate malicious writes, and in the case of paid systems, to be charged). Afterwards, encrypted datasets may be deposited into the vault for safekeeping, with initial record numbers returned to the clients for future retrieval. This reduces user anonymity since submissions must come from registered sources.

5. Core Implementation

To initially implement SAVE, its core encryption functionality was encoded in the C programming language as a Linux kernel module, so that it could be utilized via the Linux CryptoAPI [24]. Electronic Code Book (ECB) keying was used to wrap SDES as a block cipher. Ideally, stand-alone SDES keying would wrap an XOR cipher; however, using ECB allows for more direct comparison vs. AES performance at this early level. Basic Application Programming Interface (API) calls include decrypting/encrypting a block, and setting the cipher key. The API also comes with built-in AES functionality, useful for benchmarking purposes.

In order to access CryptoAPI calls from user-space, the libkcapi library was employed [23]. This was used to run benchmark simulations to compare CryptoAPI performance of SDES vs. AES. SDES operates on 256-byte blocks (otherwise a modulus operation must also be used to prevent out-of-bounds addressing). AES operated on 16-byte blocks. The algorithms were fed multi-gigabyte plaintext, with their performance compared to each other, as well identical file operations without added security functionality (as an overhead baseline). Specifically, 1-10 gigabyte data files were encrypted and decrypted using each cipher suite, along with plain access of the data (by record),

Authentication	Violations	Logging	Auditing	Requests	Responses	Structure	Allocation	Submission
Free	Brute	None	None	Key/Initial	Noise/Data	Unhashed	Uniform	Blind
Host	Trap/Empty	Basic	Temporal	Key/Index	Data	Hashed	Grouped	Editing
Token	Spoofing	Full	Full	Key/Index/Total	Single record	Empty/Trap	Dynamic	Authenticated

Table 1. SDES feature summary

without any other features active. This provided a relative comparison of the SAVE engine runtime versus runtime performance of AES encryption (traditionally employed in commercial products).

Results of this basic benchmark indicate that SDES exhibited a range of 8-15% overhead penalty vs. baseline data access and handling. Matching AES operations incurred a 100-190% overhead penalty. The actual overhead for each scheme varied steadily with each other, and established SDES as consistently 12 times faster (on average) than AES, which is often used as the industry standard for encryption, and commonly found in commercial Big Data security solutions on the market. These initial results demonstrate the feasibility of SAVE achieving multifold performance improvement over existing security offerings. Further simulation-benchmark results follow.

5.1. Authentication

Overhead from authentication is incurred by having to prove identity to the host and/or other server. This involves the issuance of a token/cookie, and the verification of it with each request. Free authentication incurs no penalty, since nothing is performed. Host authentication requires identity proof via cookie encryption/decryption. Token authentication is similar to host authentication, but is done via a dedicated server in order to maximize anonymity to the storage system. To simulate usage, a 256-byte cookie is both encrypted/decrypted each time a record is also both encrypted/decrypted from a gigabyte dataset. The average of these runs is presented below.

Authentication Type	Average Overhead
Host/Token	30%
AES with None	150%

Table 2. Authentication benchmark

AES without any authentication functionality was 5 times slower than SAVE running with authentication enabled.

5.2. Logging

Logging overhead occurs from storing log data, as well as auditing logs for malicious behavior. Simulation was performed by encrypting/decrypting a gigabyte of records and appending log data to a file. Simple auditing additionally involved scanning the logfile to search for a known-malicious indicator of attack.

Logging Type	Average Overhead
Full	15%
Simple Auditing	16%

Table 3. Logging benchmark

Full logging did not significantly differ from baseline overhead. Naïve auditing did add a minor penalty, although advanced auditing (e.g. machine learning) would increase overhead more so.

5.3. Verification

It is possible for SAVE to simply return record contents, or to first verify decryption success using a hash stored in the encrypted record. To measure performance penalty, a gigabyte of records was encrypted/decrypted first without verification, then again followed by content verification. Verification overhead itself is heavily tied to the choice of algorithm used. For demonstration purposes, Fletcher's Checksum was utilized [13].

Record Content	Average Overhead
Unverified	15%
Verified	17%
AES with None	190%

Table 4. Record verification benchmark

Even with check-summation, SAVE was 11 times faster than AES with no verification at all.

5.4. Multi-Feature

With authentication, logging, auditing, and verification enabled, the relative overhead of a simple SAVE configuration was simulated across a gigabyte of records.

Features	Average Overhead
Multiple	31%
AES with None	150%

Table 5. Multi-feature benchmark

When compared to just AES computation, a basic SAVE configuration offering much more functionality clocked-in at 5 times faster.

6. Applications

There are a wide variety of applications where SAVE can provide benefit with its use. Several examples are noted. High throughput sharing of

sensitive data is possible using SAVE as a secure storage escrow, especially if implemented via in-memory database or through a similar paradigm. Public data stores can be set up and secured using SAVE as an anonymous information repository, where multiple clients can store and retrieve keyed records. Organizations can setup a SAVE system within their network to protect against insider threats, reduce data leakage, and maintain a competitive edge. SAVE can potentially be adapted for use atop high performance-focused Big Data stores (e.g. Hadoop) as a security layer, to facilitate the processing of sensitive data. Finally, SAVE can be used to statically store Big Data until data analysis or mining operations are performed.

7. Conclusion

Although much effort has been put into improving the performance of Big Data tools and benchmarks, relatively less has been placed in securing Big Data. This is exacerbated by many organizations lacking the funds necessary to host Big Data systems themselves and instead turning to third party services. This can potentially result in sensitive data leaking as the result of a breach, and thus discourages the handling of such data (and waste of its potential benefit). By utilizing SAVE, service providers can offer anonymous and secure storage of Big Data, even when shared among multiple clients. The high level of SAVE's customizability should be useful for tailoring it to the unique requirements of various applications and data demands. Overall, SAVE attempts to address high velocity, volume, and variety attributes by providing a high-throughput storage scheme that can be combined with other Big Data tools to provide security with minimized overhead.

Future work includes increased integration within the Linux CryptoAPI, full implementation as a Linux filesystem, construction of a stand-alone deployment, and creation of Hadoop and MongoDB plugins.

8. References

[1] Amazon, "Introduction to AWS security", awsstatic.com, July 2015

[2] Amazon, "Security Pillar AWS Well-Architected Framework", awsstatic.com, July 2018

[3] Bertino et al., "Big Data – Security with Privacy", NSF Big Data Security and Privacy Workshop, October 2014

[4] Box, "Box Keysafe Manage Your Own Encryption Keys", box.com, May 2018

[5] Box, "Box Security Protect the Flow Of Information", box.com, May 2018

[6] Brown et al., "Are You Ready for The Era of 'Big Data'?", McKinsey Quarterly, McKinsey Global Institute, October 2011

[7] N. Chakraborty and G. Patra, "Functional Encryption for Secured Big Data Analytics", IJCA, December 2014

[8] Clarke et al., "Freenet: A Distributed Anonymous Information Storage and Retrieval System", International Workshop On Designing Privacy Enhancing Technologies: Design Issues In Anonymity And Unobservability, 2001

[9] Dingledine et al., "The Free Haven Project: Distributed Anonymous Storage Service", In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, 2000

[10] DoubleHorn, "Comparing the Big 3: Aws, Azure, and Google Cloud", doublehorn.com, June 2018

[11] DropBox, "Under the Hood: Architecture Overview", dropbox.com, May 2018

[12] E. Dumbill, "What Is Big Data?", Strata, O'Reilly, January 2012

[13] J.G. Fletcher, "An Arithmetic Checksum for Serial Transmissions", IEEE Transactions on Communications, January 1982

[14] Google, "Google Infrastructure Security Design Overview", google.com, January 2017.

[15] HP, "Annual Study Reveals Average Cost of Cyber Crime Per Organization Escalates to $15 Million", HP Inc. Press Release, October 2015

[16] IBM, "The Four V's of Big Data", IBM Big Data & Analytics Hub, 2013

[17] IBM, "Top Tips for Big Data Security", IBM Corporation, 2015

[18] IDC, "Big Data and Business Analytics Revenues Forecast to Reach $150.8 Billion This Year, Led by Banking and Manufacturing Investments, According to IDC", Worldwide Semiannual Big Data and Analytics Spending Guide, March 2017

[19] IOG Communications, "The 17 Biggest Data Breaches of the 21st Century", csoonline.com, January 2018

[20] B. Madan and Y. Lu, "Attack Tolerant Big Data File System", ACM SIGMETRICS, June 2013

[21] D. Meyer, “Why the Collision of Big Data and Privacy Will Require a New Realpolitik”, Gigaom, March 2013

[22] Microsoft, “Introduction to Azure Security”, microsoft.com, November 2017

[23] S. Mueller, “libkcapi -- Linux Kernel Crypto API User Space Interface Library”, GitHub, March 2018

[24] S. Mueller and M. Vasut, “Linux Kernel Crypto API”, The Linux Kernel, December 2016

[25] Popa et al., “Cryptdb: Processing Queries on an Encrypted Database”, Proceedings of the 23rd ACM Symposium on Operating Systems Principles, October 2011

[26] M. Prinzlau, “6 Security Risks of Enterprises Using Cloud Storage and File Sharing Apps”, DataInsider, DigitalGuardian, March 2016

[27] N. Rubenking, “Certainsafe Digital Safety Deposit Box”, PC Mag, June 2016

[28] Schroeck et al., “Analytics: The Real-World Use of Big Data”, IBM Institute for Business Value, October 2012

[29] J. Silver, “Lavabit Held in Contempt of Court for Printing Crypto Key in Tiny Font”, arstechnica, April 2014

[30] H. Soliman and M. Omari, “New Design Strategy of Dynamic Security Implementation”, IEEE GlobeCom 2004, December 2004, pp. 442-446

[31] SpiderOak, “Protect What Matters Most”, spideroak.com, May 2018

[32] Stephen et al., “Practical Confidentiality Preserving Big Data Analysis”, USENIX, June 2014

[33] C. Tankard, “Encryption as the Cornerstone of Big Data Security”, Network Security, Elsevier, March 2017, pp. 5-7

[34] University of Texas at Austin, “Risks of Cloud Services”, Information Security Office, November 2015

[35] P. Wood, “How to Tackle Big Data from a Security Point of View”, ComputerWeekly.com, March 2013

[36] Yakoubov et al., “A Survey of Cryptographic Approaches to Securing Big-Data Analytics in the Cloud”, IEEE, 2014