Big Data

Security and Privacy Handbook:

100 Best Practices in Big Data Security and Privacy

Presented by Big Data Working Group

CSA cloud security alliance®

The permanent and official location for *Cloud Security Alliance Big Data research* is

https://cloudsecurityalliance.org/group/big-data/

Table of Contents

Acknowledgements

Editors

Daisuke Mashima
Sreeranga P. Rajan

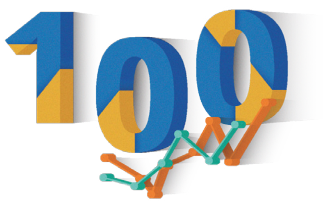
Contributors

Alvaro A. Cárdenas
Yu Chen
Adam Fuchs
Wilco Van Ginkel
Janne Haldesten
Dan Hiestand
Srinivas Jaini
Adrian Lane
Rongxing Lu
Pratyusa K. Manadhata
Jesus Molina
Praveen Murthy
Arnab Roy
Shiju Sathyadevan
P. Subra Subrahmanyam
Neel Sundaresan

CSA Global Staff

Ryan Bergsma
Frank Guanco
JR Santos
John Yeoh

Introduction



The term "big data" refers to the massive amounts of digital information companies and governments collect about human beings and our environment. The amount of data generated is expected to double every two years, from 2500 exabytes in 2012 to 40,000 exabytes in 2020. Security and privacy issues are magnified by the volume, variety, and velocity of big data. Large-scale cloud infrastructures, diversity of data sources and formats, the streaming nature of data acquisition and high-volume, inter-cloud migration all play a role in the creation of unique security vulnerabilities.

It is not merely the existence of large amounts of data that creates new security challenges. In reality, big data has been collected and utilized for several decades. The current uses of big data are novel because organizations of all sizes now have access to the information and the means to employ it. In the past, big data was limited to very large users such as governments and more sizeable enterprises that could afford to create and own the infrastructure necessary for hosting and mining large amounts of information. These infrastructures were typically proprietary and isolated from general networks. Today, big data is cheaply and easily accessible to organizations large and small through public cloud infrastructure. Software infrastructures such as Hadoop enable developers to easily leverage thousands of computing nodes to perform data-parallel computing. Combined with the ability to buy computing power on-demand from public cloud providers, such developments greatly accelerate the adoption of big data mining methodologies. As a result, new security challenges have arisen from the coupling of big data with public cloud environments, characterized by heterogeneous compositions of commodity hardware with commodity operating systems, as well as commodity software infrastructures for storing and computing on data.

As big data expands through streaming cloud technology, traditional security mechanisms tailored to secure small-scale, static data on firewalled and semi-isolated networks are inadequate. For example, analytics for anomaly detection would generate too many outliers. Similarly, it is unclear how to retrofit provenance in existing cloud infrastructures. Streaming data demands ultra-fast response times from security and privacy solutions.

This Cloud Security Alliance (CSA) document lists out, in detail, the best practices that should be followed by big data service providers to fortify their infrastructures. In each section, CSA presents 10 considerations for each of the top 10 major challenges in big data security and privacy. In total, this listing provides the reader with a roster of 100 best practices. Each section is structured as follows:

- What is the best practice?
- Why should it be followed? (i.e. what is the security/privacy threat thwarted by following the best practice?)
- How can the best practice be implemented?

This document is based on the risks and threats outlined in the Expanded Top Ten Big Data Security and Privacy Challenges.

1.0 Secure Computations in Distributed Programming Frameworks

In distributed programming frameworks such as Apache Hadoop, it is important to ensure trustworthiness of mapper and secure data in spite of untrusted mappers. Also, it is necessary to prevent information leakage from mapper output. Hence, the following guidelines should be followed to ensure secure computations in distributed programming frameworks.

1.1 Establish initial trust

1.1.1 Why?

To ensure trustworthiness of mappers.

1.1.2 How?

Establish initial trust by making master authenticate worker using Kerberos authentication or equivalent when worker sends connection request to master. The authentication should be mutual to ensure authenticity of masters. Besides authentication, use of integrity measurement mechanisms, i.e. one using Trusted Platform Module (TPM), should be considered.

1.2 Ensure conformance with predefined security policies

1.2.1 Why?

To achieve a high level of security in computations.

1.2.2 How?

Periodically check security properties of each worker. For example, the master nodes Hadoop-policy.xml should check for a match with the worker nodes security policy.

1.3 De-identify data

1.3.1 Why?

To prevent the identity of the data subject from being linked with external data. Such linking may compromise the subjects' privacy.

1.3.2 How?

All personally identifiable information (PII), such as name, address, social security number, etc., must be either masked or removed from data. In addition to PII, attention should also be given to the presence of quasi-identifiers, which include data items that can almost uniquely identify a data subject (e.g., zip code, date of birth, and gender). Technologies such as k-anonymity [Swe02] should be applied to reduce re-identification risks.

1.4 Authorize access to files with predefined security policy

1.4.1 Why?

To ensure integrity of inputs to the mapper.

1.4.2 How?

Mandatory access control (MAC).

1.5 Ensure that untrusted code does not leak information via system resources

1.5.1 Why?

To ensure privacy.

1.5.2 How?

Mandatory access control (MAC). Use Sentry for HBASE security using RBAC (role-based access controls). In Apache Hadoop, the block access token is configured to ensure that only authorized users are able to access the data blocks stored in data nodes.

1.6 Prevent information leakage through output

1.6.1 Why?

To ensure security and privacy. Data leakage may occur in many ways, which needs to be prevented (i.e. improper use of encryption). Debugging messages, uncontrolled output streams, logging functions and detailed error pages help attackers learn about the system and formulate attack plans.

1.6.2 How?

• Use function sensitivity to prevent information leakage.
• Shadow execution (i.e. communication towards external networks to obtain software version updates) is another aspect that needs to be taken into consideration.
• Additionally, all data should be filtered on the network level (in-transit), in line with data loss prevention policies.
• Sufficient de-identification of data also contributes to mitigation of the impact.

1.7 Maintain worker nodes

1.7.1 Why?

To ensure proper functionality of worker nodes.

1.7.2 How?

Frequently check for malfunctioning worker nodes and repair them. Ensure they are configured correctly.

1.8 Detect fake nodes

1.8.1 Why?

To avoid attacks in cloud and virtual environments.

1.8.2 How?

Build a framework to detect fake nodes introduced by creating snapshots of legitimate nodes.

1.9 Protect mappers

1.9.1 Why?

To avoid generating incorrect aggregate outputs.

1.9.2 How?

Detect the mappers returning wrong results due to malicious modifications.

1.10 Check for altered copies of data

1.10.1 Why?

To avoid attacks in cloud and virtual environments.

1.10.2 How?

Detect for the data nodes that are re-introducing the altered copies and check such nodes for their legitimacy. Hashing mechanism and cell timestamps of cell data will enforce integrity.

2.0 Security Best Practices for Non-Relational Data Stores

Non-relational data stores such as NoSQL databases typically have very few robust security aspects embedded in them. Solutions to NoSQL injection attacks are not yet completely mature. With these limitations in mind, the following suggestions are the best techniques to incorporate while considering security aspects for non-relational data stores.

2.1 Protect Passwords

2.1.1 Why?

To ensure privacy.

2.1.2 How?

- By encryption or hashing using secure hashing algorithms.
- Use cryptographic hashing algorithms functions such as SHA2 (SHA-256 or higher) and SHA3.
- When hashing, use salt to counter offline, brute-force attacks.

2.2 Safeguard data by data encryption while at rest

2.2.1 Why?

To reliably protect data in spite of weak authentication and authorization techniques applied.

2.2.2 How?

Use strong encryption methods such as the Advanced Encryption Standard (AES), RSA, and Secure Hash Algorithm 2 (SHA-256). The storage of code and encryption keys must be separate from the data storage or repository. The encryption keys should be backed up in an offline, secured location.

2.3 Use transport layer security (TLS) to establish connections and communication

2.3.1 Why?

To maintain confidentiality while in transit; to establish trusted connections between the user and server; and to securely establish communication across participating cluster nodes.

2.3.2 How?

Implement TLS/SSL (secure sockets layer) encapsulated connections. Ideally, each node is equipped with a unique public/private key pair and digital certificate so that client authentication is enabled.

2.4 Provide support for pluggable authentication modules

2.4.1 Why?

To certify users are able to program to pluggable authentication module (PAM) interface by using PAM library API for authentication-related services.

2.4.2 How?

Implement support for PAM. Hardening with benchmarks established by the Center for Internet Security and hardening at the operating system (OS) level (e.g., SELinux) can be considered.

2.5 Implement appropriate logging mechanisms

2.5.1 Why?

To expose possible attacks.

2.5.2 How?

- Implement logging mechanisms according to industry standards, such as the NIST Log

Management Guide SP800-92 [KS06] and ISO27002 [ISO05].
• Use advanced persistent threat (APT) logging mechanisms like log4j, etc. For example, ELK Stack (Elasticsearch, Logstash, Kibana) and Splunk can be used for log monitoring and on-the-fly log analysis.

2.6 Apply fuzzing methods for security testing

2.6.1 Why?

To expose possible vulnerabilities caused by insufficient input validation in NoSQL that engages hypertext transfer protocol (HTTP) to establish communication with users (e.g., cross-site scripting and injection).

2.6.2 How?

• Provide invalid, unexpected or random inputs and test for them. Typical strategies include dumb fuzzing, which uses completely random input, and smart fuzzing, which crafts input data based on knowledge about the input format, etc.
• Guidelines are provided by the Open Web Application Security Project (OWASP) (https://www.owasp.org/index.php/Fuzzing), MWR InfoSecurity (https://www.mwrinfosecurity.com/our-thinking/15-minute-guide-to-fuzzing/), etc. Fuzzing should be done at separate levels in a system, including the protocol level, data node level, application level, and so forth.
• Use tools for fuzzing, such as Sulley.

2.7 Ensure appropriate data-tagging techniques

2.7.1 Why?

To avoid unauthorized modification of data while piping data from its source.

2.7.2 How?

Use security-tagging techniques that mark every tuple arriving on a specified data source with a special, immutable security field including timestamp.

2.8 Control communication across cluster

2.8.1 Why?

To ensure a secure channel.

2.8.2 How?

Ensure each node validates the trust level of other participating nodes before establishing a trusted communication channel.

2.9 Ensure data replication consistency

2.9.1 Why?

To handle node failures correctly.

2.9.2 How?

Use intelligent hashing algorithms and ensure that the replicated data is consistent across the nodes, even during node failure.

2.10 Utilize middleware layer for security to encapsulate underlying NoSQL stratum

2.10.1 Why?

To have a virtual secure layer.

2.10.2 How?

By inducing object-level security at the collection, or column-level through the middleware, retaining its thin database layer.

3.0 Secure Data Storage and Transactions Logs

Security is needed in big data storage management because solutions, such as auto tiering, do not record the storage place of data. The following practices should be implemented to avoid security threats.

3.1 Implement exchange of signed message digests

3.1.1 Why?

To address potential disputes.

3.1.2 How?

- Use common message digests (SHA-2 or stronger) to provide digital identifier for each digital file or document, which is then digitally signed by the sender for non-repudiation.
- Use the same message digest for identical documents.
- Use distinct message digests even if the document is partially altered.

3.2 Ensure periodic audit of chain hash or persistent authenticated dictionary (PAD)

3.2.1 Why?

To solve user freshness and update serializability issues.

3.2.2 How?

Use techniques such as red-black tree and skip lists data structures to implement PAD [AGT01].

3.3 Employ SUNDR

3.3.1 Why?

To store data securely on untrusted servers

3.3.2 How?

Use SUNDR (secure untrusted data repository) to detect any attempts at unauthorized file modification by malicious server operators or users. It is also effective to detect integrity or consistency failures in visible file modifications using fork consistency.

3.4 Use broadcast encryption

3.4.1 Why?

To improve scalability.

3.4.2 How?

Use broadcast encryption scheme [FN 93] in which a broadcaster encrypts a message for some subset S of users who are listening on a broadcast channel. Any user in S can use a private key to decrypt the broadcast. However, even if all users outside of S collude, they can obtain no information about the content of the broadcast. Such systems are said to be collusion resistant. The broadcaster can encrypt to any subset S of his choice. It may still be possible that some members of S may contribute to piracy by constructing a pirate decoder using private keys assigned to them. To ascertain the identities of such malicious members—and thereby discourage piracy—traitor-tracing mechanisms should be implemented as well.

3.5 Apply lazy revocation and key rotation

3.5.1 Why?

To improve scalability.

3.5.2 How?

• Use lazy revocation (i.e. delay re-encryption until a file is updated in order to make revocation operation less expensive).
• To implement lazy revocation, generate a new filegroup for all the files that are modified following a revocation and then move files to this new filegroup as files get re-encrypted. This process raises two issues, as stated below:

> ***Issue:*** There is an increase in the number of keys in the system following each revocation.
> ***Solution:*** Relate the keys of the filegroups that are involved.
>
> ***Issue:*** Because file sets that are re-encrypted following successive revocations are not really contained within each other, it becomes increasingly difficult to determine which filegroup a file should be assigned to when it is re-encrypted.
> ***Solution:*** Use key rotation. Set up the keys so that files are always (re)encrypted with the keys of the latest filegroup. This ensures that users are required to remember only the latest keys and derive previous ones when necessary.

3.6 Implement proof of retrievability (POR) or provable data possession (PDP) methods with high probability

3.6.1 Why?

To enable a user to reliably verify that data uploaded to the cloud is actually available and intact, without requiring expensive communication overhead

3.6.2 How?

Ateniese et al. [ABC+07] introduced a model for provable data possession (PDP) that allows a user that has stored data at an untrusted server to verify that the server possesses the original data without retrieving it. The model generates probabilistic proof of possession by sampling random sets of blocks from the server, which drastically increases efficiency. The user maintains a constant amount of metadata to verify the proof. The challenge/response protocol transmits a small, constant amount of data, which minimizes network communication.

Kaliski and Juels [JK07] developed a somewhat different cryptographic building block known as a proof of retrievability (POR). A POR enables a user to determine whether it can “retrieve” a file from the cloud. More precisely, a successfully executed POR assures a verifier that the prover presents a protocol interface through which the verifier can retrieve the given file in its entirety. Of course, a prover can refuse to release the file even after successfully participating in a POR. A POR, however, provides the strongest possible assurance of file retrievability barring changes in prover behavior.

3.7 Utilize policy-based encryption system (PBES7)

3.7.1 Why?

To avoid collusion attacks (assuming users do not exchange their private keys).

3.7.2 How?

• Allow user to encrypt a message with respect to a credential-based policy formalized as monotone Boolean expression written in standard, normal form.
• Provide encryption so that only a user having access to a qualified set of credentials for the policy is able to successfully decrypt the message.

3.8 Implement mediated decryption system

3.8.1 Why?

To avoid collusion attacks (assuming users are willing to exchange private keys without exchanging decrypted content).

3.8.2 How?

A mediated RSA cryptographic method and system is provided in which a sender encrypts a message (m) using an encryption exponent e and a public modulus n, and a recipient and a trusted authority cooperate with each other to decrypt the encrypted message by using respective components dU, dT of a decryption exponent. In order to prevent the trusted authority from reading the message in the event that it has access to the recipient decryption exponent components dU, the recipient blinds the encrypted message before passing it to the trusted authority. This blinding is affected by a modulo-n blinding operation using a factor r<e> where r is a secret random number. The trusted authority then applies its decryption exponent component dT to the message and returns the result to the recipient who cancels the blinding and applies its decryption exponent component dU to recover the message.

3.9 Use digital rights management

3.9.1 Why?

To counter collusion attacks where users are willing to exchange decrypted contents when access control is implemented by means of encryption

3.9.2 How?

Digital rights management (DRM) schemes are various access control technologies that are used to restrict usage of copyrighted works. Such a scheme is also effective to control access to protected data in a distributed system environment. To prevent unauthorized access to protected information, some frameworks restrict options for accessing content. For instance, the protected content can only be opened on a specific device or with particular viewer software, where access rights and/or policies are securely enforced (perhaps at the hardware level). Moreover, the integrity of such software or devices can be attested by a cloud storage provider, etc., by means of remote attestation techniques and/or TPM (Trusted Platform Module) when necessary.

3.10 Build secure cloud storage on top of untrusted infrastructure

3.10.1 Why?

To store information in a confidential, integrity-protected way—even with untrusted cloud service providers—while retaining service availability, reliability and the ability for efficient data retrieval and flexible data sharing

3.10.2 How?

One solution for implementing a secure cloud storage system is called cryptographic cloud storage [KL10]. This storage technique employs symmetric encryption, searchable encryption [BW07, CJJ+13], attribute-based encryption [SW05], and proof of storage (namely proof of retrievability [JK07] and provable data possession [ABC+07]). Using cryptographic cloud storage, a data owner can store encrypted data while keeping encryption keys locally. Moreover, data integrity can be verified efficiently at any time. Thereby, it can address major challenges, including: regulatory compliance; geographic restrictions; subpoenas; security breaches; electronic discovery; and data retention and destruction.

4.0 Endpoint Input Validation/Filtering

Users should ensure that the source of data is not malicious and—if it is—should filter malicious input materials generated by that source. This challenge becomes more severe with the utilization of the “bring your own device” (BYOD) model. The following are recommended practices to achieve the best-possible input validation/filtering results.

4.1 Use trusted certificates

4.1.1 Why?

To ensure trust in communication and prevent Sybil attacks (i.e. a single entity masquerading as multiple identities).

4.1.2 How?

The digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to trust that signatures or assertions made by the private key (that correspond to the public key) are certified. In this model, a certificate authority (CA) is a trusted third party that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes. There exist several open-source implementations of certificate authority software. Common to all is that they provide the necessary services to issue, revoke and manage digital certificates. Some open-source implementations are DogTag, EJBCA, gnoMint, OpenCA, OpenSSL, r509, and XCA. Validity of certificates must be verified before usage based on a periodically issued certificate revocation list (CRL) or via OCSP (Online Certificate Status Protocol). If a central authority ensures that a unique certificate is assigned to each entity in a system, then an attacker cannot fake multiple identities. A trusted certificate is the only reliable method to defend against Sybil attacks.

4.2 Do resource testing

4.2.1 Why?

To avoid the drawback of managing certificates in a large enterprise but still achieve a minimal defense against Sybil attacks instead of preventing them.

4.2.2 How?

Resource testing [Dou02] is a commonly implemented solution to averting Sybil attacks. It assumes that the computing resources of each node are limited. A verifier then checks whether each identity has as many resources as the single physical device it is associated with [BS12].

Determine if multiple fake identities possess fewer resources than expected from independent genuine identities. Examples of resources are computing capability, storage capability, and network bandwidth.

4.3 Use statistical similarity detection techniques and outlier detection techniques

4.3.1 Why?

To detect and filter out malicious input.

4.3.2 How?

• Generate models that represent "normal" behavior (e.g., a Gaussian curve, and then detect outliers that deviate from normal input, or entities that largely deviate from the Gaussian curve).
• Use a model-based approach, proximity-based approach, and an angle-based approach to detect and filter out malicious input.

4.4 Detect and filter malicious inputs at central collection system

4.4.1 Why?

To block malicious input data without requiring extra computation in resource-constrained endpoint devices.

4.4.2 How?

• Generate models that represent "normal" behavior (e.g., a Gaussian curve, and then detect outliers that deviate from normal input, or entities that largely deviate from the Gaussian curve).
• Use a model-based approach, proximity-based approach, and an angle-based approach to detect and filter out malicious input.

4.5 Secure the system against Sybil attacks

4.5.1 Why?

To detect and prevent one entity from masquerading as multiple identities in a system.

4.5.2 How?

- Trusted certificates (Section 4.1)
- Trusted devices (Section 4.7)
- Resource testing (Section 4.2)

4.6 Identify plausible ID spoofing attacks on the system

4.6.1 Why?

To detect and prevent an attacker from assuming legitimate identities.

4.6.2 How?

- Trusted certificates (Section 4.1)
- Trusted devices (Section 4.7)
- Resource testing (Section 4.2)

4.7 Employ trusted devices

4.7.1 Why?

To detect and prevent Sybil attacks and to prevent the compromise of endpoint devices and applications running on them.

4.7.2 How?

Every entity in a system is assigned to an endpoint device with an embedded unique device identifier, which is tied to a user identity in a 1-to-1 manner (e.g., a secure device identity defined in IEEE 802.1AR). Then an attacker cannot create multiple identities using a single device and the cost of acquiring multiple devices will be prohibitive.

4.8 Design parameter inspectors to examine incoming parameters

4.8.1 Why?

To detect and filter out malicious inputs.

4.8.2 How?

Every data collection system has to implement its own inspector that checks for required properties and formats in the incoming parameters.

4.9 Incorporate tools to manage endpoint devices

4.9.1 Why?

To prevent an attacker from compromising endpoint devices and applications running on the devices.

4.9.2 How?

We can use Tools, such as TPMs to ensure the integrity of devices and applications, should be utilized. Protection mechanisms such as access control and antivirus products and/or host-based intrusion detection systems on endpoint devices should also be used. Information flow control and mandatory access control within each device is also important for device management. Logging and monitoring tools to detect compromises can also be employed.

4.10 Use antivirus and malware protection systems at endpoints

4.10.1 Why?

To prevent an attacker from compromising endpoint devices and applications running on the devices.

4.10.2 How?

Install antivirus and malware protection systems at endpoints. Users should keep the antivirus signature database up-to-date.

5.0 Real-Time Security/Compliance Monitoring

Big data is generated by a variety of different gadgets and sensors, including security devices. Real-time security and compliance monitoring is a double-edged sword. On one hand, big data infrastructures have to be monitored from a security point of view. Questions—is the infrastructure still secure? are we under attack?—need to be answered. On the other hand, entities that utilize big data can provide better security analytics compared to those who do not (e.g., less false positives, more fine-grained and better quantified security overviews, etc.). The following practices should be implemented to adhere to best practices for real-time security/compliance monitoring.

5.1 Apply big data analytics to detect anomalous connections to cluster

5.1.1 Why?

To ensure only authorized connections are allowed on a cluster, as this makes up part of the trusted big data environment.

5.1.2 How?

Use solutions like TLS/SSL, Kerberos, Secure European System for Applications in a Multi-Vendor Environment (SESAME), Internet protocol security (IPsec), or secure shell (SSH) to establish trusted connections to and–if needed–within a cluster to prevent unauthorized connections. Use monitoring tools, like a security information and event management (SIEM) solution, to monitor anomalous connections. This could be, for instance, based on connection behavior (e.g., seeing a connection from a 'bad Internet neighborhood') or alerts being filed in the logs of the cluster systems, indicating an attempt to establish an unauthorized connection.

5.2 Mine logging events

5.2.1 Why?

To ensure that the big data infrastructure remains compliant with the assigned risk acceptance profile of the infrastructure.

5.2.2 How?

• Mine the events in log files to monitor for security, like in a SIEM tool.
• Apply other algorithms or principles to mine events (such as machine learning) to get potential new security insights.

5.3 Implement front-end systems

5.3.1 Why?

To parse requests and stop bad requests. Front-end systems are not new to security. Examples are routers, application-level firewalls and database-access firewalls. These systems typically parse the request (based on, for instance, syntax signatures or behavior profiles) and stop bad requests. The same principle can be used to focus on application or data requests in a big data infrastructure environment (e.g., MapReduce messages).

5.3.2 How?

Deploy multi-stage levels of front-end systems. For example, utilize a router for the network; an application-level firewall to allow/block applications; and a dedicated big data front-end system to analyze typical big data inquiries (like Hadoop requests). Additional technology, such a software defined network (SDN), may be helpful for implementation and deployment.

5.4 Consider cloud-level security

5.4.1 Why?

To avoid becoming the "Achilles heel" of the big data infrastructure stack. Big data deployments are moving to the cloud. If such a deployment lives on a public cloud, this cloud becomes part of the big data infrastructure stack.

5.4.2 How?

• Download "CSA Guidance for Critical Areas of Focus in Cloud Computing V3.0"
• Implement other CSA best practices.
• Encourage Cloud Service Providers to become CSA STAR-certified compliant.

5.5 Utilize cluster-level security

5.5.1 Why?

To ensure that security methodology for big data infrastructure is approached from multiple levels. Different components make up this infrastructure—the cluster being one of them.

5.5.2 How?

Apply—where applicable—best security practices for the cluster. These include:
• Use Kerberos or SESAME in a Hadoop cluster for authentication.
• Secure the Hadoop distributed file system (HDFS) using file and directory permissions.
• Utilize access control lists for access (e.g., role-based, attribute-based).
• Apply information flow control using mandatory access control.

The implementation of security controls also (heavily) depends on the cluster distribution being used. In case of strict security requirements (e.g., high confidentiality of the data being used), consider looking at solutions like Sqrrl, which provide fine-grained access control at the cell level.

5.6 Apply application-level security

5.6.1 Why?

To secure applications in the infrastructure stack. Over the last years, attackers have shifted their focus from operating systems to databases to applications.

5.6.2 How?

• Apply secure software development best practices, like OWASP (owasp.org) for Web-based applications.
• Execute vulnerability assessments and application penetration tests on the application on an ongoing and scheduled basis.

5.7 Adhere to laws and regulations

5.7.1 Why?

To avoid legal issues when collecting and managing data. Due to laws and regulations that exist worldwide—specifically those that relate to privacy rights—individuals who gather data cannot monitor or use every data item collected. While many regulations are in-place to protect consumers, they also create a variety of challenges in the universe of big data collection that will hopefully be resolved over time.

5.7.2 How?

Follow the laws and regulations (i.e. privacy laws) for each step in the data lifecycle. *These include:*
• Collection of data
• Storage of data
• Transmission of data
• Use of data
• Destruction of data

5.8 Reflect on ethical considerations

5.8.1 Why?

To address both technical and ethical questions that may arise. The fact that one has Big Data doesn't necessarily mean that one can just use that data. There is always a fine line between (1) technically possible; and (2) what is ethically correct. The latter is also impacted and related to legal regulations and the organization's culture, among other factors.

5.8.2 How?

There are no clear guidelines concerning ethical considerations related to big data usage. At minimum, big data users must take into account all applicable privacy and legal regulations. Additionally, users should consider ethical discussions related to their organizations, regions, businesses, and so forth.

5.9 Monitor evasion attacks

5.9.1 Why?

To avoid potential system attacks and/or unauthorized access. Evasion attacks are meant to circumvent big data infrastructure security measures and avoid detection. It is important to minimize these occurrences as much as possible.

5.9.2 How?

As evasion attacks evolve constantly, it is not always easy to stop them. Following the implementation of a defense in-depth concept, consider applying different monitor algorithms (like machine learning) to mine the data. Look for insights related to potential evasion of monitoring besides signature-based/rule-based/anomaly-based/specification-based detection schemes.

5.10 Track data-poisoning attacks

5.10.1 Why?

To prevent monitoring systems from being misled, crashing, misbehaving or providing misinterpreted data due to malformed data. These type of attacks are aimed at falsifying data, letting the monitoring system believe nothing is wrong.

5.10.2 How?

• Consider applying front-end systems and behavioral methods to perform input validation, process the data, and determine right from wrong as much as possible.
• It is also crucial to authenticate sources of data and maintain logs not only for preventing unauthorized data injection but also for establishing accountability.
• Utilize the monitoring system for strange behavior, like a spike in the central processing unit (CPU) and memory load for prolonged periods of time, or disk space running full quickly.

6.0 Scalable and Composable Privacy-Preserving Analytics

Studies show that anonymizing data for analytics is insufficient for ensuring user privacy. Below are the best techniques to ensure privacy in a big data environment.

6.1 Implement differential privacy

6.1.1 Why?

To protect privacy even when data is linked with external data sources. Anonymizing public records have failed when researchers manage to identify personal information by linking two or more separately innocuous databases

6.1.2 How?

Differential privacy [Dwo06] aims to provide a means to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records. Differential privacy is the mathematical concept to measure how much (or how little) anonymity is preserved on a database. For example, adding random noise is a method to achieve some level of differential privacy. Users are encouraged to use the appropriate mechanism for a given use.

6.2 Implement Utilize homomorphic encryption

6.2.1 Why?

To enable encrypted data to be stored and processed on the cloud. Data stored in plaintext on the cloud may be compromised and cause privacy risks. On the other hand, when only encrypted data is stored on the cloud, utility of data is significantly limited.

6.2.2 How?

Homomorphic encryption is a form of encryption that allows specific types of computations to be carried out on ciphertext. The method allows users to obtain an encrypted result that, when decrypted, matches the result of operations performed on the plaintext. Users should utilize techniques such as unpadded RSA for implementing partially homomorphic cryptosystems.

6.3 Maintain software infrastructure

6.3.1 Why?

To avoid exploitation of improperly maintained software, a major vulnerability.

6.3.2 How?

Maintain software infrastructure patched with up-to-date security solutions.

6.4 Use separation of duty principle

6.4.1 Why?

To provide robust internal control as well as information security. The separation of duty principle—coupled with the enforcement of the principle of least privilege—provides both attributes.

6.4.2 How?

Implement security controls which enforce strict separation of duties so that each operator has access to a specific set of minimal data and is only able to perform a specified set of actions on that data. Institute auditing of user actions on the system. To enforce reliable separation, access to shared resources should be carefully monitored or controlled to detect and/or block covert channels.

6.5 Be aware of re-identification techniques

6.5.1 Why?

To protect the privacy interests of consumers. Re-identification is the process by which anonymized personal data is matched with its true owner. Personal identifiers—such as names and social security numbers—are often removed from databases containing sensitive information. However, re-identification compromises consumer privacy.

6.5.2 How?

• Anonymized (or de-identified) data safeguards the privacy of consumers while still making useful information available to marketers or data-mining companies.
• Establish a formal standard for privacy which addresses possible re-identification methods.

6.6 Incorporate awareness training with focus on privacy regulations

6.6.1 Why?

To avoid potential litigation issues into the future. There are an increasing number of laws and regulations that require training and awareness activities related to privacy issues (e.g., the Health Insurance Portability and Accountability Act (HIPPA) and Health Information Technology for Economic and Clinical Health Act (HITECH) in the U.S., etc.). Awareness of these laws and regulations is critical.

6.6.2 How?

Implement awareness training focused on privacy issues and applicable regulations in each country.

6.7 Use authorization mechanisms

6.7.1 Why?

To secure applications in the infrastructure stack. Over the last years, attackers have shifted their focus from operating systems to databases to applications.

6.7.2 How?

• Apply secure software development best practices, like OWASP (owasp.org) for Web-based applications.
• Execute vulnerability assessments and application penetration tests on the application on an ongoing and scheduled basis.

6.8 Encrypt data at rest

6.8.1 Why?

To prevent access to sensitive information. Threats against end user devices may allow unauthorized parties to access personal information. To prevent such inappropriate disclosures, particularly of personally identifiable information (PII) and other sensitive data, the confidentiality of data needs to be secured on the devices.

6.8.2 How?

The primary security control for restricting access to sensitive information stored on end-user devices is encryption. Encryption can be applied granularly, such as to an individual file containing sensitive information, or broadly, such as encrypting all stored data. In the case of database infrastructure, primary keys are used for indexing and joining tables. Therefore, encryption may not be applicable. Sensitive data, such as personally identifiable information, should not be used as a primary key. Ensure that the encryption algorithm used is current and appropriate for the given data set.

6.9 Implement privacy-preserving data composition

6.9.1 Why?

To address privacy concerns preemptively. In some real-world circumstances (such as those that may occur in the healthcare industry), it is often necessary to aggregate and/or query data from multiple data sources, such as electronic healthcare record systems in multiple hospitals or research institutes. Privacy issues are likely to emerge during that process.

6.9.2 How?

Ensure that leakage of private information is controlled when multiple databases and/or services are linked by reviewing and monitoring the functionality that links them.

6.10 Design and implement linking anonymized datastores

6.10.1 Why?

To ensure privacy. Even if data in each datastore is anonymized (i.e. personally identifiable information is appropriately removed), this is not often sufficient if multiple datastores are linked. For example, more than 80 of American citizens can be uniquely identified using a combination of birthdate, gender, and zip code.

6.10.2 How?

For each datastore, implement privacy concepts like k-anonymity [Swe02], t-closeness [LLV07] and/or l-diversity [MKG07] as well as differential privacy.

7.0 Cryptographic Technologies for Big Data

The advent of big data has heralded a revolution of sophisticated new techniques in cryptography to address the security of massive, streaming and increasingly private data. There is a realization across the industry that cryptographic technologies are imperative for cloud storage and big data. Mathematical assurance of trust gives people more incentive to migrate data and computations to the cloud. Rather than burdensome requirements, there is an increased perception that cryptographic technologies are harbingers of trusted utility for impending advances in information technology. This section will highlight a few of the exciting new research directions that the cryptography community is beginning to explore, as well as best practices for cryptographic technologies for big data.

7.1 Construct system to search, filter for encrypted data

7.1.1 Why?

To balance data confidentiality and data utility. Consider a system to receive e-mails encrypted under the owner's public key. It's likely the owner does not want to receive spam. With plain public key encryption, there is no way to distinguish a legitimate e-mail ciphertext from a spam ciphertext. In this way, encryption often lowers effectiveness of information security technologies as well as usability of data.

7.1.2 How?

Boneh and Waters [BW07] construct a public key system that supports comparison queries, subset queries and arbitrary conjunction of such queries. In a recent paper [CJJ+13], Cash et al., present the design, analysis and implementation of the first sub-linear searchable symmetric encryption (SSE) protocol that supports conjunctive search and general Boolean queries on symmetrically encrypted data. The protocol scales to very large data sets and arbitrarily structured data, including free text search.

7.2 Secure outsourcing of computation using fully homomorphic encryption

7.2.1 Why?

To enable outsourcing of computation while addressing security and privacy concerns. Consider a user who wants to send all sensitive data to a cloud: photos, medical records, financial records and so on. The user could send everything encrypted, but this wouldn't be much use if they wanted the cloud to perform various computations on them, such as how much money was spent on movies in the past month.

7.2.2 How?

In a breakthrough result [Gen09] in 2009, Gentry constructed the first fully homomorphic encryption scheme. Such a scheme allows users to compute the encryption of arbitrary functions of the underlying plaintext. Earlier results [BGN05] constructed partially homomorphic encryption schemes. Gentry's original construction of a fully homomorphic encryption (FHE) scheme used ideal lattices over a polynomial ring. Although lattice constructions are not terribly inefficient, the computational overhead for FHE is still far from practical. Research is ongoing to find simpler constructions [vDGHV10, CMNT11], efficiency improvements [GHS12b, GHS12a] and partially homomorphic schemes [NLV11].

7.3 Limit features of homomorphic encryption for practical implementation

7.3.1 Why?

To balance computational cost and versatility when handling encrypted data. Although fully homomorphic encryption is an ideal solution in terms of versatility, the computation cost is still too high to be practical.

7.3.2 How?

By limiting features of homomorphic encryption (e.g., limiting only to additive homomorphic operations or to certain types of fundamental statistical computations, such as inner product) the practicality of homomorphic encryption schemes dramatically improve while retaining real-world applicability.

7.4 Apply relational encryption to enable comparison of encrypted data

7.4.1 Why?

To enable efficient comparison of encrypted data without sharing encryption keys. Typically, each organization protects its data by using its own encryption key. It is often needed (e.g., for healthcare research, or to correlate or link data stored in different organizations). Such an operation could be done by using homomorphic encryption, but it requires all organizations to use the same encryption key. In addition, its computation is very costly.

7.4.2 How?

Relational encryption [MR15] technology enables the matching of IDs, attribute values, etc., among data encrypted with different keys. Thus, each data owner can use different keys to protect sensitive information. Moreover, the entity performing such mapping or linking operations—for instance a cloud service provider—cannot decrypt the data. Therefore, confidentiality is preserved.

7.5 Reconcile authentication and anonymity

7.5.1 Why?

To balance security and privacy. Often the requirements of authentication and anonymity are antithetic to each other. However, it is sometimes possible to achieve a middle ground where authentication can be guaranteed despite preserving some degree of anonymity. For cryptographic protocol ensuring the integrity of data coming from an identified source, the core requirement is that the adversary should not be able to forge data that did not come from the purported source. However, there can also be some degree of anonymity in the sense that the source is only identifiable as being part of a group. In addition, in certain situations (depending upon regulations that may be in place), a trusted third party should be able to link the data to the exact source.

7.5.2 How?

A group signature is a cryptographic scheme that enables individual entities to sign their data but remain identifiable only in a group to the public. Only a trusted third party can pinpoint the identity of the individual. The scheme was first proposed by Chaum and Heyst [Cv91], with practical instantiations developed by Boneh, Boyen and Shacham [BBS04].

Ring signatures, first formalized by Rivest, Shamir and Tauman [RST01], are group signature schemes which have only users and no managers. Group signatures are useful when the members want to cooperate, while ring signatures are useful when the members do not want to cooperate. Both group signatures and ring signatures are signer-ambiguous, but in a ring signature scheme there are no pre-arranged groups of users; no procedures for setting, changing, or deleting groups; no way to distribute specialized keys; and no way to revoke the anonymity of the actual signer. The only assumption is that each member is already associated with the public key of some standard signature scheme. To produce a ring signature, the actual signer declares an arbitrary set of possible signers that includes himself, and computes the signature entirely by himself using only his secret key and the others public keys.

7.6 Implement identity-based encryption

7.6.1 Why?

To overcome difficulties associated with key management of a public-key crypto system. One of the major difficulties when practically deploying a system that relies on public-key cryptography is the management of keys, including provisioning, updates, and revocation. For instance, all communicating nodes must be equipped with digital certificates issued by trusted certification authorities. Moreover, sufficient key management is not feasible in a setting like the Internet of hings (IoT), where a large number of resource-constrained devices are involved.

7.6.2 How?

In identity-based systems [Sha85] (IBE), plaintext can be encrypted for a given identity and the expectation is that only an entity with that identity can decrypt the ciphertext. Any other entity will be unable to decipher the plaintext, even with collusion. Boneh and Franklin [BF01] came up with the first IBE using pairing-friendly elliptic curves. Since then, there have been numerous efficiency and security improvements [Wat09, CW13, JR13].

7.7 Utilize attribute-based encryption and access control

7.7.1 Why?

To integrate access control and encryption in a practical manner. Traditionally, access control to data has been enforced by systems—including operating systems and virtual machines—which restrict access to data, based on some access policy. The data is still in plaintext. There are at least two problems to the systems' paradigm: (1) systems can be hacked; and (2) security of the same data in transit is a separate concern.

7.7.2 How?

Attribute-based encryption (ABE) extends this concept to attribute-based access control. In [SW05], Sahai and Waters presented the first ABE, in which a user's credentials are represented by a set of string called 'attributes' and the access control predicate is represented by a formula over these attributes. Subsequent work [GPSW06] expanded the expressiveness of the predicates and proposed two complementary forms of ABE. In key-policy ABE, attributes are used to annotate the ciphertexts and formulas over these attributes are ascribed to users' secret keys. In ciphertext-policy ABE, the attributes describe the user's credentials and the formulas over these credentials are attached to the ciphertext by the encrypting party. The first work to explicitly address the problem of ciphertext-policy attribute-based encryption was by Bethencourt, Sahai, and Waters [BSW07], with subsequent improvements by Waters [Wat11].

7.8 Use oblivious RAM for privacy preservation

7.8.1 Why?

To prevent information leakage that may occur through access pattern analysis implemented by cloud providers. When data is stored in a cloud, the access pattern to the data—which is visible to cloud service providers—may leak sensitive, private information even if the data is appropriately encrypted.

7.8.2 How?

Oblivious RAM [SSS11] shuffles memory locations after each access. Thus, even a cloud service provider cannot tell which data is accessed; therefore, the access pattern can be effectively hidden.

7.9 Incorporate privacy-preserving public auditing

7.9.1 Why?

To enable public auditing without causing privacy concerns. It is a trend to outsource verification procedures to a third-party auditor (TPA), and the verification protocol is expected to be publicly verifiable. Such an operation should not compromise privacy.

7.9.2 How?

A privacy-preserving, public auditing scheme was proposed for cloud storage in [WWRL10]. Based on a homomorphic linear authenticator integrated with random masking, the proposed scheme is able to preserve data privacy when a TPA audits the data set stored in the servers at different tiers.

7.10 Consider convergent encryption for deduplication

7.10.1 Why?

To improve efficiency of storage usage. Data stored on a cloud is typically encrypted. However, using common encryption scheme, even the same file results in different ciphertext. Since the cloud service provider cannot tell whether they are actually the identical data or not, there may be a situation where duplicated copies of the same data may unnecessarily remain on the cloud.

7.10.2 How?

If deduplication is desired, convergent encryption scheme—which was originally proposed in [SGLM08]—can be considered. It uses an encryption key that is deterministically derived from the plaintext data to be encrypted (i.e. cryptographic hash value of the data, and thereby the resulting ciphertext becomes identical). This way, deduplication of the identical data is made possible.

8.0 Granular Access Control

There are two sides to any access control solution. The first is restricting data and operations from users who should not have access, and the second is granting access to users who should have access. Picking the right access control strategy can have a profound impact on how effectively users can leverage a database. To satisfy policy restrictions, coarse-grained access mechanisms often must restrict data that could otherwise be shared. Granular access control mechanisms are a tool that can be used to reduce data restriction without violating policies. The following best practices should be followed while ensuring granular access control.

8.1 Choose appropriate level of granularity required

8.1.1 Why?

To balance complexity and granularity of access control. The use of fine-grained access controls requires an increased complexity in data labeling and security attribute management, while coarse-grained access controls demand data modeling. For example, database views can be used to protect databases that do not support row-, column-, or cell-level access controls, but users must then maintain the views.

8.1.2 How?

Oblivious RAM [SSS11] shuffles memory locations after each access. Thus, even a cloud service provider cannot tell which data is accessed; therefore, the access pattern can be effectively hidden.

8.2 Normalize mutable elements, denormalize immutable elements

8.2.1 Why?

To design suitable access control mechanisms. Recent advances in database technology have opened the door to more forms of denormalized data modeling. For data elements that are more immutable, denormalized models can provide higher concurrency and lower latency than models that require more joins. Granular access controls become even more important when using denormalized data models, since data from many sources and of many types are thrown together in a single bucket.

8.2.2 How?

The core of fine-grained access control is to maintain labels with data. When denormalizing data, maintain provenance information for any provenance elements that are referenced in the data access policy. For example, if the source of data affects who can see that data, then maintain source information in tags along with fields that came from that source.

8.3 Track secrecy requirements

8.3.1 Why?

To implement a scalable access control system. Part of building a scalable granular access control mechanism is to pre-join secrecy policy with data in the form of labels. Secrecy requirements can change over time, and it is important to be able to adapt granular access control mechanisms to keep up with changing policies.

8.3.2 How?

Use a labeling scheme that labels data with elements of policy that are unlikely to change over time, while more mutable policy elements are checked at query time. Keep track of the data-labeling policies that are applied at data ingest time to reduce assumptions made in policy evaluation at query time.

8.4 Maintain access labels

8.4.1 Why?

To make policy decisions on data with complex provenance. Accurately maintaining access labels includes an amount of provenance tracking.

8.4.2 How?

Label data as far upstream as possible. Keep track of labels that are referenced in data access policy through all data transformations. Use access control mechanisms that support Boolean logic and/or label sets to simplify label tracking through data aggregation.

8.5 Track admin data

8.5.1 Why?

To re-key the system when necessary. User roles and authorities change frequently and should not be assumed to be static. Changing this information is also tantamount to re-keying the system, so a proper data-basing strategy is necessary.

8.5.2 How?

Use a database with access control and auditing capabilities to keep track of admin data. Be sure to secure the connection between the admin data and big data architecture when using it as a policy information point.

8.6 Use standard single sign-on (SSO) mechanisms

8.6.1 Why?

To reduce the administrative burden of supporting a large user base. A big benefit of granular access control mechanisms is to broaden data sharing without incurring a large administrative cost. Single sign-on (SSO) solutions offload management of user authentication to enterprise-wide or even publicly available systems.

8.6.2 How?

Defer authentication to advanced SSO systems such as Lightweight Directory Access Protocol (LDAP), Active Directory, OAuth, or OpenId.

8.7 Employ proper federation of authorization space

8.7.1 Why?

To allow data providers to maintain control of access to their data when data analysis spans over multiple providers. The concept of big data is made more powerful by including analysis across many diverse data sets. However, data protection policies are complicated through the inheritance of policies from multiple data providers. Federating the authorization space supports a more modular approach to policy management through more precise tracking of data labels.

8.7.2 How?

Use a labeling scheme that disambiguates between labels managed by different authorities. Leverage user attributes that are coupled with advanced SSO systems, such as LDAP, Active Directory, OAuth, and OpenId.

8.8 Incorporate proper implementation of secrecy requirements

8.8.1 Why?

To ensure secrecy requirements. Ensuring that the requirements are properly implemented requires constructing arguments about data identity, user identity, user purpose, and many additional environmental considerations. In a granular access control system, these elements can be disbursed throughout the architecture for performance reasons, but proper use of the tools available can lead to straightforward arguments for the preservation of secrecy.

8.8.2 How?

Deconstruct secrecy requirements into parts that are data specific, user specific, and application specific. Use a consistent labeling strategy to ensure that all of the necessary fine-grained information about the data is found in the labels. Use an authoritative source for user attributes. Infer that any purpose- or application-specific restrictions are either provided by the application and audited or inferred by application-specific authentication.

8.9 Implement logical filter in application space

8.9.1 Why?

To prevent data abuse and leakage by application. With granular access control mechanisms, applications have the ability to access data for many different purposes on behalf of a user. Any policies that prevent cross-purpose data combinations rely in some part on the application properly separating different uses of data. For example, a single health care provider may access data for the purpose of diagnosing a specific patient or for the purpose of characterizing a population of patients. If an application is reused for both purposes, then the application is responsible for avoiding leakage of personally identifiable information and personal health information from the first to the second use.

8.9.2 How?

Ensure that the application does not “change hats” without clearing session data. Defer to the database’s granular access control mechanism where possible.

8.10 Develop protocols for tracking access restrictions

8.10.1 Why?

To operate access control system as expected. Data privacy policies are often complex and difficult to implement with 100 accuracy. It is important to review what policies are in place on a given system, as well as maintaining the capacity to revise policies when deemed necessary.

8.10.2 How?

Protocols for tracking access restrictions come in two forms: those used to encode policies and those used to audit the instantiations of those policies. Logging and aggregating audits of policy decisions can provide critical information about what data users are accessing, what data users are trying to access (unsuccessfully) and how users are attempting to access the system in an unauthorized manner. Analysis of those audits is critical to refining policy for both the purposes of increased privacy and increased sharing. When encoding policies, choose a standard language such as eXtensible Access Control Markup Language (XACML). This will allow policy creators to bring in help from a community of tool builders for visualizing, editing, and reasoning during the encoding process.

9.0 Granular Audits

It is a best practice to perform granular audits. This is primarily due to the possibility that users may miss true positive alerts from a real-time security monitoring system that may warn them of an attack. The following best practices should be followed in regard to establishing a system for granular audits.

9.1 Create a cohesive audit view of an attack

9.1.1 Why?

To answer essential questions following an attack. As an attack may consist of different stages (e.g., a reconnaissance scan, followed by a vulnerability attack, etc.), it is important to get all the pieces of the puzzle collected and put into their respective places. In only this way can a cohesive view be established. It is important to build up a consistent and cohesive audit trail that answers basic questions, including: what happened? when did it happen? how did it happen? who was the perpetrator? and why did it happen?

9.1.2 How?

- Enable auditing capabilities in a big data infrastructure.
- Select the relevant capabilities depending on features of infrastructure components, such as log information from routers, applications, operating systems (OS), databases, and so on.
- Use a SIEM solution, as well as audit and forensics tools to process the collected audit information.

9.2 Evaluate completeness of information

9.2.1 Why?

To provide a full audit trail. All relevant information that builds up the trail has to be available. As such, completeness of information is key.

9.2.2 How?

- Evaluate which audit information might be relevant upfront and which audit information is available in general. This data may come from log files, OS settings and profiles, and

database configurations, for example.
• Setup and enable the necessary audit settings of the big data infrastructure, like routers, OS, Hadoop, and applications for which the audit information must be collected upfront. Setup the settings for other audit information, which might be collected in a later stage.
• Collect and process the audit data with a SIEM solution or auditing tool, when applicable.

9.3 Ensure timely access to audit information

9.3.1 Why?

To accelerate incident response. Time is the most important aspect in case of an attack, not only to determine when the attack happened, but also to have timely access to audit information in case it is needed. This goes hand-in-hand with the best practice mentioned in section 9.2.

9.3.2 How?

As described in best practice 9.2, setting up audit information upfront is key, not only for the completeness of the information, but also to get access to the information in a timely fashion.

9.4 Maintain integrity of information

9.4.1 Why?

To ensure trust in audit data. Without an integrity guarantee, there is no single version of the truth. Audit information can't be trusted and, as such, becomes useless.

9.4.2 How?

• Consider implementing integrity controls, like secure hashing. Use SHA-1, SHA-224, SHA-256, and/or SHA-512.
• Ensure the integrity of the audit information is guaranteed along the complete path of collection, processing, use and storage of the data. This helps to ensure the information's chain of custody.

9.5 Safeguard confidentiality of information

9.5.1 Why?

To prevent audit data from reaching the wrong hands. While ensuring the integrity of the audit emphasizes the accuracy of the information, confidentiality addresses the fact that not everyone needs to have access to that data. This is important because audit information contains data related to potential attackers and methods. As such, only authorized people (typically auditors and forensic researchers) should be awarded access.

9.5.2 How?

• Ensure that audit information is stored separately (see best practice 9.9).
• Ensure that audit information can only be accessed by authorized people (see best practice 9.6).
• Consider the use of encryption to encrypt the audit information, where feasible and applicable.

9.6 Implement access control and monitoring for audit information

9.6.1 Why?

To safeguard audit information. Audit information contains important data regarding the "what, when and who" of system access and data use. As this information is critical for investigations, access to this information has to be strictly controlled. Limited access also helps to avoid tampering with audit information, which may allow the attacker to erase his/her tracks.

9.6.2 How?

• When setting up the identity and access management process, carefully determine who has access to audit information and consider creating a designated "auditor" position.
• Monitor the use of this role on a regular basis, especially for exceptions or access attempts.
• Ensure that a cohesive view of the attack is created from audit information.

9.7 Enable all required logging

9.7.1 Why?

To build up an audit view. This process is only as effective as the data collected. Most of this information comes from log files (e.g., networks, OS, database, and applications). As such, enabling logging according to what needs to be audited is key.

9.7.2 How?

This is related to best practice 9.2, which describes which information is needed. Based on this data, evaluate the logging capabilities of the big data infrastructure components and enable the different logging features.

9.8 Use tools for data collection and processing

9.8.1 Why?

To find actionable information without being overwhelmed by big data. There is simply too much information (especially now with big data) to be processed manually. Tools—such as a SIEM tool—are necessary to collect and process the data.

9.8.2 How?

Use available tools such as a SIEM tool to process the information gathered from logs.

9.9 Separate big data and audit data

9.9.1 Why?

To enforce separation of duties. As the audit data contains information about what has happened in the big data infrastructure, it is recommended to separate this data from the "regular" big data.

9.9.2 How?

• Implement the audit system in a different infrastructure than the big data infrastructure. For example, this may include a different network segment or cloud.
• Ensure that only the pre-defined "auditor" has access to the audit system and audit data.
• Monitor the audit system.

9.10 Create audit layer/orchestrator

9.10.1 Why?

To facilitate audit data analysis. As a big data infrastructure contains different components, each component has its own set of logging capabilities and log format. It is very hard, if not impossible, for an audit team to learn all the different intricacies of these logging features and formats.

9.10.2 How?

An audit layer can act as a middleware layer, which abstracts the underlying technical details for the auditor. The auditor communicates with the layer using an interface, which allows him/her to search. The audit layer will take care of the technical intricacies to collect the correct log files, normalize those when needed and provide information back to the auditor in terms which are understandable by the auditor. As this is complex, don’t try to build this yourself, but evaluate third-party/open-source solutions. One example is ElasticSearch. Some SIEM/log management vendors also provide solutions in this regard (e.g., Splunk, Sumologic, Loggly).

10.0 Data Provenance

With an increase in provenance metadata from large provenance graphs in big data applications, security is of great importance. With this in mind, the following suggestions are the best practices for data provenance security.

10.1 Develop infrastructure authentication protocol

10.1.1 Why?

To prevent malicious parties from accessing data. Without infrastructure authentication, an entire group of users (including unauthorized individuals) can have access to data—including some parties who may misuse the provenance data. For example, an adversary is likely to release the provenance information or data itself—which likely includes some sensitive information—to the public.

10.1.2 How?

An authentication protocol is designed as a sequence of message exchanges between principals allowing the use of secrets to be recognized. Design consideration is needed in, for instance, whether the protocol features a trusted third party and what cryptographic schemes and key management schemes are appropriate. If the number of nodes needed to authorize is extensive, the public key infrastructure can be used to authorize each worker and a symmetric key can be sent to each of those individuals. The authorized nodes can use the symmetric key to communicate because the overhead of the symmetric key is much smaller than the public key method. The unauthorized users cannot transmit the information due to the absence of a symmetric key. In this way, users can build efficient authentication infrastructure to authorize all valid parties.

10.2 Ensure accurate, periodic status updates

10.2.1 Why?

To collect data correctly. Advances in wireless technology have increased the number of mobile devices that are now used to collect, transmit and store data. However, there may be some malicious nodes and lazy nodes in the wireless environment. The malicious nodes are active attack nodes. When a malicious node receives data transmitted from a data owner, it actively tampers with the information, eavesdrops the content, or drops the data package while sending fake data to the next relay node. Lazy nodes, on the other hand, simply choose not to store-and-forward the data due to the energy consumption required for data transmission. These manipulations of collection and transmission of information can be stored as part of a provenance record. The status of provenance records should be periodically updated.

10.2.2 How?

Trust and reputation systems can be introduced into the wireless network to address lazy and malicious node issues. Reputation systems provide mechanisms to produce a metric encapsulating reputation for each identity involved in the system. For example, if the node performs a malicious act or eavesdrops for sensitive information, the system can assign some pre-defined negative value to the node as a rating. Also, the system can assign pre-defined positive value to nodes with normal behavior, while lazy behaviors are simply assigned "zero" as a rating value. One node can also give a trust value to another node based on their interactions. This trust information can be stored as part of the provenance record, which can be periodically updated in order to reflect its most recent status. Some pre-defined threshold should be designated by the system to identify that nodes are valid. If the trust value of the node is lower than that pre-defined threshold, the node is then recognized as invalid and is not used to transmit or collect information.

10.3 Verify data integrity

10.3.1 Why?

To ensure trust in data. In real-world environments, provenance data is typically stored on a personal laptop or in a remote database center. In turn, there may be some unevaluated risks associated with losing portions of the provenance data by accident, such as unforeseen damage to the hard drive. In attacker-active environments, there may also be adversaries who modify segments of original data files in the database. To that end, it is essential to detect whether user data has been tampered with. The establishment of data integrity is one of the most important markers of provenance security.

10.3.2 How?

Checksums are considered one of the least expensive methods of error detection. Since checksums are essentially a form of compaction, error masking can occur. Error detection in serial transmission by arithmetic checksum is an alternative to cyclic redundancy checks (CRC). Reed-Solomon (RS) codes are a special class of linear, non-binary block codes with the ability to correct both errors and erasure packets. An RS code achieves ideal error protection against packet loss since it is a maximum distance separable (MDS) code. Another efficient way to maintain the integrities of both data and provenance information is the use of digital signatures, which keeps data from being forged by adversaries. Digital signatures are the most common application of public key cryptography. A valid digital signature indicates that data was created by a claimed sender and the information was not altered during transmission. This method provides a cryptographic way to secure the data and its provenance integrity; furthermore, the verification cost is reasonable even when the volume of data becomes extensive. New, efficient digital signature schemes will likely be developed to ensure the integrity of information in big data scenarios.

10.4 Ensure consistency between provenance and data

10.4.1 Why?

To ensure provenance information is trustworthy. Separating provenance from its data introduces problems and potential inconsistencies. Provenance should be maintained by the storage system. If the provenance does not exactly match with the corresponding data, users cannot use the provenance data confidently. Effective use of provenance will establish a record of process history for data objects. Historical information about the data can be constructed in the form of a chain, which is also referred to as a provenance chain. The provenance chain of a document is a non-empty and time-ordered sequence of provenance records. For provenance data stored in the database, the provenance chain should be well-organized to make provenance records consistent. Otherwise, provenance information cannot be trusted if consistent record keeping cannot be established.

10.4.2 How?

In order to guarantee the consistency between the provenance and its data, the hash function and hash table can be used to address this challenge. Hash map keys can be utilized to locate data. It is not advised to utilize the provenance data directly because the information is likely enormous in size and the structure of the data is complicated. Before the provenance data is stored in the database, the hash function can be used to generate the hash value of the selected data block. Users can then apply the provenance information and the hashed provenance data to build the hash table. As a basic component of the provenance chain, a provenance record will denote a sequence of one or more actions performed on the original data. In order to achieve the consistency of the provenance record, the cryptographic hash of both the newly modified provenance record and history chain of the provenance record are taken as input. The consistency of the provenance chain can be verified by checking, at the beginning of each editing session, whether the current provenance chain matches the provided hash value. As massive provenance data is generated in big data applications, users need highly efficient methods to keep the consistency between the provenance, its data, and the provenance chain itself.

10.5 Implement effective encryption methods

10.5.1 Why?

To maintain security of provenance data. As cloud computing techniques continue to evolve, users often outsource large amounts of scientific provenance data to the cloud for storage/computation. When the data provider transmits the provenance data to the cloud servers in the way that the data is expressed in plaintext form, the transmitted data flow can be easily eavesdropped by an adversary. Additionally, the cloud server will always be considered a third-party server which cannot be fully trusted.

10.5.2 How?

One existing method to keep data secure during transmission is to transform the original data into ciphertext form. Before the provenance data is sent to the untrusted server, the data owner encrypts the information by using the secret key. Anyone who is untrusted by the transmitter cannot have the secret key and, therefore, cannot decrypt the ciphertext to get the original data. Only the authorized party can decrypt the ciphertext by using the secret key. Users can also utilize encryption to secure outsourcing of computation tasks on untrusted servers. Before users send out the high-burden computation task to the cloud server, the data owner can first “blind” the original information by using lightweight encryption technology. At that junction, the data owner can outsource the encrypted information to the cloud server to handle the high-burden computation task. Because the cloud server does not have the secret key, this server cannot disclose the original provenance data. When the computation task is finished, the data owner can use the secret key to recover the data handled by the cloud server. By using encryption, the data owner can outsource the computational task and enable confidential storage of the task and data in the untrusted server.

10.6 Use access control

10.6.1 Why?

To prevent abuse and unauthorized disclosure of provenance records and data by malicious parties. One advantage of cloud computing is that the cloud provider has the power to share the data across different user environments. However, only authorized users should have access to shared data. The volume of stored provenance data is normally extensive in the cloud server, so under most circumstances the user (data owner) may wish to restrict information accessibility. On the other hand, the provenance record may also contain some private information, such as a user’s personal profile and/or browsing log. The adversary may offensively access and misuse the provenance record or data itself without appropriate access control being applied. Moreover, the adversary may publically disclose sensitive data, which could damage the interests of the data owner.

10.6.2 How?

Appropriate access control helps avoid illegal access and private data leakage by limiting the pool of actions or operations that a legitimate system user may perform. One popular access control technique is role-based access control (RBAC). In this scenario, provenance data owners store data in an encrypted form and grant access to the information solely for users with specific roles. Only authorized users have data access according to the RBAC method. Attribute-based encryption (ABE) techniques can be implemented in order to achieve fine-grained access control. Ultimately, however, a ciphertext-policy attribute-based encryption (CP-ABE) scheme is the most advisable method for cloud-computing environments. In this scheme, different users in the system have their corresponding attribute sets created according to their inherent properties. Authorized users are assigned private keys which are associated with different attribute sets. The data owner encrypts the provenance data with access structure and assigns encrypted data to those who possess privileges.

10.7 Satisfy data independent persistence

10.7.1 Why?

To preserve indistinguishability of provenance data. When updates occur between two adjacent pieces of the provenance data, the user cannot distinguish among these pieces. This is referred to as "independence." For example, the provenances of two segments of derived data, such as simulation results, won't reveal differences between the data. Sometimes, a user is granted partial privileges to access pieces of provenance data. If data has not met the standards for the "independence" designation, the user is able to distinguish the difference between two pieces of provenance data. Some segments of the provenance data may involve sensitive information, which the data owner does not want the data consumer to access. "Independence" designation for different provenance records should also be achieved because some users are only able to access part of the provenance chain.

10.7.2 How?

Symmetric keys and distributed hash tables (DHTs) technology can be used to establish and maintain independence among different pieces of provenance data. The symmetric keys are selected randomly and independently to encrypt different pieces of the provenance data. Due to differences among independent symmetric keys, encrypted pieces of provenance data are also independent. Moreover, different pieces of the same provenance data should not be stored successively because they have the same patterns. In order to distribute the different pieces of provenance data, users can utilize the distributed hash tables (DHTs). Segments of provenance data can be retained, in a distributed fashion, with DHTs. Moreover, the hash function can be recalled to help different provenance records achieve independence. The individual provenance record should be hashed, and then different hashed provenance records can be reconstructed as a chain. The modified individual provenance record only affects the corresponding hashed component and will not impact other components in the hash chain. That is to say, different parts of provenance records can achieve independent persistence.

10.8 Utilize dynamic fine-grained access control

10.8.1 Why?

To allow only authorized users to obtain certain data. Fine-grained data access control provides users (data consumers) with access privileges that are determined by attributes. In most real-world cases, user-assigned privileges and/or attributes vary with time and location, which may need to be incorporated in access control decision.

10.8.2 How?

Using attribute-based encryption, fine-grained access control can be applied to encrypted provenance data. In order to reach the dynamic property, users can introduce the dynamic attribute and weighted attribute into the attribute-based encryption. The dynamic attribute can be described as a frequently changing attribute, such as a location coordinate, while other attributes are considered weighted attributes. These attributes have different weights according to their importance, which are defined in the access control system. Every user in the system possesses a set of weighted attributes, and the data owner encrypts information for all users who have a certain set of attributes. However, a user's private key has a specific kind of weighted access structure. In order to decrypt a message, a ciphertext with a set of weighted attributes must satisfy the weighted access structure. The weight of the attribute can be increased or decreased to reflect the dynamic property.

10.9 Implement scalable fine-grained access control

10.9.1 Why?

To protect large-scale provenance data. A considerable amount of provenance data is stored and exchanged in databases. Database systems allow data consumers access to various types of provenance data in accordance to access policies designed by the data owner. However, an access policy should be scalable in order to meet the ever-growing volume of provenance data and user activity within a group. If the access policy is not scalable, any future policy modifications that may be necessary will be difficult to implement.

10.9.2 How?

Attribute-based encryption can be utilized to achieve fine-grained access control, which was introduced in the previous section. To achieve scalability, data owners can introduce a semi-trusted proxy that exploits the "so-called" proxy re-encryption technology. This will allow the proxy to decrypt the message and re-encrypt it with the new access structure. Since the proxy is "honest-but-curious"—which indicates it will try to gather as much private information as possible based on possession—users cannot allow the proxy to fully decrypt the message. Ideally, users would control the policy while the proxy is utilized to produce a partial ciphertext for the policy specified by that purpose. When access structure attributes in the ciphertext need to be modified by the data owner, the newly specified policy is given to the proxy. Initially, the proxy uses the previous partial attribute set to decrypt the partial ciphertext, and then applies the newly specified policy to encrypt the partially decrypted ciphertext. Because the proxy does not have the full attribute set, the proxy can only partially decrypt the message and not the entirety of the provenance data.

10.10 Establish flexible revocation mechanisms

10.10.1 Why?

To prevent access by unauthorized entities. With such high volumes of provenance data stored in databases, maintaining effective access control is a constant challenge for data managers. Access to the data can be easily abused (even if the privilege of accessing the data is expired) when the access permission is not appropriately updated. (For instance, a fired employee could still abuse access privilege if access is not revoked in timely manner.) In some cases, data may only be valid during a particular time interval. Data managers also need an efficient data revocation scheme after the data has expired.

10.10.2 How?

One existing method to keep data secure during transmission is to transform the original data into ciphertext form. Before the provenance data is sent to the untrusted server, the data owner encrypts the information by using the secret key. Anyone who is untrusted by the transmitter cannot have the secret key and, therefore, cannot decrypt the ciphertext to get the original data. Only the authorized party can decrypt the ciphertext by using the secret key. Users can also utilize encryption to secure outsourcing of computation tasks on untrusted servers. Before users send out the high-burden computation task to the cloud server, the data owner can first "blind" the original information by using lightweight encryption technology. At that junction, the data owner can outsource the encrypted information to the cloud server to handle the high-burden computation task. Because the cloud server does not have the secret key, this server cannot disclose the original provenance data. When the computation task is finished, the data owner can use the secret key to recover the data handled by the cloud server. By using encryption, the data owner can outsource the computational task and enable confidential storage of the task and data in the untrusted server.

References

[ABC+07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson and Dawn Song. Provable data possession at untrusted stores. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, ACM CCS 07: 14th Conference on Computer and Communications Security, pages 598–609. ACM Press, October 2007.

[AGT01] Aris Anagnostopoulos, Michael T. Goodrich and Roberto Tamassia. Persistent authenticated dictionaries and their applications. Information Security. 379-393. Springer Berlin Heidelberg, 2001.

[AKS07] Onur Aciiçmez, Çetin Kaya Koç and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In Masayuki Abe, editor, Topics in Cryptology – CT-RSA 2007, volume 4377 of Lecture Notes in Computer Science, pages 225–242. Springer, February 2007.

[BBS04] Dan Boneh, Xavier Boyen and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, Advances in Cryptology – CRYPTO 2004, volume 3152 of Lecture Notes in Computer Science, pages 41–55. Springer, August 2004.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, Advances in Cryptology – CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pages 213–229. Springer, August 2001.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, TCC 2005: 2nd Theory of Cryptography Conference, volume 3378 of Lecture Notes in Computer Science, pages 325–341. Springer, February 2005.

[BS12] Nitish Balachandran and Sugata Sanyal. A review of techniques to mitigate sybil attacks. arXiv preprint arXiv: 1207.2617, 2012.

[BSW07] John Bethencourt, Amit Sahai and Brent Waters. Ciphertext-policy attribute-based encryption. In 2007 IEEE Symposium on Security and Privacy, pages 321–334. IEEE Computer Society Press, May 2007.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, TCC 2007: 4th Theory of Cryptography Conference, volume 4392 of Lecture Notes in Computer Science, pages 535–554. Springer, February 2007.

[CJJ+13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Ran Canetti and Juan A. Garay, editors, CRYPTO (1), volume 8042 of Lecture Notes in Computer Science, pages 353–373. Springer, 2013.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Advances in Cryptology – CRYPTO 2011, Lecture Notes in Computer Science, pages 487–504 Springer, August 2011.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, Advances in Cryptology – EUROCRYPT'91, volume 547 of Lecture Notes in Computer Science, pages 257–265. Springer, April 1991.

[CW13] Jie Chen and Hoeteck Wee. Fully, (almost) tightly secure IBE and dual system groups. In Ran Canetti and Juan A. Garay, editors, CRYPTO (2), volume 8043 of Lecture Notes in Computer Science, pages 435–460. Springer, 2013.

[Dou02] John R. Douceur. The sybil attack. International Workshop on Peer-to-Peer Systems. Springer Berlin Heidelberg, 2002.

[Dwo06] Dwork C. Differential privacy. InAutomata, languages and programming 2006 Jul 10 (pp. 1-12). Springer Berlin Heidelberg, 2006.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, 41st Annual ACM Symposium on Theory of Computing, pages 169–178. ACM Press, May/June 2009.

[GHS12a] Craig Gentry, Shai Halevi and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In Advances in Cryptology – EUROCRYPT 2012, Lecture Notes in Computer Science, pages 465–482. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Advances in Cryptology – CRYPTO 2012, Lecture Notes in Computer Science, pages 850–867. Springer, August 2012.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, ACM CCS 06: 13th Conference on Computer and Communications Security, pages 89–98. ACM Press, October/November 2006. (Available as Cryptology ePrint Archive Report 2006/309.)

[ISO05] ISO. ISO 27002: 2005. Information Technology-Security Techniques-Code of Practice for Information Security Management. ISO,. 2005.

[JK07] Ari Juels and Burton S. Kaliski Jr. Pors: proofs of retrievability for large files. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, ACM CCS 07: 14th Conference on Computer and Communications Security, pages 584–597. ACM Press, October 2007.

[JR13] Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In ASIACRYPT, 2013.

[KL10] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2010.

[KS06] Karen Kent and Murugiah Souppaya. Guide to Computer Security Log Management. NIST, September 2006.

[LLV07] Ninghui Li, Tiancheng Li and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, pp. 106-115. IEEE, 2007.

[LMSV11] Jake Loftus, Alexander May, Nigel P. Smart and Frederik Vercauteren. On cca-secure somewhat homomorphic encryption. In Ali Miri and Serge Vaudenay, editors, Selected Areas in Cryptography, volume 7118 of Lecture Notes in Computer Science, pages 55–72. Springer, 2011.

[MKG07] A. Machanavajjhala, D. Kifer, J. Gehrke and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD), 1(1), 3, 2007.

[MR15] Avradip Mandal and Arnab Roy. Relational Hash: Probabilistic Hash for Verifying Relations, Secure Against Forgery and More. Advances in Cryptology--CRYPTO 2015. Springer Berlin Heidelberg, 2015.

[NLV11] Michael Naehrig, Kristin Lauter and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, CCSW, pages 113–124. ACM, 2011.

[Per05] Colin Percival. Cache missing for fun and profit. In Proc. of BSDCan 2005, 2005.

[RST01] Ronald L. Rivest, Adi Shamir and Yael Tauman. How to leak a secret. In Colin Boyd, editor, Advances in Cryptology – ASIACRYPT 2001, volume 2248 of Lecture Notes in Computer Science, pages 552–565. Springer, December 2001.

[SGLM08] Mark W. Storer, Kevin Greenan, Darrell DE Long and Ethan L. Miller. Secure data deduplication. In Proceedings of the 4th ACM international workshop on Storage security and survivability, pp. 1-10. ACM, 2008.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, Advances in Cryptology – CRYPTO'84, volume 196 of Lecture Notes in Computer Science, pages 47–53. Springer, August 1985.

[SSS11] Emil Stefanov, Elaine Shi and Dawn Song. Towards practical oblivious RAM. arXiv preprint arXiv:1106.3652 , 2011.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, Advances in Cryptology – EUROCRYPT 2005, volume 3494 of Lecture Notes in Computer Science, pages 457–473. Springer, May 2005.

[Swe02] L. Sweeney. k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10(05), 557-570, 2002.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Advances in Cryptology – EUROCRYPT 2010, Lecture Notes in Computer Science, pages 24–43. Springer, May 2010.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, Advances in Cryptology – CRYPTO 2009, volume 5677 of Lecture Notes in Computer Science, pages 619–636. Springer, August 2009.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography, Lecture Notes in Computer Science, pages 53–70. Springer, 2011.

[WWRL10] Cong Wang, Qian Wang, Kui Ren and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In INFOCOM, 2010 Proceedings IEEE, pp. 1-9. Ieee, 2010.