Cryptographic access control for big data platforms

Christoffer Viken

Master of Science in Communication Technology
Submission date: June 2017
Supervisor: Colin Alexander Boyd, IIK
Co-supervisor: Mari Grini, Telenor Digital

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Cryptographic access control for big data platforms
Student:	Christoffer Viken

Problem description:

The General Data Protection Regulation (GDPR) requires a routine or scheme to ensure that "Personal data should be processed in a manner that ensures appropriate security and confidentiality of the personal data, including for preventing unauthorised access to or use of personal data and the equipment used for the processing.".

Traditional data warehouses or databases have good security mechanisms. The data lake model of data storage, however, has few good and established solutions for data access control. One possible solution is to try to protect the data using encryption.

This project will look at using encryption as an access control mechanism for a data lake. The challenges this presented are different form those of traditional data warehouses. The challenges grow with the granularity of the access control requested.

This project will present possible solutions with their strengths and weaknesses. Other issues that may be addressed: what kind of pre-processing needs to be applied to the data, which security guarantees it provides, and the limitations of analysing the data in an encrypted state.

Responsible professor:	Colin Alexander Boyd, IIK, NTNU
Supervisors:	Mari Grini, Telenor Digital
	Gareth Thomas Davies, IIK NTNU

Abstract

This thesis has looked into the performance of select cryptographic algorithms and compared them to each other in the context of big data. The algorithms compared are Advanced Encryption Standard (AES), RSA, and a Attribute-Based Encryption (ABE) algorithm called GPSW. The results confirmed common knowledge that schemes with advanced properties are slow. The results show the performance ratio between the algorithms on Java and served as a baseline for comparing performance impacts of encryption with different algorithms.

Later in the thesis an attempt to design a system using the algorithms tested, and looked into the performance impacts of using different encryption algorithms. Along with the systems were an estimation function of how long encryption/decryption will take from a pure cryptographic standpoint.

Because the cost/benefit of implementing a pure cryptographic access control scheme is going to be marginal, it was deemed beneficial to look into existing systems that can be used to implement access control, if not by cryptography, then maybe in conjunction with it.

Lastly the thesis looked into the impact of encrypting data. For instance, there exist a lot of avenues of data leakage even though the data itself may be encrypted. Some of these issues are described and strategies to mitigate them are outlined. Most of these problems are avoidable as long as one is aware of them. Another related impact is the performance loss/gain from using different schemes. These impacts are going to be more prominent when hardware accelerated encryption is used. There were carried out some tests on this and the results showed that if care is taken with the crypto-implementation the performance gain can be significant.

While the conclusion were that pure cryptographic solutions are unsuitable for production use, there exist precautions that apply to any system where data is encrypted. These things impact both performance and confidentiality and should be taken into account by anyone looking into encrypting their data.

Sammendrag

Denne avhandlingen har undersøkt ytelsen til utvalgte kryptografiske algoritmer, og sammeliknet dem med hverandre for bruk i big data. Algoritmene som er sammenliknet er Advanced Encryption Standard (AES), RSA, og en Attributt-Basert Kryptering (ABE) omtalt som GPSW. Resultatene bekreftet konsensusen om at kryptoalgoritmer med avanserte egenskaper er tunge og tar lang tid. Ytelsesforholdet mellom disse algoritmene, implementert i Java, tjente som et grunnlag for å sammenlikne ytelsen av å bruke forskjellige algoritmer.

Senere i avhandlingen gjøres et det forsøk på å designe et system som bruker algoritmene som ble testet. Forskjellen i ytelse mellom de forskjellige systemene, og bruken av de forskjellige algoritmene, er så blitt evaluert. I dette inngår et estimat av hvor lang tid kryptering/dekryptering vil ta sett fra et rent kryptografisk perspektiv.

Fordi kost/nytten av å implementere en ren kryptografisk adgangskontrollmekanisme er marginal, ble det vurdert som hensiktsmessig å underøke andre eksisterende løsninger for adgangskontroll. Om disse løsningene ikke var av en direkte kryptografisk natur, så kunne de kanskje fungere sammen med kryptering av dataene.

Til sist er ringvirkninger av å kryptere data tatt i betraktning. For eksempel finnes det mange måter informasjon om kryptert data kan lekke selv om det er kryptert. Noen av disse problemene, og strategier for å unngå dem, er blitt beskrevet. De fleste av disse problemene er enkle å unngå, så lenge man er klar over at de eksisterer. Det ble også utført tester som viste at ytelse varierer betydelig avhengig av implementasjon.

Selv om konklusjonen ble at rene kryptografiske løsninger er uegnede for praktisk bruk, finnes det tiltak som alle som skal kryptere data bør ta hensyn til. Disse tiltakene påvirker både ytelse og krypteringens evne til å skjule data, og alle som ønsker å kryptere dataene sine burde gjøre seg kjent med disse.

Preface

This project is a 30p (1 semester) project and the finishing part of a masters degree. The project grew out from the info-sec angle of specialisation and spread out from there. While none of the primary research is going to be relevant to any immediate development, there are minor elements and data points that are highly relevant to developers.

Along the way I did a few discoveries and very few setbacks. In the 21 weeks of this project I spent a lot of time running numbers and trying to find something of interest. While I never intended to implement a system from this research, working with the ABE implementations available to me killed all residual ambitions.

Credit for the elements used in the cover photo goes to iStock.com/scanrail and iStock.com/Epifantsev.

I would like to thank everyone involved in the project for helping me get through the frustrations and confusions experienced during the project. Apart from my supervisors I would like to thank Simon Randby for proof reading. I would also have to thank the Infrastructure Team at Telenor Digital Tyholt for providing sufficient distractions for me to keep my maintainable concentration. If it wasn't for the people around me the results would be quite different.

Christoffer Viken
Trondheim, June 2017

Contents

List of Figures

List of Terms

AES-NI	Advanced Encryption Standard (AES) New Instructions, an extension to x86 that implements “AES Round” as an instruction (6 instructions in total).
Amazon EC2	Amazon Cloud service for provisioning [Virtual Machines].
Amazon EMR	Amazon Cloud service to provision a Apache Hadoop compatible cluster, allowing Amazon Simple Storage Service (S3) as a storage system.
Amazon RedShift	A data processing platform by Amazon that implements a Structured Query Language (SQL)-like interface to the data it holds.
Amazon S3	Amazon Cloud service for storing data.
Apache Kafka	A message passing system originally designed by LinkedIn, designed to handle lots of data efficiently.
Asymmetric Cryptography	Also known as public-key cryptography, a form of cryptography where the decryption key is different form the encryption key.
AWS	Amazons services for providing computing resources to different actors..
Broadcast Encryption	An encryption scheme that allows for more keys to decrypt the ciphertext.
Ciphertext	The encrypted data in a encryption scheme.
Cleartext	The unencrypted data in a encryption scheme.

GPSW	An attribute-based encryption scheme named for its authors: Goyal, Pandey, Sahai, and Waters..
Hadoop	A framework for big data and processing.
HBase	A database modelled after Google BigTable, built on top of Hadoop.
HIVE	A framework for processing big data trough an SQL interface.
MapReduce	A paradigm for data processing developed by Google, and still in wide use, see section 2.2.
PCI-DSS	Payment Card Industry Data Security Standard, a standard for the security of credit card and banking systems.
PostgreSQL	An Open Source SQL database, favoured by the student and therefore used for examples.
Pseudomysation	Replacing a name or property with a unrelated placeholder (pseudonym).
Query scheduler	An algorithm that looks at a data processing query and devises a plan for an optimal way of executing this query.
REST	A system (often API), where each request is statefully independent. Therefore each request do not rely upon previous or later requests.
RSA	A public-key scheme named after its authors: Rivest, Shamir, and Adleman.
Spark	A framework for making big data processing applications.
Symmetric Cryptography	As opposed to Asymmetric Cryptography symmetric cryptography uses the same key for decryption as for encryption..

UNIX	Also known as UNIX time sharing system, an operating system developed at Bell Labs in the 1970s the fifth version (System V) has since become a standard for similar philosophy operating systems, UNIX is here used to refer to this standard.
x86	Name used to refer to microprocessors evolved from Intel's 8086 series. (80286, 80386, etc. later Pentium).
Yet Another Resource Negotiator	The Hadoop 2.0 resource management system.

List of Acronyms

ABAC Attribute-Based Access Control.

ABE Attribute-Based Encryption.

AES Advanced Encryption Standard.

API Application Programming Interface.

BI Business Intelligence.

CBC Cipher Block Chaining.

CPU Central Processor Unit.

CSV Comma-Separated-Value file.

CTR CounTeR mode.

ECB Electronic CodeBook.

FPGA Field-Programmable Gate Array.

GDPR General Data Protection Regulation.

GFS Google File System.

HDFS Hadoop File System.

HSM Hardware Security Module.

JDBC/ODBC Java Database Connectivity/Open Database Connectivity.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

KDS	Key Distribution System.
NSA	National Security Agency.
NTNU	Norwegian University of Science and Technology.
RDBMS	Relational Database Management System.
RDD	Resilient Distributed Dataset.
REST-API	REpresentational State Transfer (RESTful/REST) Application Programming Interface (API).
SLA	Service-Level Agreement.
SQL	Structured Query Language.
TLS	Transport Layer Security.
TPM	Trusted Platform Module.
VoIP	Voice over IP.
YARN	Yet Another Resource Negotiator.

Chapter 1 Introduction

Big data is a hot subject. It refers to a paradigm for processing data too large for traditional databases. Most philosophies in this paradigm have the common factor that they use a distributed processing model to spread the workload across multiple machines.

One paradigm within big data is the *data lake*. Data lakes are described in more detail in section 2.1. A data lake is essentially a common storage platform for files. Unlike databases systems there are no structural requirements in data lakes. The data could be as free form or as structured as required. The idea is that structuring the data could throw away potentially vital information, and therefore storing the data in the “raw” form allows for better information to be extracted.

The use case is the company Telenor Digital who operates a data lake for their data insights systems. They store logs for different services and query them for data. With the introduction of EU’s General Data Protection Regulation (GDPR) (described in detail in Appendix B) the need for appropriate access control has become stricter. While companies currently may have some form of protection on their systems to prevent information from leaking, GDPR introduces a regulatory demand for strict control. While there exist solutions to this problem already, the idea of solving this problem with cryptography is interesting.

1.1 Justification

There are several reasons why privacy is a big deal. This is especially true for big data. While a small dataset may be of no consequence, a big dataset have a lot more data to do correlation on, as a result the bigger datasets need more protection. In section 3.4 the discussion on how data that is anonymised can be analysed and matched to non-anonymous datasets. This again highlights the need to protect the datasets properly.

For GDPR references a basic threat model is constructed. This is based on the kinds of threats the GDPR was meant to regulate from a technical point of view.

There are two kinds of threats that need to be taken into consideration. The first is an insider or data analyst that wants to do unauthorised analysis on a dataset. This need not be malicious, it may be simple curiosity, or even in the eagerness to improve upon something accidentally stepping over bounds. The second threat is the outside attacker that might want to use the data for malicious purposes.

Both of these threats can be mitigated using a cryptographic access control system. The bounds-stepping analyst would ideally be stopped by a policy, while the outsider is stopped by the encrypted data. With the purely cryptographic solution all adversaries are granted infinite access to the ciphertext. This allows for easy threat modelling, a real system would most likely use restricted access to ciphertexts as well, and this would just make them more secure than the threat scenarios.

1.2 GDPR

The European Union (EU) has had a lot of focus on privacy in the last years. This has been motivated by a set of data breaches by different companies [RFa, RBF]. They have also focused on privacy issues from companies collecting or storing data from their users [RFb]. As a result of these the GDPR were suggested, and passed by the EU parliament on April 2016. It is scheduled for enforcement on May 2018.

Because GDPR is a great motivator for implementing access control in commercial systems some knowledge of the law in question may be beneficial. While the regulation covers a lot of subjects the majority of relevant information can be summarised with the following paragraphs:

In (83) it is stated that

> (...) the controller or processor should evaluate the risks inherent in the processing and implement measures to mitigate those risks, such as encryption.

In Article 32 it is stated that

> (...) the processor shall implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk (...) [In particular, specific provisions at least, where relevant:] (...) (d) the safeguards to prevent abuse or unlawful access or transfer;

These measures are going to be the focus of the thesis.

1.3 Goals

Privacy, security and GDPR compliance is the problem, cryptography could be the answer. The thesis will focus on trying to answer these concrete questions.

– Is cryptography a good solution for access control? In essence: Quantify the existence of any good schemes to use cryptography itself as an access control mechanism. One exist, explain how granular is it would be possible to grant access.

– If there is a cryptographic access control solution does this scale to big data applications? Some solutions may be suited well for a doctor's office and medical records where few records are encrypted and decrypted in each batch. Look into if there are schemes that scale up to big data levels. If the schemes do not scale, look into whether it is possible to trick one scheme into scaling.

– If there are no cryptographic mechanisms that allow practically usable access control are there any other ways of achieving GDPR compliance? Not an in-depth study but a basis for future research.

– Are there other avenues that could be interesting to look into into? A basis for further research.

1.4 Thesis structure

The thesis has 8 chapters, each chapter represents a logical partition of the thesis. The first chapter introduces the rest of the thesis.

Chapters 2, 3, and 4 are background information. They cover the relevant information that are either directly used, or deemed helpful for understanding the principles and ideas that are in use.

Chapter 5 contain a set of practical benchmarks on a chosen subset of algorithms to be able to compare the results to each other. Even though these algorithms can not be compared directly it is possible to establish a basis of equivalence using these benchmarks.

Chapter 6 takes the results from chapter 5 and tries to build a scalable system with them. This includes extreme cases that have scalability issues on purpose to illustrate the scalability issues.

Chapter 7 cover considerations for confidentiality and performance with encrypting data. Just because data is encrypted does not mean it is secure, there are avenues of leakage through encryption that need to be considered. A different aspect is hardware acceleration functionality that allows encryption to be faster, and even then there are seemingly insignificant choices that might yield surprising results.

Chapter 2

Big Data Technologies and Data Lakes

This chapter will cover some of the technologies used for big data. Technologies like MapReduce and Apache Hadoop are covered for their relevance to the big data philosophies and their widespread usage. Other technologies Apache Ranger and superscalar processors are included because they are relevant for later discussion.

2.1 The Data Lake Architecture

The basic idea of a data lake is that data is kept on a distributed storage medium in a raw form. The advantage is that there is no need for shaping the data to a schema. The disadvantage is that it will increase the processing cost when the data is handled [LS16].

The architectures available for data lakes are inspired by Google's Google File System (GFS) [GGL03] and MapReduce [DG08] architectures. These architectures are designed specifically to run on consumer-grade hardware with relatively high failure rates. This means that in theory the expensive processing cost is balanced out by low hardware costs. On a cloud architecture that advantage is slightly reduced, but it means that it is possible to use machines with a lower Service-Level Agreement (SLA) and thus a lower price.

2.1.1 Implementations

Data lake architectures are "typically, although not always, built using Hadoop"[LS16, p.2]. This is to make use of the Hadoop File System (HDFS) for the storage. The applications for processing the data can be run on Hadoop and Yet Another Resource Negotiator (YARN). Another solution for some users is to use Amazon Simple Storage Service (S3) as storage and use Amazon Elastic Compute Cloud (EC2) or Amazon Elastic Mapreduce (EMR) as a compute platform. Although there are more types of implementations these are the most commonly publicised, especially for users of Amazon Web Services (AWS) like Telenor Digital.

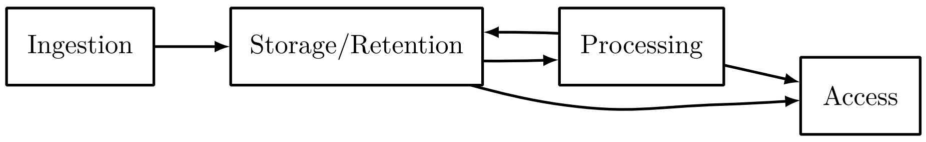


Figure 2.1: The flow of data in a data lake.

2.1.2 Stages

The data lake architecture has a four-“stage” system for data processing and storing the data. The different stages can in theory be replaced with different products independently, but in practice they tend to be tied into each other. They are described here so that they can be referred to later on.

ingestion

The ingestion stage is the stage that inserts the data into the data lake. For existing content Hadoop has a system for loading ordinary files into HDFS. For purposes where the data is being produced continually a stream-processing system to preprocess the data and insert it into the data lake as it is being produced is used.

Examples of ingestion systems are Apache Flume, Apache Kafka, and Apache Storm.

A cryptographic access control system should encrypt the data during ingestion. This would minimise the modifications that would have to be done on the data storage platform in order to implement such a solution. Note that encrypting data at the ingestion level would require support at the processing stage as well.

Storage/Retention

The storage system is where the data is stored. As mentioned earlier this is typically done using HDFS, but other Hadoop based systems are also used. Some of those other systems include Apache Hive, Apache HBase and Amazon RedShift.

Processing

The processing stage is responsible for the processing of data and queries and writing the result. Here there are a lot of options, more than are worth mentioning, but the interesting ones for this report would be MapReduce, HIVE, Amazon RedShift, and Apache Spark.

Access

The access stage is used for extracting data from the data lake, this can be done with any technology that is able to read from the datastore. Some common methods are data-piping frameworks like Apache Kafka, or REpresentational State Transfer (RESTful/REST) Application Programming Interface (API)s (REST-APIs).

2.1.3 Advantages of Design

The major selling point of the data lake design is that it is schemaless and therefore there is no need for data modelling in order to store data in it for later processing use. The other selling point of a schemaless data storage is that the data does not have to be put trough a lossy processing in order to fit it to the schema. It also means that it is possible to do queries that a schema might not have been designed for.

The second selling point is that a data lake does not create independent silos for data storage. Thus the entire organisation has access to all the data of the organisation.

2.1.4 Disadvantages for access control

While there are data lake management systems available that grant partial or full access control over the data, it is hard to manage access on a fine-gained level. This is inherent in the data lake openness by design. There are fine-grained access control systems, but the ones looked at for this report were all for Structured Query Language (SQL) style interfaces.

The schemalessness does not offer any form of access control other than a per-file basis. The only way of doing more fine-grained access control beyond that is to implement some sort of schema. The advantage of a data lake is that a schemaless data store and a data store using schemas can co-exist on the same platform, and a processing job could be employed to convert from one to the other.

2.2 Mapreduce

Whenever big data processing is mentioned one has to talk about the mapreduce scheme. It was developed at Google and described by Dean and Ghemawat in a white-paper from 2008 [DG08]. It is one of the concepts behind the original Hadoop framework and remains relevant for most big data applications.

The basic concept in mapreduce is simple.

1. Split the data into chunks

2. Process each chunk (map) and output a processed stream of data records with a reduction key.
3. Shuffle the records so that all records with the same key reside in the same location.
4. Join records together (reduce) so that there is only one record with each key.

This is some times referred to as map-shuffle-reduce because the shuffling strategy influences the over all performance.

The example that seems to be the most commonly used is a word counter application. This application is very basic but covers the concepts pretty well. the goal of the word counter application is to count occurrences of words in a document or a collection of documents.

The following process is illustrated in Figure 2.2. The mapper gets fed a line or a paragraph e.g. “No man is an island entire of itself; every man is a piece of the continent, a part of the main;”[Don24]. The mapper splits the paragraph into words and outputs key-value pairs, e.g. $(no, 1)$ $(man, 1)$ $(is, 1)$ $(an, 1)$ $(island, 1)$ $(...)$. Notice that the mapper only outputs one occurrence for every word, this is perfectly fine, and keeps the mapper simpler. These pairs are distributed to reducers that adds the occurrences together.

The programmer writes a mapper and a reducer for each application. The value field of each key-value pair can be any data. As a result most tasks can be solved with one or more passes of map-reduce.

2.2.1 Chaining operations

To make more advanced applications several map+reduce operations can be chained together in a tree. Note that most map-reduce frameworks store the results to disk so in practice this means that an advanced application would have to read from disk between each stage in the application. This could have performance issues compared to a system capable of temporarily storing the data in memory.

2.2.2 Joins

An example of a traditional Relational Database Management System (RDBMS) join in mapreduce can be done by a reduce-side-join. The assumption is that the “tables” are Comma-Separated-Value file (CSV) files and are read to the mapper one line (row) at a time. The mappers extract the join column as a key and reformats the data part we are interested in to a format that can be joined together in a reasonable

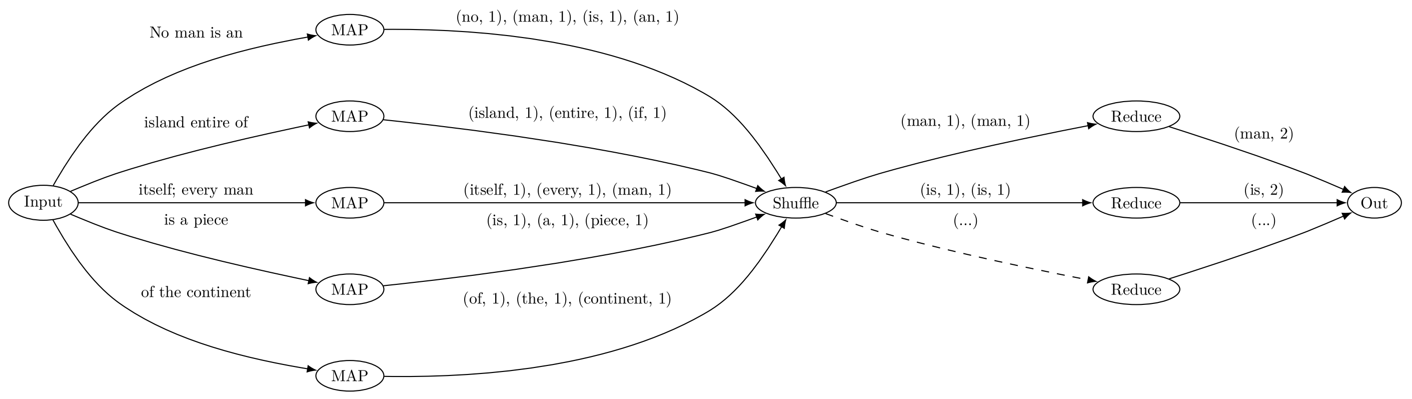


Figure 2.2: Block diagram of how mapreduce works. This is an example of the word counter application in action.

fashion. The reducer can aggregate the duplicate-keys together. Note that this is done on both tables separately. The second pass is done on both output sets in one batch. In this case there is a mostly passive mapper that just passes the data trough to the reducer. The reducer then joins the rows together.

Example

This example taken from PostgreSQL's documentation [Gro] for join has two tables: cities and temperatures.

City ID, City Name, Population, Year (population)

1,	San Francisco,	837442,	2013
2,	Hayward,	151574,	2013

City ID, Date, Temperature (Fahrenheit)

1	1994-11-27	46
2	1994-11-29	37
1	1994-11-29	43

Intermediate City Table:

```
1: c:(San Fransisco),
2: c:(Hayward)
```

Intermediate City Table:

```
1: w:(1994-11-27, 46), w:(1994-11-29, 43)
2: w:(1994-11-29, 37)
```

Final result:

```
1: c:(San Fransisco), w:(1994-11-27, 46), w:(1994-11-29, 43)
2: c:(Hayward), w:(1994-11-29, 37)
```

Optimising

The example used 3 map-reduce operations to do one join. It is possible to do the same operation with a single map-reduce. It requires a more complex mapper, but the speed advantage should be noticeable.

In the optimised version the operation is done to both tables. The mapper is smart enough to figure out what table the particular record belongs to and choose the appropriate mapper logic for each table. The reducer has the same logic as the second reducer, but may have to be able to to a many-to-manny join, as duplicate-key rows are not reduced in the first step.

2.2.3 Filtering

Because mapreduce expects any number of key-value pairs emitted from a map a filtering operation can simply be done by the mapper not emitting the filtered-out records. The one thing to keep in mind as a developer is that everything you need to in order to do filtering must be available in the record. As a result if one needs to do filtering on advanced criteria these criteria needs to be collected beforehand.

2.3 Apache Spark

Spark is a big data solution that was designed to be easier to use than mapreduce and also abstract away some advanced techniques [SD15]. As with mapreduce it operates well on plain text file formats, but it is capable of loading to and from most popular Hadoop storage systems. The rising popularity of this framework makes this framework a candidate for almost every processing framework evaluation at the time of writing.

2.3.1 Basic programming

Spark makes use of anonymous functions. One operation could be *map*(mapperfunction) that works like a map, and the mapper function is passed to it.

The basic unit in Spark is a Resilient Distributed Dataset (RDD) that can be seen as an abstract concept of an output (or input) of a mapreduce operation. One applies operations to the RDD is returned a new RDD. Some operations can combine RDDs by for instance joining them on a key. The RDDs are not directly data, but a series of steps required to generate the data. For this reason Spark can be lazy in data generation and only does data processing once an output is specified.

2.3.2 Behind the scenes

Behind the scenes Spark splits the RDDs into chunks and processes these chunks on the node they are stored on. It tries to minimise the stages in processing so if two operations can be fused together it will join them to one operation. For optimising the operations Spark also tries to put off shuffling across nodes as long as possible.

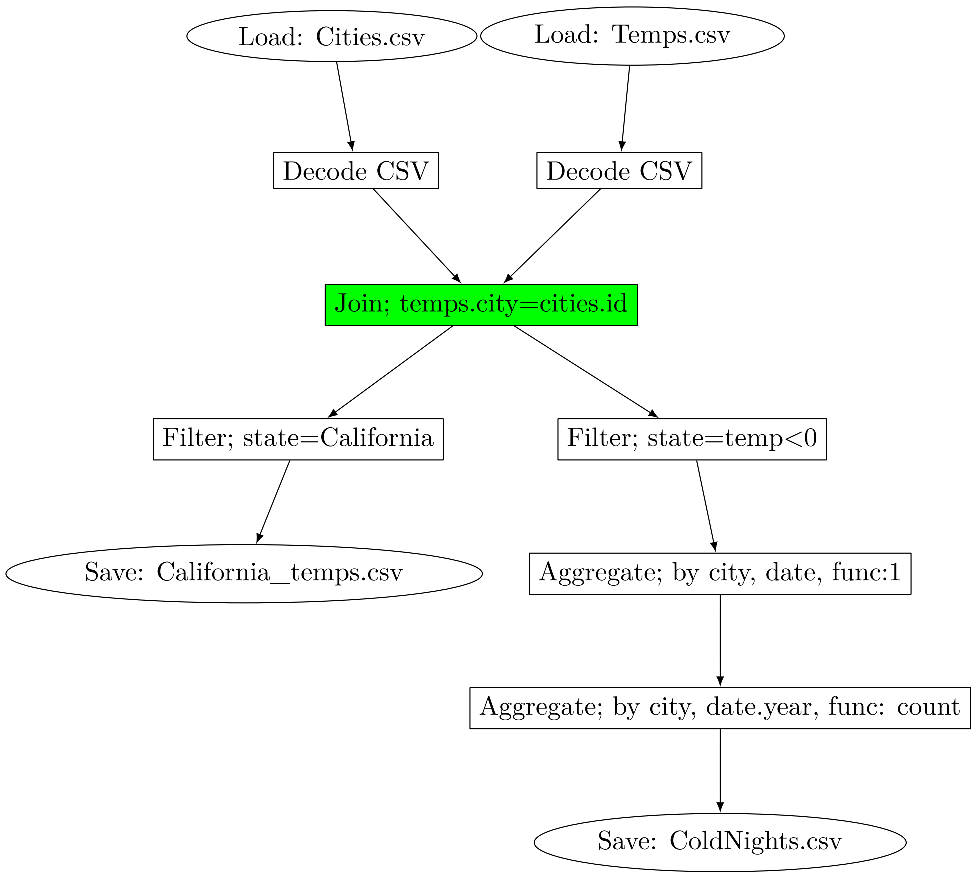

Load: Cities.csv
Load: Temps.csv
Decode CSV
Decode CSV
Join; temps.city=cities.id
Filter; state=California
Filter; state=temp<0
Save: California_temps.csv
Aggregate; by city, date, func:1
Aggregate; by city, date.year, func: count
Save: ColdNights.csv

Figure 2.3: Example of an Apache Spark directed asyclic graph for data processing.

Because Spark uses lazy data-processing it is possible to speed up operations when operating with only partial data sets. While mapreduce in itself does not offer a "first n records" mechanism, Spark uses this actively to speed up jobs. Another advantage over chaining mapreduce jobs is that Spark attempts to process chunks as long as possible and start the next step as early as chunks are available to do it.

An example of how Spark handles queries can bee seen in Figure 2.3. The example illustrated are the join example used in section 2.2, and produces 2 outputs.

– *Partial Result:* Temperatures with city information

  1. Load Cities.csv and Temps.csv
  2. Decode CSV

3. Join them on city ID
4. Save/persist

– *First output:* Temperatures in California with complete city information.

    1. Read from partial result
    2. Filter out records wher city is not California
    3. Store to California_temps.csv

– *Second output:* Number of days where the temperature was below 0, per city per year.

    1. Filter out records leaving those where temperature is below 0
    2. Aggregates by city and date with a constant as output
       This results in a list of days per city where one sample shows a temperature below 0
    3. Aggregate by city and year using the counter function
       This results in the number of days where temperatures were below 0 in a given city per year
    4. Save to ColdNights.csv
       Possibly incorrect assumption that the below 0 sample occurs at night

The join outputs are cached (marked by green circles) so it does not have to re-compute it between data.

2.3.3 Spark SQL

Spark SQL is a different interface to Spark, it uses an SQL-like syntax to generate the same processing graphs and the same operations as the programatic interfaces offer. This interface can also be accessed via Java Database Connectivity/Open Database Connectivity (JDBC/ODBC) interfaces and thus be handled by Business Intelligence (BI) Applications in the same manner as any other database.

There are other projects that offer the same interfaces and connectivity as Spark SQL. Among these are HIVE, a platform developed by Facebook Inc. for making big data processing easier for the programmer than Mapreduce.

2.4 Apache Hadoop

Apache Hadoop was originally developed as an implementation of Google's GFS and mapreduce. Hadoop 2.0 however, is a big data processing framework that allows the user to write any application trough the YARN API.

2.4.1 GFS/HDFS

GFS was designed with the following assumptions: [GGL03]

- The system consists of many unreliable (consumer grade) nodes.
- The system stores primarily large files.
- The main workload consists of either large streaming reads or small random reads.
- Files are rarely modified, but may often be appended.
- Files may be appended to by multiple clients.
- High continuous bandwidth is more important than low latency.

It is not a traditional file system per se because it does not implement traditional file system interface. Instead it implements a file system protocol that is accessed trough the applications that use them. This interface supports several untraditional commands like “snapshot” and “append”. Snapshot copies the file or directory three at low cost by reusing data blocks. Append allows for several clients to write records to the same file at the same time and guarantee that data is not corrupted by the operation.

GFS runs in user-space on standard linux machines, this means that GFS is a application guest just like any standard database server. It uses a single-master design where one node acts as a GFS master and holds an index of all the files on GFS. The single master design simplifies the architecture, but a hot spare node is keep ready to take over should the primary node fail. The master node index keeps track of several chunkservers.

Files are stored in GFS in chunks. A chunk is a part of a file, for GFS they are described as 64MB each. Each chunk is then stored at at least two chunkservers, optionally more. This ensures that if a chunkserver goes down the data is still available in GFS. In the event of a chunkserver going down the master will detect this and ensure that replication is done and distributed properly.

When a client wants to read a file it queries the master and the master replies with chunk information. The client then uses the chunk information to query the chunkservers directly for the chunks. The client would typically choose the closest chunkserver to load the chunk from. If the client is an application that runs on the same nodes as the GFS cluster it can distribute the workload across the nodes and process the chunks locally in order to minimise network traffic.

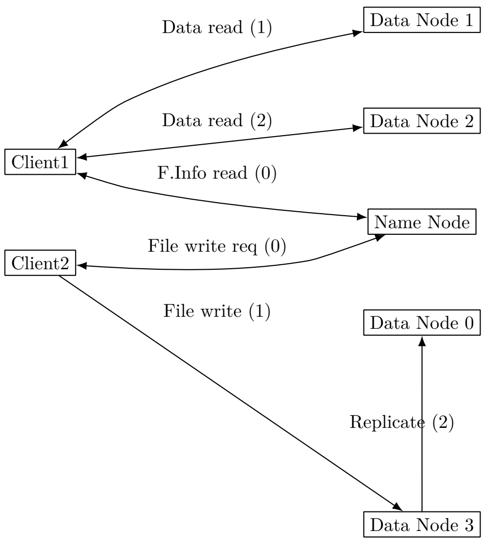


Figure 2.4: Overview of different participants in a HDFS system. Client 1 is doing a read of a file. Client 2 is writing a file chunk, replicated to Node 0.

Writing operations and replication is handled internal to GFS. The client only asks for a chunkserver to write to and that chunkserver handles replication onwards. This means that bandwidth usage is distributed, instead of having the client replicate the chunk written the chunkserver redistributes to the second copy, the second copy distributes to the third and so on.

HDFS is the implementation of GFS within Hadoop. It has a slightly changed terminology, but still uses the GFS paper as a design document. All the concepts described about GFS is true for HDFS.

2.4.2 YARN

YARN is a workload distribution model created for Hadoop 2.0. While Hadoop 1.0 was a Mapreduce platform Hadoop 2.0 is a general compute resource platform. YARN is the model for managing and distributing the workloads across a hadoop cluster [VMD+13].

There are three roles in a YARN cluster. First is the Resource Manager that administers the cluster and functions as the resource arbiter. Second is the Application Master that is in charge of coordinating the application. The last role is the Node Manager, it is responsible for managing a noce in the cluster and allocate resources for workloads.

YARN can be used to host both Mapreduce, Apache Spark, and other distributed applications. The primary reason to use Hadoop [1] is to run HDFS with YARN based data processing applications on top, so the assumption that this the only use case will be taken.

2.5 Apache HBase

Apache HBase [Geo11] is an implementation of Google's BigTable. It is designed to run on top of Hadoop and HDFS on the same nodes [Tay10]. It is one of the more commonly used architectures. It's a so called sparse wide column store. This means that it is able to store a lot of data in each column/feld, and has capabilities to handle a lot of empty columns easily.

The mechanism for storing the data is a "multidimentional sparse map"[Tay10]. Each cell in this map is identified by its table, row, columnfamily and column plus a timestamp. This timestamp is used to settle disputes about values, but are also used to store previous versions. As with HDFS, HBase duplicates data to keep redundancy, and to manage the database itself it uses a technology called Apache Zookeeper, not discussed in this report.

For accessing HBase one need to access it by the primary key. It is not possible to access data based on anything that is not the primary key. It is however possible to create secondary tables with either mappings to the main primary key, or the entire dataset with a different primary key. Additionally, there are no spesific query language for HBase, just an API.

The term columnfamily is an expression used to denote a group of columns that will be stored together. It can be used as a form of hierarchical organisation system, but the real use is that they are stored together and thus can be retrieved together.

2.5.1 BigTable

Google presented BigTable a the OSDI conference in 2006. The structure is inspired by, and for the purposes of this explanation, identical to that of HBase. The access to BigTable uses a API that allows for scanning, sequential reads, and random reads.

The BigTable paper states that BigTable is used as a back end for several services like "web indexing, Google Earth, and Google Finance"[CDG+06]. This a significant span of service types and requirements. The authors claim that because BigTable performs well on all these services, a BigTable like system has a wide range of use cases.

[1]Within the scope of what will be covered by this project

2.6 Apache Ranger

Apache Ranger is a security application/framework for Hadoop [hora]. Most of the information here is retrieved from Hortonworks [hora], a company specialising in Hadoop ecosystem solutions and providing support to them.

Ranger works as a plugin to the different applications and products used. For instance the plugin to HDFS provides file-level access control to the files in HDFS. The auditing component keeps track of who has accessed what and how on the system. The same goes for the HBase and HIVE modules, except they are able to provide column-level access control.

Ranger explicitly states that it does not provide UNIX access control on the underlying systems. So it could be possible for a malicious entity to use the file system as a back door. If used in conjunction with transparent data-at-rest encryption the dangers of this threat is minimal.

Ranger also provides its own alternative to the default key-management system for data encryption in Hadoop. The claim is that it uses a more secure storage mechanism at the cost of some extra complexity in the application.

Ranger is a traditional access control system for Hadoop. There may be some crypto-key access in the system, but there are little documentation indicating this. In either case, this should not be too hard to add as Ranger manages the keys and the policies.

2.7 Apache Kafka

Apache Kafka [KNR+11] is a distributed messaging system developed at LinkedIn [KNR+11] for their internal use and open sourced. The idea was to address some scalability problems with then existing messaging systems like the specifically mentioned ActiveMQ, IBM Websphere MQ, and Oracle Enterprise Messaging Service. A message passing system or messaging system is a system for managing message queues. One or more services can send messages while others can consume them. Example usages of this is Celery [cel], a system for distributing work tasks to different workers. Apache Kafka is intended for passing large amounts of log messages to different systems that process them; others may be suitable for passing messages between systems for coordination.

The first concept in Apache Kafka is the “stream”. A stream is a channel of messages that are produced and consumed together. A “producer” inserts messages into the stream and a “consumer” reads them from the stream. A stream may have one or several “partitions”, that are used for sectioning distributed topics. The systems that work in between the two are called “brokers”.

Unlike the messaging systems mentioned earlier, Apache Kafka uses more logic in the consumer and the producer. For instance the Apache Kafka broker does not keep track of what messages each client has received, that is for the consumers to keep track of. The other advantage is that since a Apache Kafka Subscriber uses a pull model rather than a push model, there is no risk of the consumer not being able to handle all the messages being sent to it, or logic to prevent it.

Some other messaging systems like ActiveMQ were designed with the goal of minimising delay. This is not the case in Apache Kafka, the design in Apache Kafka was made without delay in mind, and then optimised to minimise the delay in implementation. For instance to maximise performance, new messages are flushed to disk only after a certain number of messages or time has passed, this optimises speed in the number of messages processed, but it introduces the buffer-time as a delay. This also integrates into the design choice to leave caching in memory to the operating system’s file system cache rather than maintaining one of their own.

Another advantage that makes Apache Kafka faster is that there are no explicit message id. Instead the file offset to the start of the message is used as an identity. When a consumer requests a new batch, the consumer sends the offset and a number of bytes to accept; the broker then sends a number of messages to the consumer up to the number of bytes requested. The consumer calculates the next offset by counting the bytes in the messages. This also means that the consumer always consumes sequentially.

Because the broker does not keep track of what consumers has read what data, it also has no way of knowing if it is safe to delete a record. Apache Kafka’s solution was to use a time-based SLA. This means that the broker has an implicit permission to delete messages that are over a certain age.

In terms of parallel consumption Apache Kafka uses consumer side logic. A consumer may be a member of a consumer group. The group internally keeps track on who consumes from what partition of what topic and keeps track of the last consumed message for each partition. The coordination is done using Apache Zookeeper. The consumers keep track of each other and using Zookeper’s consensus features distributes and re-distributes the load as consumers are added or removed.

The delivery guarantee in Apache Kafka is at-least-once, this means that if an error occurs during processing, another consumer may resume consumption from a point prior to the last processed message. As this only occurs when critical errors, most systems that has tolerance for more than zero duplication should be fine. E.g.

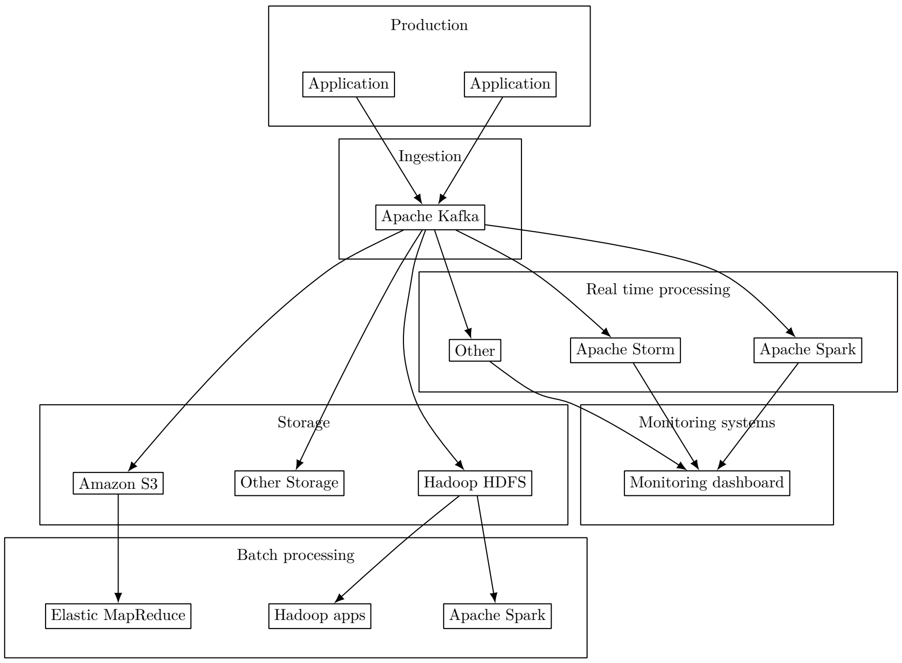


Figure 2.5: An abstract overview of Telenor Digital's Big Data architecture. Data flows in the direction of the arrows.

If the application is a statistics application the result may be one or two extra records counted in the interval where the error occurred, it may not be a big deal in the long run.

2.8 Telenor Digital's Architecture

The basic structure of Telenor Digital's architecture is outlined in Figure 2.5. This architecture is fluent, and all of the specific products may be replaced with other ones and some are only hypothetical. All the conclusions reached in this report are intended to be sufficiently generic to apply to all similar systems.

The major element of the architecture is the usage of a real-time processing system. The real time systems have different requirements for encryption/decryption speeds than a batch system. This will influence where access-control encryption can be applied for some designs.

In terms of how typical this is, the architecture follows the data lake architecture (section 2.1), but with an added real-time component. There are very few specific systems here that have properties relevant for later use, so it can be considered a standard hadoop architecture.

The architecture as described has six major stages. These stages are different classes of systems. For simplicity in the drawing the extraction stage has been left out, it is not that important for these purposes. Because some systems like Spark can fill multiple roles they can appear in multiple stages.

- The production stage are the applications emitting these logs. The logs could be in the form "(timestamp) user x paid y for z" or in the more structured form "{time:(timestamp), user:x, amount:y, product:z}". There may be adaptor applications that are used to interface to Apache Kafka. In other use cases the data might be the primary effect of the application or it might be data produced in batch.
- The ingestion stage is where data is sent into the bg data system. For Telenor Digital this is is in majority done by Apache Kafka, but there are other similar systems out there. A description of Apache Kafka is available in section 2.7. In essence it acts as a buffer between the producer and the storage system or batch processing systems.
- The real time processing stage is where "live" stats are generated. These are passed on to the monitoring systems in the Monitoring system stage. It is possible to pass real time data on to the regular storage system. This would work just like the normal storage ingestion; in this system that path is not drawn.
- The storage system is where data is retained for longer periods of time. While Apache Kafka can act as a buffer for a week the storage systems are designed to keep the data for months or years. The main storage system for most systems are HDFS. There are other systems for storage available, and because Telenor Digital is a user of AWS Amazon S3 is included as a storage mechanism. To get from Apache Kafka to the other systems, an adaptor application may be required. While these adaptor applications are not drawn in, they can be seen as a prat of the vertex (arrow) between the systems.
- In the batch processing system the queries on data is done. The systems read from storage, does the processing, and outputs the data. The output may be a sent back to storage, or to another system. The focus is going to be on Hadoop Applications, but other applications follow the same basic idea. For Amazon S3 storage Amazon offers a product known as Elastic MapReduce, this is a product that provides a Hadoop cluster with Amazon S3 as its storage system [amab].

– Not pictured is the extraction system, while there are many, including adaptors for Apache Kafka, the extraction system is not that important.

2.9 Superscalar Processors

While a Central Processor Unit (CPU) technology may not seem like something relevant for cryptography; it might be relevant for the performance aspect. This is only going to be a very basic introduction for understanding the results in section 7.7. For a more in depth explanation there is a talk from 2014 [God14] outlining a lot of what goes on inside a modern CPU. Superscalar architectures used to be a big deal in high performance computing [Dra93] and optimisation; today Intel's x86 dominates the market and makes superscalarism the de-facto standard.

2.9.1 Introduction to superscalarism

A superscalar processor is a sequential processor architecture that logically executes instructions in order. Internally however, the superscalar architecture re-schedules instructions that do not depend on each others result to be executed in parallel. "As a consequence, the order in which results are produced with this type of architecture is not necessarily the same as the order of the instructions in the program."[BD13, chapter 13] The ability to execute instructions in parallel depends on the instruction not using output data from an instruction that has not been executed yet. There are a lot of research on optimal algorithms for instruction level parallelism [ABKN16].

Branch predictors are not perfect, so it happens that a CPU guesses incorrectly on the direction a branch takes. If the superscalar architecture has executed a branch incorrectly it will simply discard the results. This brings back the statement that the architecture "logically executes instructions in order". For anything external to the CPU the CPU will appear to execute the instructions in the program order.

Chapter 3
Data Lake Security

The previous chapter introduced a lot of technologies were introduced. This chapter covers the academic research on security with these technologies. Most of the technologies described here are not implemented in a way that is feasible for use in production, but some of them are used in products like Apache Ranger/Sentry already, either as a result from this research, or by convergent research by the developers.

3.1 Big Data Access Control

Access control in big data does not seem to be a popular subject in academia. There is some research, but the majority of solutions available fall into a few categories that are well established in the "small" data fields.

File or directory level access control Systems like Apache Ranger (section 2.6) uses a form of middleware that issues an error if the program accesses files it has not been granted access to. This is similar to how multi-user operating systems handle file-system access control.

Access control in table-like data systems Another popular model is used on table-like systems and offer more granularity. This comes from the inherent data awareness of table-like systems like Hive and HBase. Depending on the implementation the granularity may be on a table-by-table basis, per column, or even based on the content. Exactly how the access control is enforced varies. Some may issue errors on attempting to access restricted data, others treat data that the user/application does not have access to as non-existing. The big differentiator between solutions in practice is how they do the access policy mappings.

The real problem with big data as opposed to "small" data is that some schemes do not scale properly. The above mentioned file/directory level systems scale pretty well because a file/directory is a pretty big attribute. For the table like systems the implementation can be done either in the query scheduler by checking access to the table/column or it could be implemented by injecting filter conditions into the query.

3.1.1 Research/theoretical

In the academic fields there are some interesting developments like the introduction of a policy mechanism called Content-based Access Control (CBAC) [ZYL13] that grants access to content with similar properties to the users own content. This particular idea is not as relevant as it may seem at first. The content based access control mechanisms for the use cases relevant for this report can usually be implemented by injecting filter-conditions.

Another interesting paper is about updating Attribute-Based Encryption (ABE) policies [YJR+14]. The main problem they were addressing was how to securely outsource the policy updating to the cloud server rather than having to transfer the data back and forth for a policy update. If a solution using ABE is used, this paper is very relevant.

A 2014 paper by a group from Moscow Engineering Physics Institute [MSTZ14] tries to address the problem of information security in big data by writing a set of guidelines for how to set up the architecture. An example of a guideline is “It is a good practice to uninstall unneeded services (for example FTP) and to have a timely software and operating systems patch management process in place, backup services and backup encryption.”[MSTZ14]. The entirety of the guidelines can be crudely summed up with the paraphrasing “Use firewalls, reduce attack surface, and do lots of audit logging especially in the firewalls”

A couple of researchers from Temple University Philadelphia published an article in 2015 [HD14]. They outlined a image scrambling algorithm that supposedly achieved 390-590 times faster “encryption” compared to Advanced Encryption Standard (AES). These numbers may be misleading as it seems Matlab’s AES implementation does not report to utilise the AES-NI instructions set [Gue12]. Even so the numbers are impressive.

A 2014 paper by Bertino [Ber15] outlines the problems with scaling to big data. According to the paper the major challenges that require research in the field of security are scaling policy systems to big-data levels and automating policies and management of the dynamic environments. On the integrity side there are existing schemes, but a call for more tools for using data reliability in processing is made. In the field of privacy there are several fields that need to be addressed, including scaling of encryption and a “Privacy-aware Data lifecycle framework”[Ber15].

An article from 2014 [GHMJ14] discusses how big-data can be used in healthcare solutions. It focuses on the privacy and security issues. They only bothered trying to protect tabular data in tabular systems. The main focus is so healthcare specific that it has little applications here.

3.1.2 Sticky policies

The concept of sticky policies were introduced in 2002 [KSW02]. Sticky policies are access control policies provided by the end users (or an agent on their behalf). The exact mechanism is not specified in their paper, but it is possible to use a ciphertext-policy ABE to implement it. The idea is that the use policy is attached to the encrypted data and the encryption enforces the policy. The policy follows the data around and across multiple vendors so there is one policy framework across all the vendors.

An implementation of sticky policies called EnCoRe was done in 2011 by a team at HP [PM11]. They used among other things, Public-Key Encryption (PKE), Identity-Based Encryption (IBE), Attribute-Based Encryption (ABE), and Proxy Re-Encryption (PRE) to achieve this. Their model looks good at first glance, but as shown in chapter 5 ABE is too slow to be used on individual records in data lake sized environments.

An implementation in 2015 [LZGP15] provided and confirmed most of the formulations used above. It also provides an implementation that uses a policy enforcement agent rather than using ABE. This was due to the implementation focusing on Big Data. No benchmarks were published of their framework so there is no good data on this.

3.1.3 Frameworks and Whitepapers

In 2016 Google presented a system called GOODS that creates and maintains metadata internally at Google [HKN$^+$16a, HKN$^+$16b]. The goal is to make the datasets themselves searchable and maintainable. GOODS has similar properties to Apache Atlas, and given the record of Apache big-data tools (e.g. Hadoop, HDFS, mapreduce, HBase) it is not unlikely that future Atlas development will draw inspiration from Google's papers.

In 2010 a team from University of Texas developed a system they called Airavat [RSK$^+$10]. The system intended to add privacy to big data architectures by adding access control and tracking of sensitive data files. It has several limitations, the foremost of these is that it only works on a modified MapReduce. Another drawback is that it uses modified versions of Java Virtual Machine (JVM) and HDFS and these modifications would require maintenance. There are no signs of activity on this after the publication in 2010, so porting Airavat up to recent versions of Hadoop could be as big a task as implementing a new system that support other frameworks, not just modified MapReduce on modified JVM and HDFS.

In 2016, Intel released a whitepaper on how they secured their data lake architecture [ODS13]. They used a product called Apache Sentry and Cloudera Navigator. From the published information [cloa, clob] this combination promises similar functionality to Apache Ranger with Apache Atlas [hore]. Because Cloudera Navigator is a commercial product minimal focus will be devoted to it.
One thing that should be noted, Apache Sentry and Apache Ranger are developed and promoted by Cloudera and Hortonworks respectively.

3.1.4 Problems with Traditional deny-access schemes

Deny access schemes are any access control scheme that uses some sort of middleware (like the operating system) to check the access policy and simply not allow access unless the requirements are meet. These systems may have bugs in them that may be bypassable.

An example of a complex deny access system is Security enhanced Linux (SELinux). It offers advanced policies beyond what regular UNIX offer by also including context in the policy. The complexity of SELinux also means that it may contain bugs that could let the user going access to data they should not.

In 2016 and 2017 a few bugs allowed for partial or complete bypassing of SELinux' policy mechanisms.

CVE-2016-5195 [CVE16a] Dirty COW, allows for privilege escalation without SELinux ever being involved. The escalated privileges can then be used to effectively gain full access. As far as SELinux is aware, the process has access to the files it is trying to access.

CVE-2016-7545 [CVE16b] A bug in SELinux that allowed the user to execute arbitrary commands outside of their sandbox.

CVE-2017-6074 [CVE17] Another privilege escalation bug unrelated to SELinux, but usable for bypassing.

In contrast a crypto based access control mechanism would remain secure even if the enforcement middleware is bypassed. The assumption made here is that the cipher is secure, the keys are securely generated, and securely stored. Even if the attacker was able to bypass the middleware, the only thing they would gain access to is the ciphertext.

A key management system is effectively a deny access system, but a remote key management system would offer fewer attack vectors to work with and effectively make it earlier to keep secure than a local deny-access system.

3.2 Data Reservoir

A data reservoir is a concept described by IBM and ING in 2014. [CSN+14] The starting premise was that uncritical use of a data lake would turn it into a Data Swamp. A Data Swamp is a data lake where the data is unreliable and untraceable. As the name implies the data from a data lake is not “fit for consumption”, and can even be harmful if used for business purposes.

The paper does not describe how a data reservoir is implementationally different form a well maintained, governed, and audited data lake. The main focus seems to be on the use cases for businesses rather than how it differs form a data lake. For more information, IBM’s book “Designing and Operating a Data Reservoir”[CJL+15] is recommended reading for designing a General Data Protection Regulation (GDPR) compliant data lake, regardless of technologies used.

3.3 Attribute-Based Access Control

Attribute-based access control is access control schemes that are based on attributes of the user (accessor) and the data (being accessed). The general idea is that the implementation makes a language that can be used to describe access control relationships between the attributes [HKF15].

Examples of user attributes could be: age, department, position, clearance, or nationality. The representation of these attributes are up to the implementation. Examples of data attributes are: owner, department, language, classification level, or creation date.

An access policy could be as simple as “content with classification level X requires a clearance higher than X”, or as complex as the computational language allows.

3.4 Privacy

Several studies have claimed that “Big data has the potential to revolutionise the art of management.”[WAE+15] and “big data provides deeper insights into achieving value through big data strategy and implementation.” [WAE+15]. The study quoted here kind of acts like a meta study in that it reviews a lot of other studies in the value made from big data in a lot of fields.

The other side of this is the privacy issues. An article from 2012 talks about the issues faced by entities that want to release public datasets [MR12]. For instance, the mere "anonymization" of data by replacing names with random numbers are not enough. This kind of replacement is known as pseudomysation, and is now no longer considered enough to mask "sensitive" data. In the article there are several solutions on how to make the data fit for public release. The problem with those methods is that they usually lower the quality of the data for direct insights. For example a phone plan recommendation system looses its value if users are aggregated, data is scrambled, or simply synthesised.

The pseudomysation problem is a well established field of study. For instance a paper by Korayem and Crandall form 2013 [KC13] demonstrates the ability to link social network accounts to the same user. This didn't even require any strong features to map.

Another study from 2013 by Unnikrishnan and Naini [UN13] looked at WiFi clients at the Ecole Polytechnique Federale de Lausanne (EPFL) campus. They used client connection logs (pseudomysed MAC address, Access Point ID, and time) to be able to match devices form the Monday one week to the Mondayof the week after with about 50% accuracy.

Lastly on the de-pseudomysation front is a couple of papers from Narayanan and Shmatikov in 2008 and 2009 [NS08, NS09]. They talk about the loose statistical matching framework that is used for all the de-anonymisation efforts mentioned here. They apply these methods to the Netflix-price dataset and for matching Flickr accounts with Twitter accounts. Their conclusions were that pseudomysation is simply insufficient to establish privacy.

The efforts of their report is on limiting unauthorised use of "sensitive" information internally, and the usability of any illegally leaked data. At the same time, the authorised use of the data should still be made possible. For this reason any permanently destructive measures like aggregation, noise, deletion, data swapping, and synthesised data [MR12] are unusable.

Chapter 4 Cryptography

The two previous chapters covered big data technologies, this chapter will try to cover the different cryptographic subjects that need to be covered for this thesis. Like all chapters this is not an in-depth discussion, but a short introduction to the basic concepts.

Why use cryptography as an access control mechanism?
The advantages of using a cryptographic access control mechanism was described in a paper on a hierarchical crypto-access control system in 1983: "[The] protection it offers against illegal disclosure depends neither on the physical security of the storage medium where the information resides nor on the trustworthiness of the people managing it."[AT83] The solution described in that paper is not as relevant as some other more modern mechanisms.

Some of the mechanisms discussed in this chapter are not crypto access control mechanisms per se, but by controlling access to the keys it is possible to use them as that anyway. None of the schemes below are key-management schemes, but rather schemes for protecting the data in different ways.

4.1 Secret Sharing Schemes

A Secret Sharing Scheme is in the context of this report a system that allows for a secret to be derived if two or more parties come together. How manny parties are needed are determined by the policy and influences the key generation. This section will describe two schemes and how to extend them to make even more complex policies.

4.1.1 Shamir's Secret Sharing

Shamir's Secret Sharing [Sha79] is a scheme based on interpolating polynomials. The idea is that 2 points are sufficient to definitely define a line, 3 are sufficient to define a parabola and so on. It also means that there is an infinite number of polynomials of degree $t-1$that can be defined by $t-1$ points.

As an abstract example, the creation of n parts of the secret S, where t parts are required in order to retrieve the secret the following steps are taken. S is encoded into a tth degree polynomial. The parts are extracted by sampling the curve.

4.1.2 Blakley's Secret Sharing

The Barkley scheme [Bla79] is based on the fact that two non-parallel 2-dimensional lines intersect at only one point. Similarly in a 3 dimensional space three non-parallel (2-dimensional) planes only intersect in one point. The secret is encoded in the point, the parts are the $t-1$ dimensional spaces.

4.1.3 More Complex Structures

Making more complex structures is more or less easy. Any part could be treated as a secret, and have a key-sharing scheme applied to it. This way it tis possible to construct a tree structure. Note that a part can not be reused, so if a user needs to fit in on multiple leaves in the tree, they have to have several parts. This is the basis for the construction know as monotone circuit building [Sti92]. This construction is the basis of the GPSW scheme (section 4.3) and how the circuits made in chapter 5 were constructed.

4.2 CryptDB

CryptDB is an encryption layer that is put on top of RDBMSs in order to enable encryption of the data on it. The reason to include it in this report is because it is relevant to take inspiration form.

CryptDB is implemented as a proxy-server thatoperates as a man-in-the-middle between the application and the database server. It handles encryption of column names and the data. The proxy server has an annotated version of the database schema that it uses to get information on the columns, the encryption schemes, and what encryption keys are used where. The CryptDB proxy then re-writes the queries sent to it to work on the encrypted data. The encrypted result is returned form the server to the proxy, who decrypts it and sends the decrypted result to the client.

The basic premise to take inspiration from is the schemes for encrypting the data in a non-/semi-computable way. CryptDB has a set of encryption modes that can be used to protect the data. The details of the relevant modes are outlined in section 4.7, a list follows:

– Random (RND)

– Deterministic (DET)

– Order Preserving (OPE)

– Homomorphic (HOM)

– Join (JOIN), same as DET, but 2 columns share a key so join operations can be done on them,

– OPE-Join (POE-JOIN), same as JOIN, but using OPE instead of DET

4.3 Attribute-Based Encryption

There are several forms of ABE, the focus here is going to be on Key-Policy based ABE. The property of Key-Policy ABE is that the policy is embedded in the decryption key upon generation. An other option is a Ciphertext-Policy ABE, where the policy is embedded in the ciphertext. A more detailed description is available in the original paper [GPSW06].

The idea of ABE is that of a broadcast encryption scheme but with attribute policies like those in Attribute-Based Access Control (ABAC) (section 4.1). The schemes looked at for the purposes of this report uses binary (true/false) attributes and the operations are an AND/OR tree with the attributes as leaves.

In many ways it is similar to binary and/or tree Secret Sharing Schemes (section 4.1). The main difference is that instead of coming together to create a secret, the key represents the policy, and the attributes represent the parties. The scheme was also designed in a way so that collusion should be impossible. In the words of the authors:

> For instance, if Alice has the key [to] "X AND Y", and Bob has the key [to] "Y AND Z", we would not want them to be able to decrypt a ciphertext whose only attribute is Y by colluding. [GPSW06, 1. Introduction, p90]

The Goyal-Pandey-Sahai-Waters scheme supports delegation. This is a mechanism where a new key can be derived from a users key. The new key must be more restrictive than the old key.

4.4 Hardware Securiy Modules

Some times there are high level security requirements for a system. This could be because of PCI-DSS compliance [pci], a governmental privacy law like GDPR, or a possibly stricter successor to any of them.

Hardware Security Module (HSM) is according to the encyclopaedia of encryption and security "a physically secure, tamper-resistant security server that provides cryptographic functions to secure transactions in retail and financial applications."[vTJ11, p. 254] They are considered to have higher level of security than software applications and usually conform to a standard.

Amazon Web Services offers a HSM service called AWS CloudHSM [amaa] that could be used to avoid having a HSM on the customers premises, and thus increasing latency. It would require the service consumer to trust Amazon's HSMs.

While on the subject of hardware based security modules smart cards should be mentioned as well. Smart cards are a form of security module that can contain a program or be used to hold cryptographic keys (in practice this is also programs). Using a smart card for keeping personal keys is a possibility.

4.5 Cryptographic algorithms

This is a short description of the cryptographic algorithms used in the tests set up in later chapters.

4.5.1 AES

The "Rijndael" crypto-system is also known as AES. It is commonly used and is certified by National Security Agency (NSA) for encryption of classified material and the ciphertext can be treated as "unclassified"[aes]. It was published in 2003 by Daemen and Rijmen [DR98].

It has a structure called a substitution-permutation network that works by "substituting" groups of bits, then shuffling bits around and lastly XORing them with a "key" derived from the encryption key. One of these processes are called a "round". The substitution is a mapping defined in the algorithm.

4.5.2 RSA

The RSA crypto-system is used a lot in asymmetric cryptography applications. It was first described by Rivest, Shamir, and Adleman in 1977 [RSA78]. The crypto-system relies on two large primes and uses discrete logarithms to do the encryption/decryption.

The basic idea is that it is possible to make a system where:

$$\left((m^e)^d\right) \bmod n = m \qquad \text{or rather} \qquad (m^e)^d \equiv m \pmod n$$

The interesting part is that because of some properties of modular algebra the following is also true:

$$\left((m^e \bmod n)^d\right) \bmod n = m$$

Textbook RSA works by publishing e and n as the public key. And d keept as the secret key. To calculate these numbers one takes two large primes. The product of these primes are n, e i usually set to $2^{16}+1$. The calculation of d is done using a version of Euler's totient function and a fitting to e.
In this system $m^e \bmod n$ is the ciphertext.

In practice RSA is implemented in a manner like the one described in PKCS 11 [Lab09] standard. It still useless the same mathematical principles, but has some practical adaptations that makes it more predictable and computer friendly.

4.5.3 GPSW ABE

As a representative for ABE is the algorithm described by Goyal, Pandey, Sahai, and Waters (GPSW) [GPSW06]. The algorithm implements something that looks like the secret-sharing trees described in subsection 4.1.3.

The cipher is a so called pairing-based system. A pairing is a pair of cycles (a RSA system is a cycle) that can be combined to create another cycle system. How this works is too advanced to look into, but the pairing is the method for combining these. The GPSW system uses pairings to create structures for decrypting data. Advanced circuits are built using circuit building similar to the monotone circuit building described in subsection 4.1.3.

4.6 Block cipher operating modes

Block ciphers like AES has different operating modes. These modes influence the performance and some times how much parallelisation is possible. The only interest in operating modes in this thesis is the performance and parallelisation aspect, because in big data this is going to be the greatest factor.

The simplest of these operating modes is the Electronic CodeBook (ECB) mode. In ECB mode each block is encrypted using the same key. This means that if a block has the value m_0 its ciphertext will always be C_0 for the same key. This leaves the ciphertext vulnerable to frequency analysis. This vulnerability makes ECB a rarely used mode.

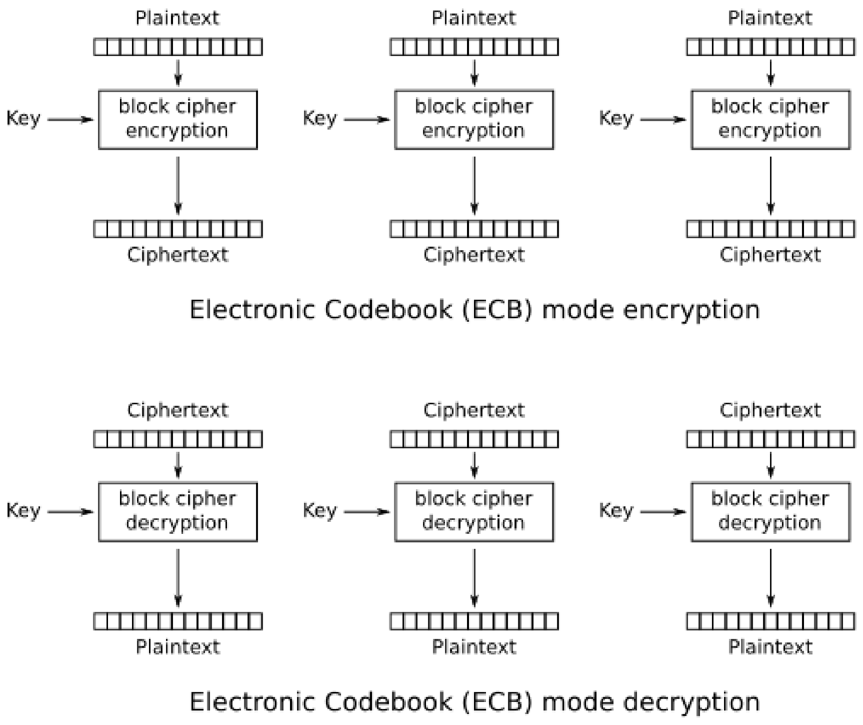


Figure 4.1: ECB mode of operation. [Com13f, Com13e]

Another operating mode is Cipher Block Chaining (CBC) where the plaintext is XORed with the ciphertext of the previous block, and a so called initialisation-vector for the first block [Sch95, chapter 9.3]. This ensures that the same block of cleartext does not produce the same block of ciphertext every time. The ting to take note of is that each block is dependent on the ciphertext of the previous block. This means that during encryption a superscalar CPU is unable to introduce any instruction level parallelism. On decryption, however the ciphertext is available form the start, and the CPU could, in theory, execute all blocks in parallel.

The optimal mode for parallelisation is CounTeR mode (CTR). In this mode the block cipher is turned into a stream cipher. This is done by encrypting the initialisation vector (in this mode referred to as a nonce). The next block encrypts the nonce plus one, the third nonce plus two and so on. Because the nonce does not

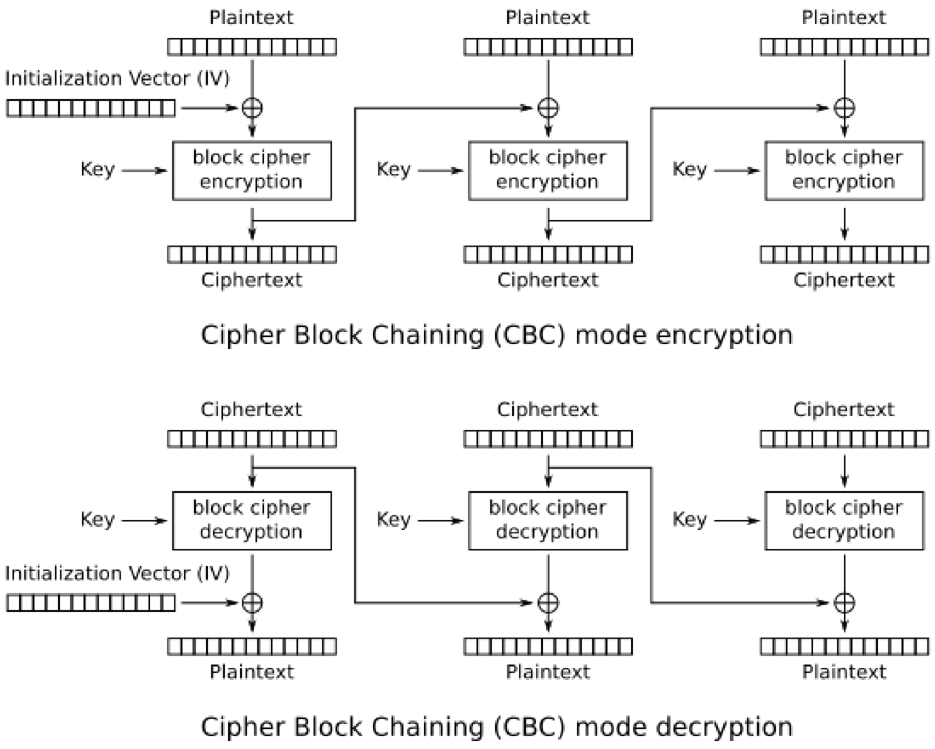


Figure 4.2: CBC mode of operation. [Com13b, Com13a]

change, and every block is independent, this mode is infinitely paralleliseable in both directions. To encrypt a cleartext the cleartext is just xored with the encryption stream. The decryption mode is identical to the encryption mode. This means that CTR mode does not use the decryption part of AES, and is its own inverse.

4.6.1 Overview of operating modes supported in Oracle Java

Because most big data architectures are implemented in Java or on the JVM most tests are going to be performed on that platform. Oracle has described the following block cipher modes [orab].

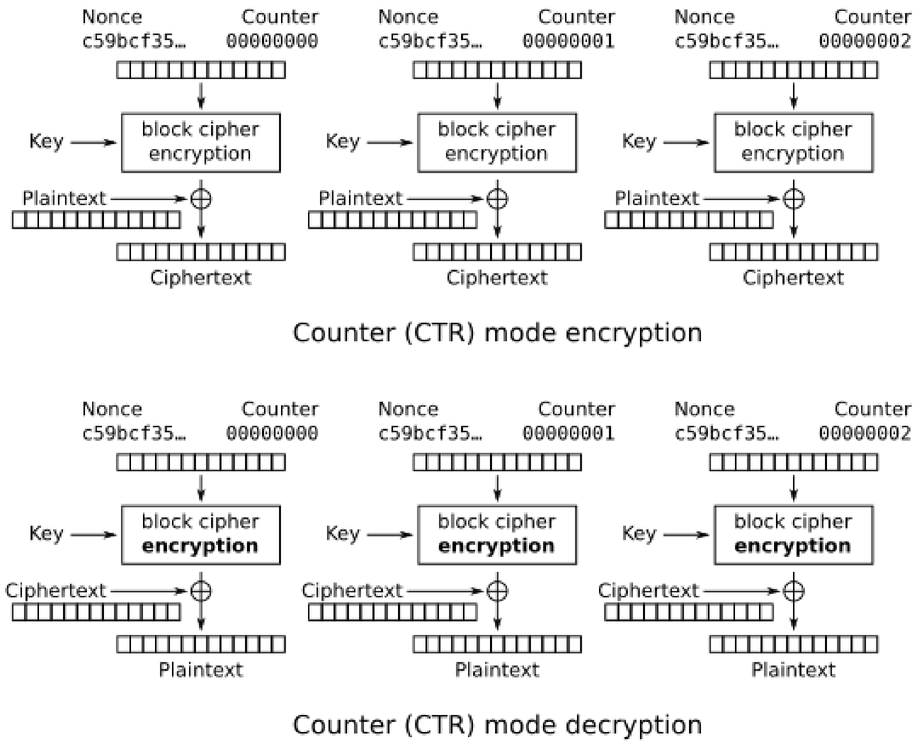


Figure 4.3: CTR mode of operation. [Com13d, Com13c]

CBC	"Cipher Block Chaining Mode, as defined in FIPS PUB 81 [NIS]."
CFB	"Cipher Feedback Mode, as defined in FIPS PUB 81 [NIS]."
CTR	"A simplification of OFB, Counter mode updates the input block as a counter." Described in Figure 4.3
CTS	"Cipher Text Stealing, as described in Bruce Schneier's book Applied Cryptography-Second Edition, John Wiley and Sons, 1996 [Sch95]." This operating mode is not listed in the operating modes section of the book [Sch95]. Additionally the Encyclopaedia of Encryption and Security [vTJ11, p. 791] lists it as a method for making ciphertext the same length as cleartext. Because this mode is not properly documented do not use this mode in production.
ECB	"Electronic Codebook Mode, as defined in FIPS PUB 81 [NIS]." The documentation also states that this cipher should be avoided for multiple blocks. The documentation is right.
OFB	"Output Feedback Mode, as defined in FIPS PUB 81 [NIS]."
PCBC	"Propagating Cipher Block Chaining, as defined by Kerberos V4."

4.7 Analysis of encrypted data

For the purposes of this report analysis of encrypted data can mean one of two things. Firstly it refers to how much an attacker can discern from the ciphertext. Secondly it refers to computations that can be intentionally done on ciphertext.

The former is undesirable for sensitive data, as it partially or even full defeats the privacy purpose of the encryption. Schemes to defeat this exists and will be described later in the section.

The latter may be desirable for some data, but at the cost of granting the same ability to a hypothetical attacker that has obtained encrypted data. Again, schemes for this are used in CryptDB [PRZB11] and a summary of their description follow.

4.7.1 Fully Scrambled

Data that is scrambled completely have no computational value in encrypted form. This can be achieved by padding the data up to a certain length and by randomising an initialisation vector in the cipher block chaining mechanism. This will minimise the leakage of information in the length of the data being encrypted. Because the initialisation vector is randomised it is harder, if not impossible, to find duplicate values in the ciphertext.

4.7.2 Deterministic

The deterministic method means that data is encrypted in such a manner that same value gives same ciphertext. This allows for comparison and aggregation on exact match values. In GDPR ciphertext would be treated as pseudomysed. It is still possible to mask length by deterministically padding the data to a certain length. The limitation of using a padded text is that every record is going to end up as the maximum length text. A detailed description of how and why is available in section 7.8.

It is also possible to use ciphertext versions of data to perform filtering for specific values. This could minimise the time cryptographic keys spend in memory on the cloud platform.

A paper by Bellare, Boldyreva and O'Neill in 2007 discusses a scheme for deterministic searchable encryption using public key that they call Efficiently Searchable Encryption (ESE) [BBO07]. They also discuss the privacy issues with these schemes and when they should not be used.

4.7.3 Order Preserving

Order preserving encryption is an encryption method where the ciphertext comes in the same order as the cleartext. This allows for sorting data in encrypted form. It is also possible to do range queries by supplying the ciphertext of the limit.

This scheme allows for more computation to be done on ciphertext, but it also makes leaked ciphertext at a greater risk in case of a breach. If the value range and number of distinct values in that range is limited it is possible to obtain all the values if the constraints are known.

4.7.4 Homomorphic

The homomorphic scheme has properties that allows for calculations like $C(a+b)$ or $C(a \cdot b)$ without having to decrypt them. Homomorphic schemes allows for several calculations to be done on the encrypted data.

An attacker could look at the data and by calculating manny groups of $C(a+b) = C(x)$, if there are a sufficient number of pairs it would eventually produce collisions that can be used to guess the data. The number of pairs required would go up drastically if the specific scheme introduced an element of noise to the ciphertext.

4.8 Schemes for access control

There are a lot of strategies for how to do cryptographic access control. This section covers some of the strategies for encrypting data for access control.

4.8.1 File-by-File encryption

Description

The per file encryption scheme uses one key per file (or record) the data is encrypted on ingress. Each file, or record would have a header added to them with a encryption key for the file/record encrypted using euther asymetric cryptography or symmetric cryptography.

Advantages

– The advantage of using asymetric cryptography is that the ingester does not have access to the decryption key, so a compromise of the ingestion processor would result in a limited data compromise.

– The advantage of using symmetric cryptography is that he scheme is faster than asymetric cryptography, especially for high throughput applications.

– Because the data is encrypted in raw form there is no need for the ingester to know anything about the data.

Disadvantages

– No columnar access control.

– For the symmetric cryptography variant a compromise of the ingestion node is a full compromise of a decryption key.

Limits on analysis of encrypted data

– Because the length and structure of each record can not be guaranteed it is impossible to do any sensible computation or analysis on the data.

– To maintain HDFS append capability the encryption must be done in chunks. These chunks could contain several records of data, but should have a limited length.

– For processing purposes the chunk boundaries should be on record boundaries. The consequences of misalignment might be that a record get split across batches in processing.

– If per-record encryption is used, and the records are encrypted in one record chunks, it is possible to infer some information form the length of the record.

Access control granularity

– The granularity of access control in this scheme is "all or nothing", one key will grant access to all data in that file until the key is rotated.

– A mechanism could be employed to grant access based on certain properties of the data, but it might be just as easy to direct and duplicate the data to different files.

4.8.2 Columnar encryption

Description

Encrypting the data one column at the time allows for a column by column access control scheme. The main idea is inspired by CryptDB [PRZB11], described in section 4.2. Even if column level access control is not required, it is possible to do partial computation on select encrypted columns of data. In this context a column is a field across several records. Because ordering can not be guaranteed each record needs to be self contained, but it is possible to use the same key for all instances of field A.

This scheme can work on "sparse table" architectures where the datastore has many columns and many rows, but each row may only have a subset of the columns.

Advantages

- The scheme is fast and it is possible to do partial computation on encrypted data by using encryption schemes with some/limited homeomorphism.
- Column names can also be scrambled if required. Making it harder for an attacker who has obtained encrypted data to guess what the different columns mean.
- This scheme can also be applied do any database like architecture in a way similar to how CryptDB (section 4.2) does it.

Disadvantages

- The ingester is more complex because it has to separate out the "columns" of the data record and encrypt each of them with a different key that it needs to have access to.
- If asymmetric crypto is used in order to protect the decryption keys one either looses the homeomorphic properties or one looses performance.

Limits on analysis of encrypted data

- The advantage of copying the CryptDB idea is that it is possible to use the same computation and processing mechanisms [PRZB11].
- CryptDB has schemes for all the levels described in section 4.7. A columnar scheme would be able to apply all of these schemes and therefore allow complete protection of certain columns while partially protecting others.

4.8.3 Attribute-Based Encryption

Description

Attribute-based encryption is an encryption scheme described in section 4.3. The concept of ABE was derived with the intention of creating a crypto-based access control mechanism. The promises given for the Goyal-Pandey-Sahai-Waters [GPSW06] scheme is impressive. The requirement/promise structure is a very good match for several cryptographic applications.

Advantages

- The scheme can be used in a per-column, per-row, or per-column-per-row faction.
- The access control granularity is practically unlimited, and it is all done using cryptographic schemes.

Disadvantages

The scheme is slow and presumably becomes slower with more attributes. This means that this option might be discarded on the basis of performance/throughput. A test of throughput is made in chapter 5.

Limits on analysis of encrypted data

It is possible to read information from the attributes themselves. This could be a source of information leakage that could be exploited by an attacker.

4.8.4 Combining methods

It is possible to combine methods in order to gain the strengths of several methods. An example is outlined below.

A per-file or per record based method could be used for the ingestion stage. The data is quickly processed and saved in encrypted form. This would probably be best if used with a asymetric cryptography method to minimise the impact of an ingester compromise.

At a later stage the file can be split into a column based encryption scheme. This means that the decryption and encryption keys only stays in memory of a targetable machine during this process, thus the attack surface is minimised.

Chapter 5
Benchmarking the Algorithms

Different cryptographic algorithms have different properties. One of these properties for instance the Rijandel cipher also known as Advanced Encryption Standard (AES) [DR98] was designed to be able to be run effectively on lightweight processors. The practical difference is in encryption speeds. To get an overview of the different speeds offered by different ciphers, a test for comparing them is done in this chapter.

Asymmetric ciphers are slower than symmetric ciphers[1]. For this reason the asymmetric ciphers in this test are not used to encrypt data, but rather a small block of data representing a encryption key for another cipher. Therefore the comparisons between asymmetric ciphers and symmetric ciphers are to be able to compare the performance impact of storing the key for a given amount of data using a asymmetric cipher. Note that all Attribute-Based Encryption (ABE) schemes to date are, and must be asymmetric ciphers, and it is unlikely that a symmetric ABE scheme would ever be made to work.

5.1 Setting

The test bench is a Raspberry PI 2 Model B Version 1.1 [Rpia].

It has a “900MHz quad-core ARM Cortex-A7 CPU” and “1 GB RAM”. The tests were carried out on a fresh installation of the Raspbian variant of Debian [Rpib].

The reason for choosing this setup is to eliminate as much interference from any background processes as possible. Unfortunately the ARMv7 instruction set does not have a AES-NI-like instruction set, and Oracle’s java implementation does not utilise it even if it had. Wikipedia claims ARM offers a AES assistance module (on the AES-NI page), ARM’s product list does not have such a product [arm].

[1]While it is possible to make a slower symmetric, cipher, none of them are in common use

The tests were carried out over a period of two to three months while the Raspberry Pi itself were not connected to the internet. Even though it was checked in on irregularly, no keyboard, mouse, or screen were connected to the machine for most of the runtime of the program. All of this to minimise random factors in the performance of the tests.

5.2 Methodology

The results of these tests are not intended as an apples-to-apples comparison. The intention of these tests is to be able to make a comparison of different crypto-systems to estimate performance impact and where the data is useful.

As a baseline algorithm AES with a 128 bit key was chosen. This will serve as the minimal performance impact possible by encryption. This assumption relies on the AES implementation using a AES instruction set on the Central Processor Unit (CPU) [Gue12]. The testbench does not support this instruction set so the comparative difference in speed is going to be greater on a production system with an x86 CPU if properly set up.

5.2.1 Benchmark suite

The benchmark is done by a Java program that has a collection of tests, run them in randomised order, and prints out the results.

Each test implements an interface used to run the tests and print information about the test.

```java
public interface BenchmarkEntry {
    /**
     * Run the timed test
     *
     * @return
     */
    boolean run();

    /**
     * Prepare for timed test.
     *
     * This is where keys are generated, mock data
     * allocated etc.
     * This is called just before the test.
     *
     * @return true iff no errors occurred
     */
    boolean setup();
```

```
/**
 * Clean up after timed test.
 *
 * Clean up after the test, to save memory.
 * This is called just after the test.
 *
 * @return true iff no errors occurred
 */
boolean cleanup();

/**
 * Name of test for output
 *
 * @return a name for the test
 */
String name();

/**
 * Variation/variable value
 *
 * @return the value variable of this particular
 * test
 */
```

The main program generates up the test collection. It re-uses instances to save memory.

If any test fails it will get re-queued and run again at the end of the list. This will repeat for a number of attempts until the suite has made $2n$ tests where n is the number of tests in the original list. A successful test will get removed from the cycle.

All tests has been set to repeat for 1000 cycles. To avoid key generation slowing things down the cycles use the same cleartext, the same keys and the same ciphertext every time. This means that if Java optimises its systems by caching results (called memorisation), all these tests are invalid. These kinds of optimisations does not seem likely because Java do not have a mechanism for telling the optimiser about deterministic functions.

The code used for these tests are included in Appendix C

5.2.2 GPSW ABE

The GPSW cipher is described in subsection 4.5.3.

For these tests AND gates are emulated using $2of2$ nodes. Or gates could be emulated with $1of2$ nodes, but the specific implementation uses the threshold 1 to indicate a leaf node.

The implementation of this algorithm is one done by Liang Zhang [Zha14]. This implementation was in that project found suitable for a health record encryption. For big data there is a different requirement for speed. Note that it is assumed that the implementation is correct, at least in terms of computing power. No major effort has been put into checking the implementation's correctness.

This implementation uses a library for pairing based encryption written by De Caro and Iovino [DI11]. This library does the major heavy lifting of the implementation. The impression of this library is that it is suitable for academic work, not yet for production.

Another candidate that were considered was De Caro's implementation of a system by Brent Waters that uses a regular language as policy [Wat12]. This implementation was dismissed on basis on it being hard to automate variations and the preliminary run times being too long to even be considered.

The implementation uses the GPSW algorithm to seed the generation of an AES key. In the tests this AES key is used to encrypt/decrypt 32 bytes of data (one 256 bit AES key). This is a redundant way of doing things, but it it allows for a self chosen key to be used.

The tests for this algorithm has three variables: The number of attributes in total, the number of attributes assigned to a message, and the number of attributes used in the decryption key. Each of these variables have a separate test set where the other variables are kept fixed.

In the tests varying the total number of attributes in total the master key varied between 6 and 57. The number of attributes assigned to each message was one for encryption and two for decryption. The number of attributes in the decryption key was set to one.

In the tests varying the number of attributes in a message the number was varied between 6 and 57. The total number of attributes was set to 101. The number of attributes in the decryption key was set to one.

In the tests varying the number of attributes in the decryption key the number was varied between 2 and 56. The number of attributes in total was set to 101. The number of attributes in each message was set to 100. The key was built using a simple AND tree as seen in Figure 5.1.

5.2.3 RSA

The RSA algorithm is described in subsection 4.5.2

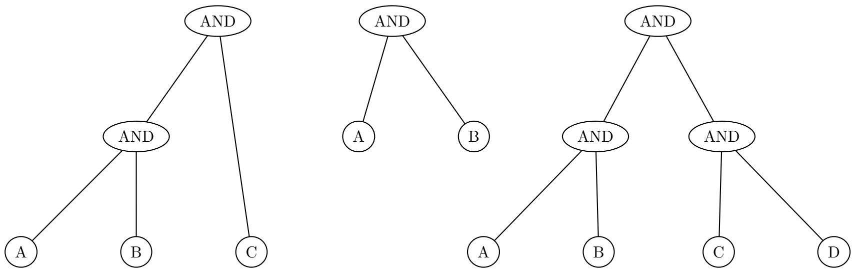


Figure 5.1: Examples of how AND trees are built for the tests, using 3, 2 and 4 attributes respectively

For these benchmarks the key lengths used are 1024, 2048, 4096, and 5120 bits. The length of the data to be encrypted is 32 bytes or 256 bits.

The implementation used is Java's built-in implementation of the crypto-system.

5.2.4 AES

The AES algorithm is described in subsection 4.5.1

The implementation used it Java's built in implementation with Cipher Block Chaining (CBC) and PKCS5 for padding. This implementation should use the AES instruction set [Gue12] for encrypting the data [Koz] and thus be very fast.

The variable for this test is the size of the data to be encrypted. The tests are using a base number of 64 Bytes and bit shifting it between 1 and 19 bits. The result is every factor of two between 128B and 32MB.

5.3 Results

5.3.1 GPSW ABE

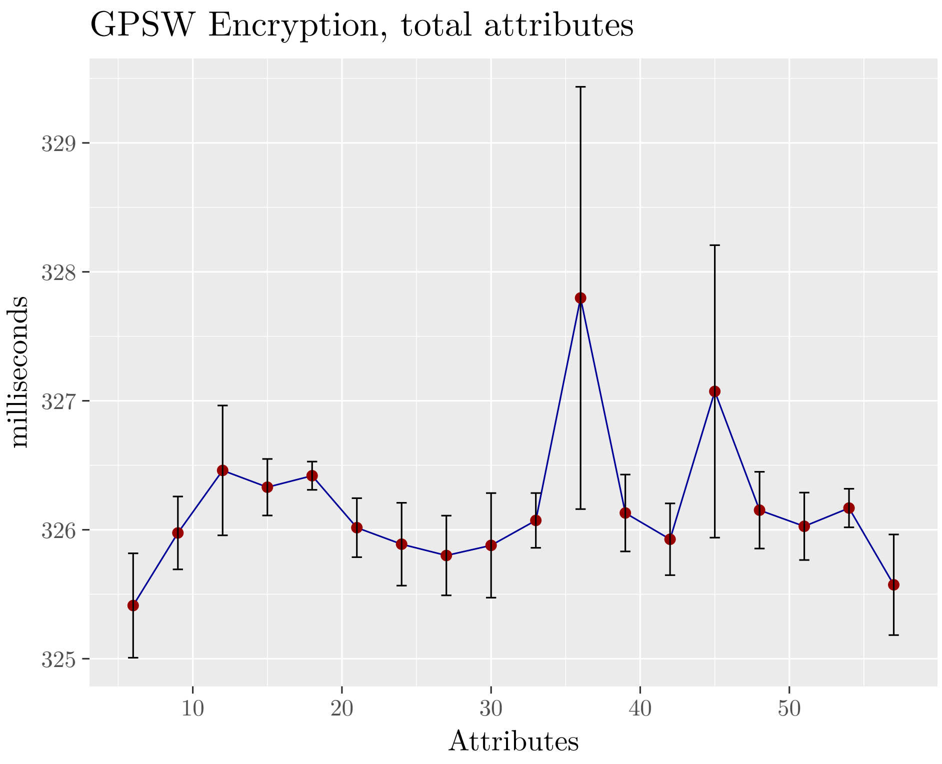


Figure 5.2: GPSW Encryption times, Variable: Total number of attributes in master key.

The results from the GPSW benchmarks are quite interesting. As we can see from Figure 5.2 and Figure 5.3 there is little correlation with the number of attributes and the time it takes to encrypt/decrypt data. This could be as a result of the low number of attributes in the message, or it could be a result of several other factors. Advise from supervisor explains no correlation as a valid expected result, so the avenues of error will be left unexplored.

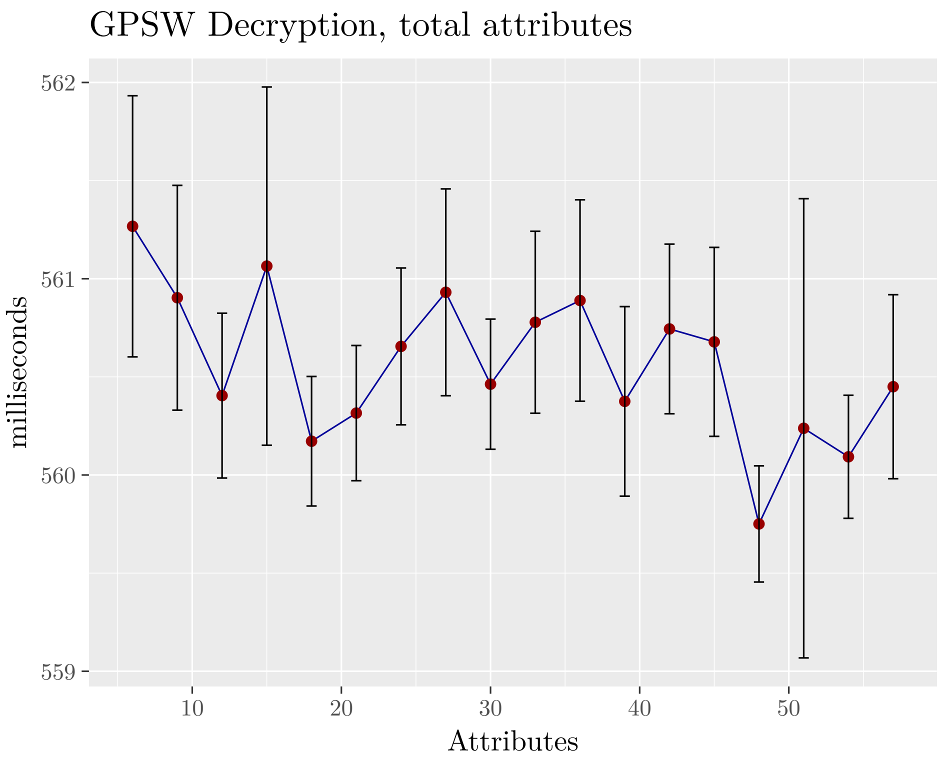


Figure 5.3: GPSW Decryption times, Variable: Total number of attributes in master key.

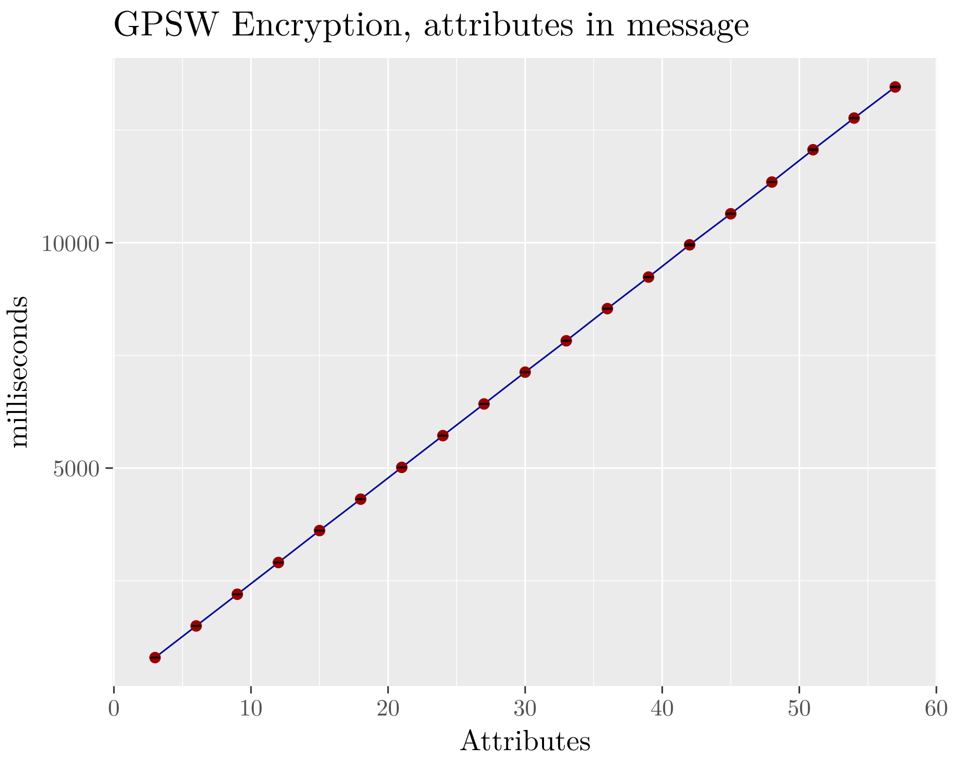


Figure 5.4: GPSW Encryption times, Variable: Total number of attributes on message.

The results in Figure 5.4 and Figure 5.5 shows that there is a linear growth with the number of attributes present in the message. This applies to both encryption and decryption. This seems to contradict Liang's results [Zha14], where the number of attributes in the message should not affect the decryption time. Also note that the variation of values in the graph is < 3ms for Figure 5.4 and < 2ms for Figure 5.5.

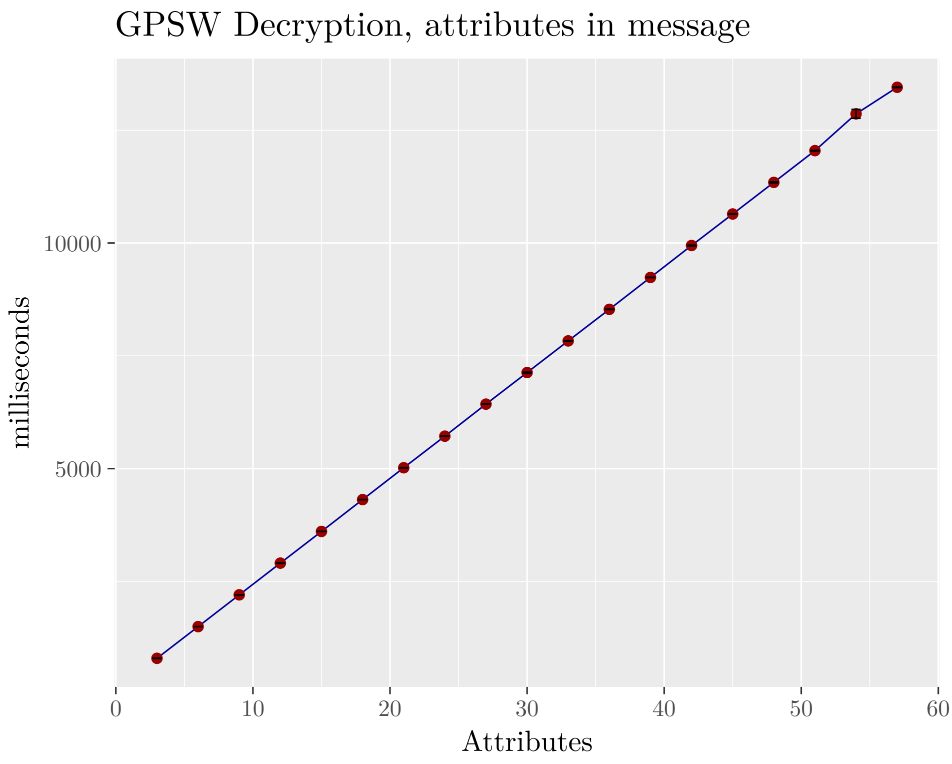


Figure 5.5: GPSW Decryption times, Variable: Total number of attributes on message.

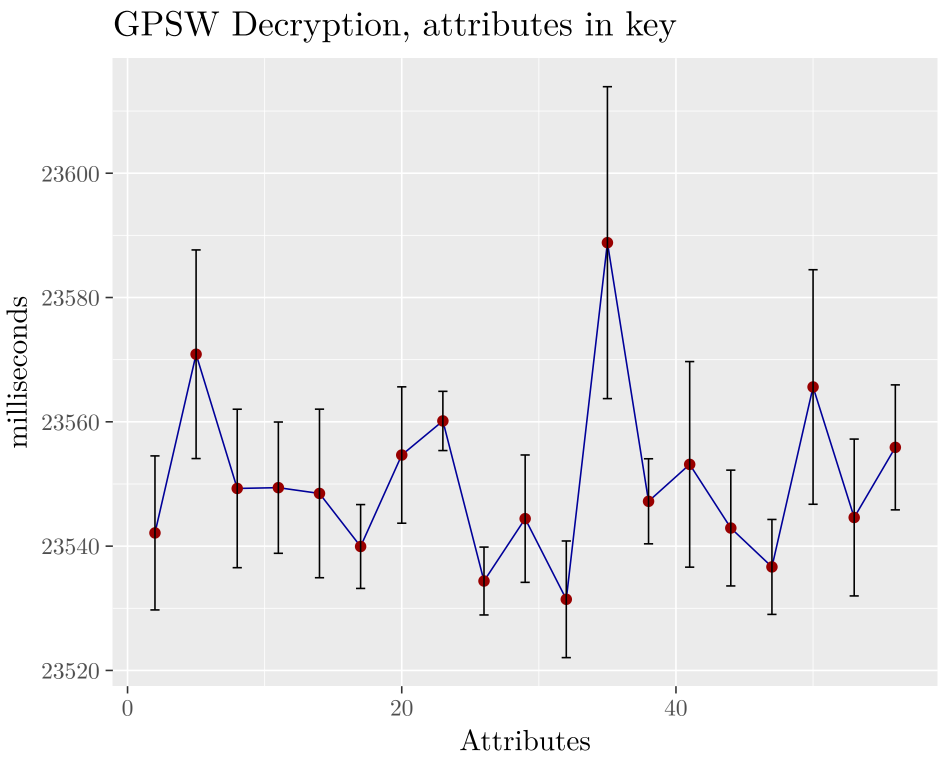


Figure 5.6: GPSW Decryption times, Variable: Total number of attributes/nodes decryption key.

The results in Figure 5.6 show little to no correlation between nodes in policy circuit and decryption times. Again, this is contrary to Liang's results [Zha14] where this was a linear relationship. In this case it could be because Liang increased the number of attributes used, and accidentally did not fix that variable. Another, more likely explanation is possible if the number of attributes in the message has any influence on the exponent of the discrete logarithm. Given that the number of attributes in the message was fixed at a high number, the raise-to-power operation could have become by far the most expensive part of the operation and thus drowning all other variables. However, the little to no variation in the results (50ms from min to max) in relation to the size of the data itself (23s) makes this theory unlikely, or at least that factor negligible.

All graphs are drawn with error bars. On some graphs they are only visible as a "-" on the circle denoting a data point.

5.3.2 RSA

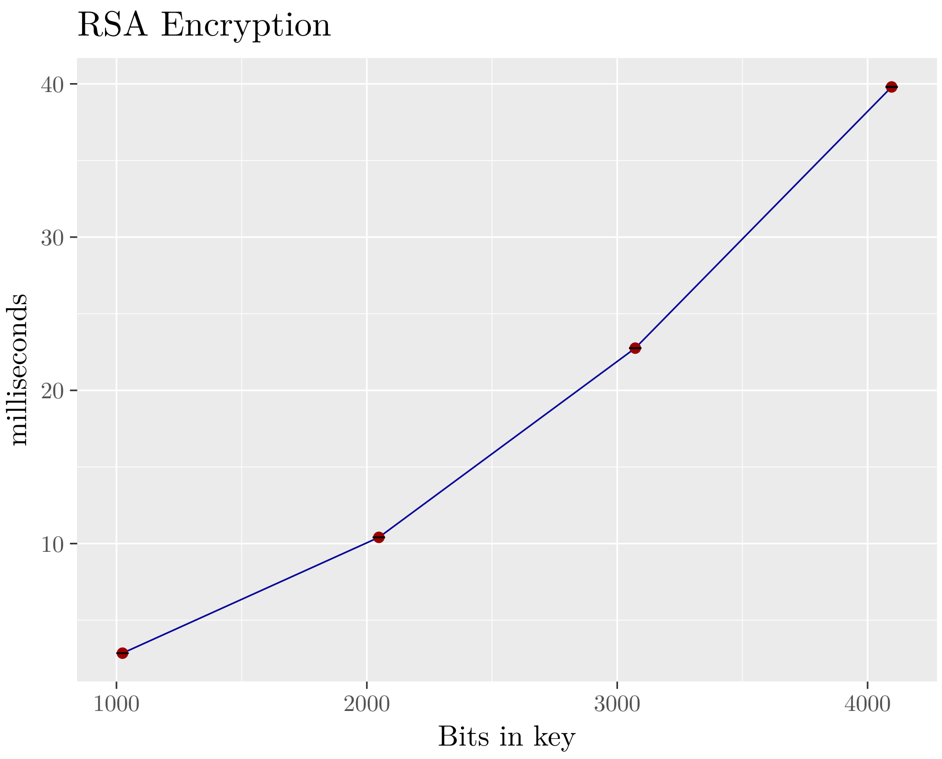


Figure 5.7: RSA Encryption times, Variable: Bits in key

For RSA we expect standard results, and that is what we get. In Figure 5.7 we see a near linear (not quite linear) relationship between key length and encryption times. In Figure 5.8 the results are the same for decryption.

It should be noted that in an encryption/decryption sense the method here described as “encryption” is the operation used to verify a signature. The operation in RSA is the same for encryption and decryption, the difference is in the choice of key. In “decryption” the private exponent is used. In “encryption” the public exponent is used. The confusing part is that for “signing” the private exponent is used and for “verification” the public exponent is used. This does sound like a RSA signing first decrypts the hash, then it is verified by encrypting it. This is not true, because the encryption and decryption is the same operation, but with different exponents. The RSA function with one exponent reverses the operation of the same operation using the other exponent.

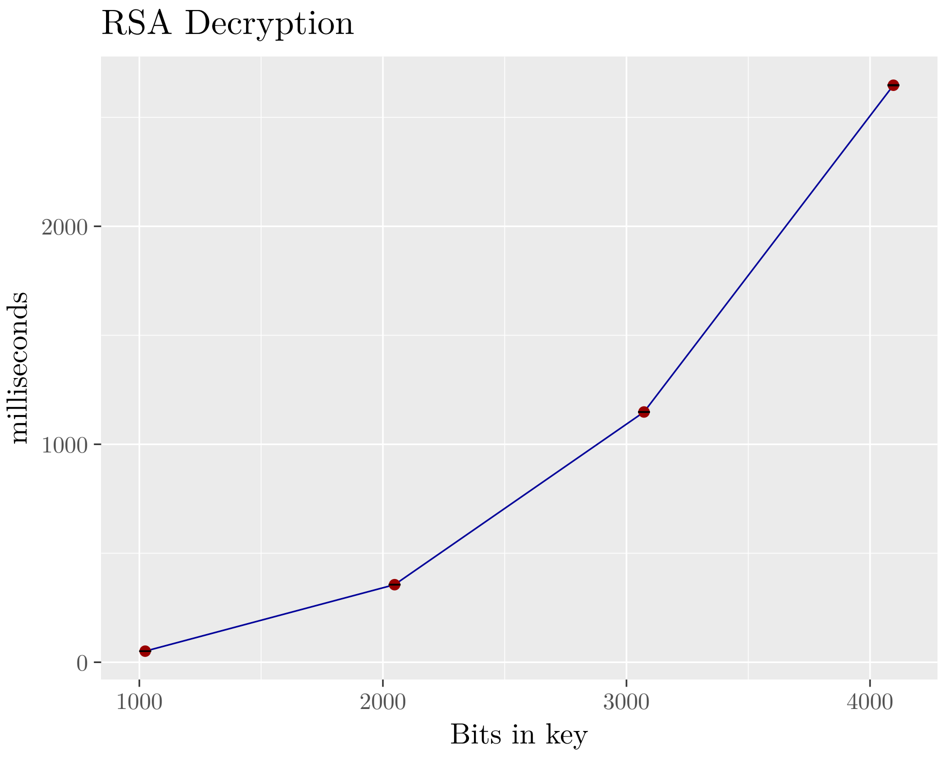


Figure 5.8: RSA Decryption times, Variable: Bits in key.

The thing to take note of is that “encryption” is two orders of magnitude faster than “decryption”. For signature-verification, this is perfect, one verifies a lot more often than one signs. For an application this means that it is quick to encrypt something (like a AES key), but it is a lot slower to decrypt them.

Note that all graphs are drawn with error bars. The error is so small it is only visible as a line in the point.

5.3.3 AES

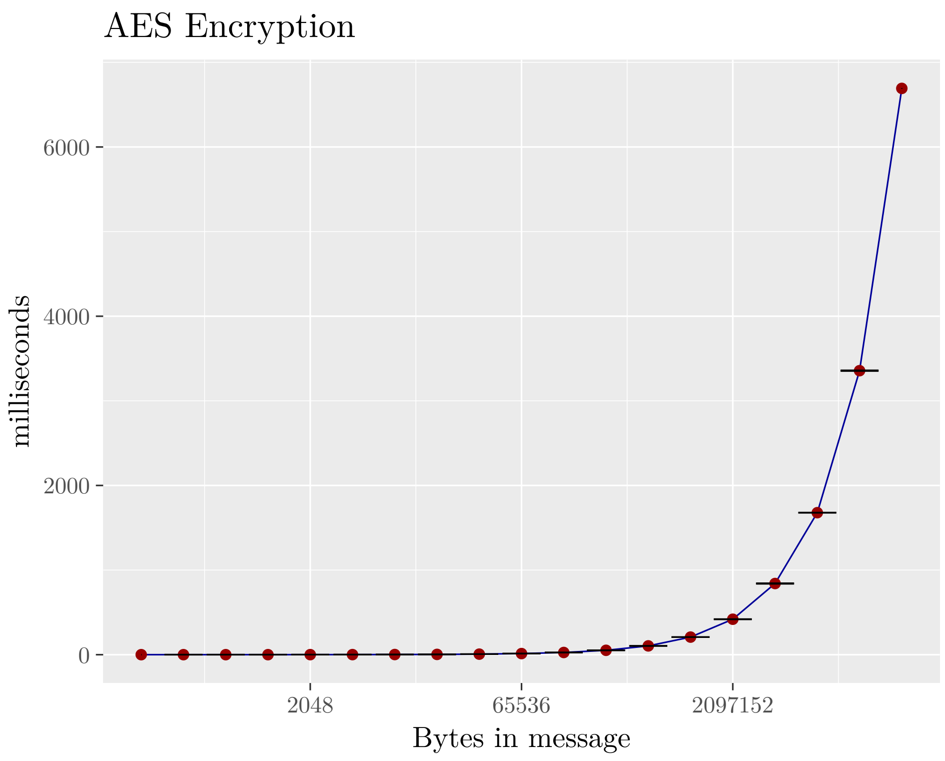


Figure 5.9: AES Encryption times, Variable: Bytes of data, Scales: Logarithmic along x-axis

The results from the AES tests are not that interesting, but it gives a baseline. Because the sizes were set by bit shifting a logarithmic X axis seems reasonable. In Figure 5.9 and Figure 5.11 these results give an impression of the times.

In Figure 5.10 and Figure 5.12 both axes are logarithmic, and as a result it is possible to see that the growth is as good as linear.

From this one can extrapolate the amount of data one could encrypt/decrypt with AES in the time it takes to decrypt the key using RSA or GPSW.

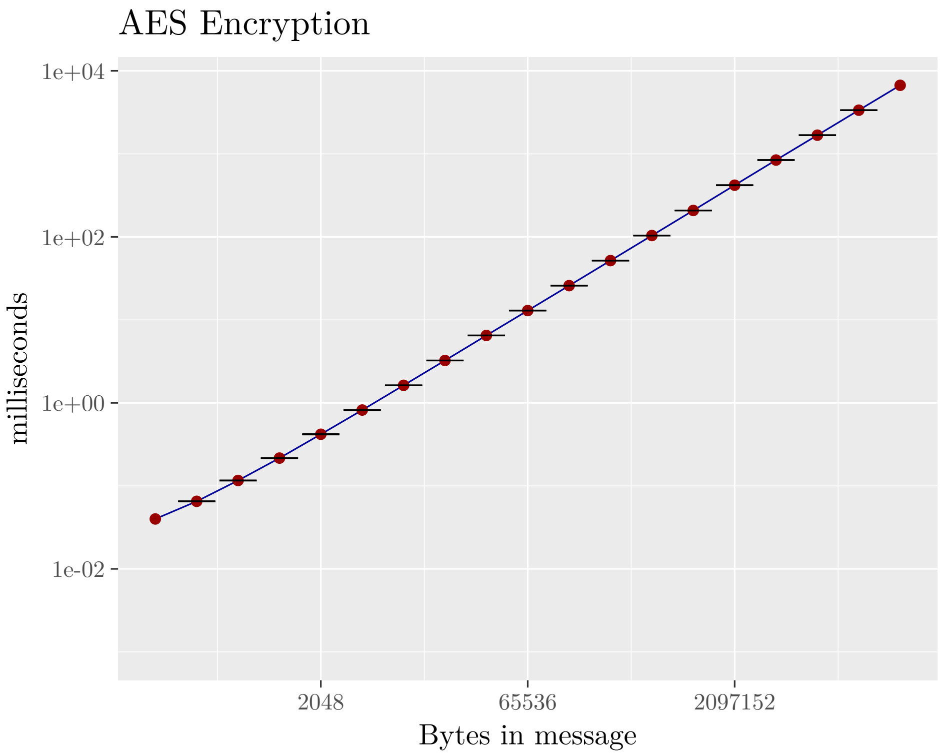


Figure 5.10: AES Encryption times, Variable: Bytes of data, Scales: Logarithmic along x-axis and y-axis

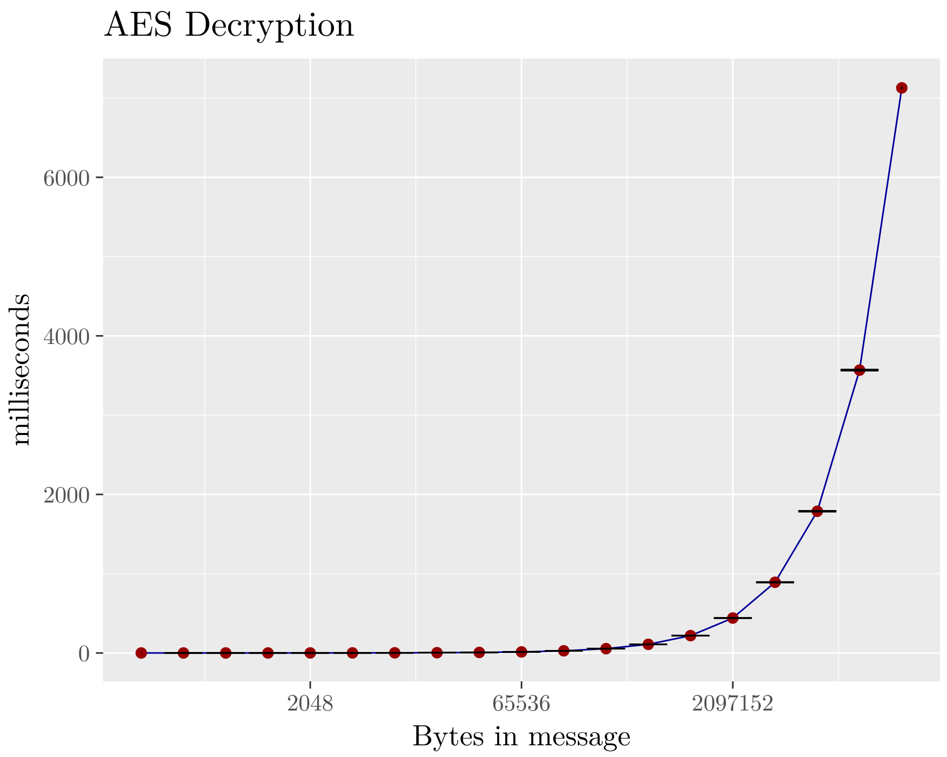


Figure 5.11: AES Decryption times, Variable: Bytes of data, Scales: Logarithmic along x-axis

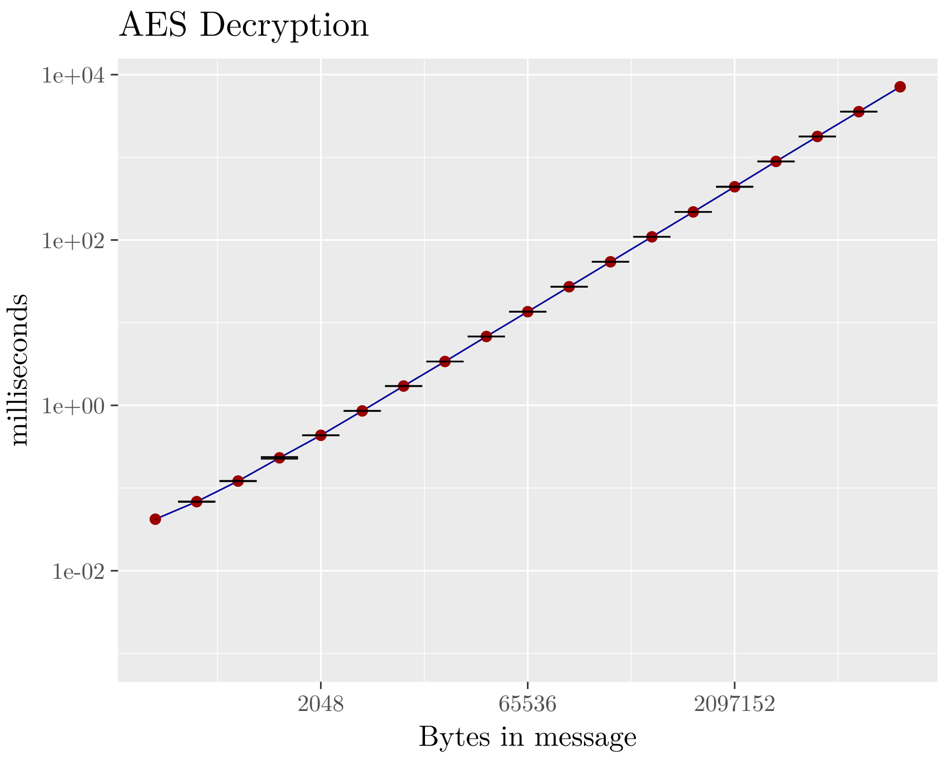


Figure 5.12: AES Decryption times, Variable: Bytes of data, Scales: Logarithmic along x-axis and y-axis

Table 5.1: An excerpt of the graphed data earlier in this chapter.

Result set	Var		Time
RSA decrypt	1024	bit	50.801
AES encrypt	16	MiB	335.631
RSA decrypt	2048	bit	355.821
GPSW encrypt au	3	attr	794.441
GPSW decrypt au	3	attr	795.702
RSA decrypt	3072	bit	1148.037
GPSW encrypt au	6	attr	1497.383
GPSW decrypt au	6	attr	1498.000

Table 5.2: Ratios between ciphers. For reference, 2048 bit RSA is used.

CipherA	CipherB	Ratio calculation	ratio	unit (per decryption)
RSA	AES	$\frac{(355\text{ms})(16M)}{355\text{ms}}$	16	MiB*
GPSW	AES	$\frac{(794\text{ms})(16M)}{(355\text{ms})(3\text{attr})}$	11.9	MiB/attributes
RSA	GPSW	$\frac{355\text{ms}(3\text{attr})}{794\text{ms}}$	1.34	attributes*

5.3.4 Ratios

To get the ratio data we need to get some raw data, so a relevant subset is presented in Table 5.1.

Because we can look at GPSW as linear growth with the number of attributes and AES as linear as a function of data it is possible to make comparisons. The Table 5.2 contains calculations for these ratios. The number represents what the variable of CipherB must be set to to be equivalent to one decryption of CipherA. In the case GPSW AES the number is a ratio between the variables. Because RSA is non-linear the sample point chosen is at 2048 bit, this is marked with an asterisk (*).

These ratios are approximately the same as seen in Appendix A.

Chapter 6 Building a System

This chapter will attempt to build an architecture based on the things described in the previous chapters. The goal is to have something that can heavily influence a design document for the implementing team.

Take note that several designs may have been fused together in the process of writing them in order to avoid repetition. For instance if two designs differ in only one aspect, both aspects are described as options but under the same design.

6.1 Prerequisites

No attention is put to communication between nodes. The assumption is going to be that the communication uses state-of-the-art Transport Layer Security (TLS) with bi-directional authentication or comparable mechanisms.

A solution must provide authorisation to data using cryptography as the access control mechanism. This means that if the user had access to all files the solution must prevent a user from reading the contents of the files or records they are not authorised to access. The solution does not need to mask the size of the files or records that can not be decrypted. For the purposes here a user with unauthorised access to a key is considered an authorised user.

There are no direct requirements for auditing, as the auditing can be implemented in the trusted code part of the application (section 7.1).

The granularity of access control should be adjustable, either by varying the size of data blocks or by some other means. This is not a strict requirement as it would show up as a selection characteristic for choosing a system.

There is no focus on trying to protect data currently being processed, so if the data is being protected it is assumed to be freely available in memory. Some solutions may contain homeomorphic schemes that would allow for processing of encrypted data, so even if it is not a goal to protect "hot" data it is still possible to do so.

All symmetric cryptography will be represented with AES. There are other algorithms available, but AES will be used as a generic representation of all of them. If another symmetric algorithm is being used it will be for a specific property of that scheme and the property will be described.

It is assumed that all the cryptosystems used have not been broken, for most systems it is seen as likely that a new cryptosystem would be able to replace a broken one. For special cases like ABE, these changes may involve a change of the access control scheme used in the cryptosystem itself.

Even if not specified Key Distribution System (KDS) systems should be interchangeable with a Hardware Security Module (HSM) in KDS operation. So any KDS system should not have highly specialised operations. This is to ensure ease of implementation and that implementing HSM support is relatively pain-free.

6.2 Solution anatomy

Every solution will be presened in the following structure:

1. A short description of the solution
2. Character stories.
    - Featured characters:
        - Alice: The data producer, or the module that does the encryption.
        - Bob: The consumer or data scientist, the user that needs access.
        - Eve: Eavesdropper, a passive attacker. Is an entity that can get access to all the ciphertext versions of all records. Eve is not infinitely powerful in terms of computing power, but can in different scenarios obtain different kinds of keys. Eve is considered infinitely powerful when it comes to data mining ciphertext for metadata and mining attached metadata.
        - Faythe: The trusted authorisation agent, can be any number of individuals, or the key management system.
        - Trudy: The intruder.
          Has the ability to "Hack" every machine in the system and gain complete access to the computer's memory. (assumption: HSMs not

hackable) For scenarios only a limited number of systems are "hacked" for a limited amount of time.

- Wendy: (Whistleblower) here the malicious insider
  has access to some data, and tries to gain access to more data. Roumor has that she may some times be affiliated with the entity Julian. She has access to KDS exploits if there are any.

– Scenarios:

- Starting conditions
  The process needed to cold-start the system, mainly from the first-run, but also for cases where a shut-down requires reloading keys.
- Data ingestion
  The process of loading the data into the data lake. (Alice)
- Data consumption
  The process for accessing data from the data lake. (Bob)
- Authorisation
  The process of granting the user (Bob) access to new data. Depending on the solution the authorisation agent (Faythe) is involved.
- What information Eve needs to get (decrypt) her stolen data, or get information from it.
- What systems Trudy can compromise in order to get (decrypt) data
- What data Wendy can obtain without authorisation (including collusion schemes)

3. Pros

4. Cons

5. Encryption times
   An overview of how long it takes to do encryption/decryption as a function of several variables that depend on the scheme. It will also contain the constants K_{AES}, K_{RSA}, K_{GPSW} that represents the time to encrypt/decrypt data with different schemes; note that while the ciphers have specific names, they represent their class of cipher, not necessarily the specific cipher. For K_{AES} the number represents the time it takes to encrypt/decrypt $1024B = 1KiB$ of data. For K_{RSA} and K_{GPSW} the number is the time it takes to encrypt/decrypt one AES key ($256b = 32B$).
   The length of each "record" is irrelevant as the heavy lifting is done by a symmetric scheme, and as shown in section 5.3 the symmetric schemes have performance that can be seen as negligible compared to asymmetric schemes. So the impact on performance with growth is mainly going to be the amount of data decrypted using asymetric cryptography.

6. Viability for “very small” data (e.g. e-mail)
   Data sizes of tens to hundreds of records handled at one time. Examples would be personal e-mails or medical records.

7. Viability for “small” data
   Data sizes of thousands to ten thousands of record at the time. For example a central medical database or HR records.

8. Viability for big data
   Data sizes of over a million records handled at a time.

6.3 Terminology

Clearly define terms being used in the rest of the chapter. It makes it easier to handle ABE vs. Key manager and merge the two ideas into one.

Secure Random
A random number with a guaranteed level of unpredictability.

Insecure Random
A random number but no guarantee for the predictability. This does not mean that it should be inherently predictable, but there is no need to use extra computing power to guarantee unpredictable randomness.

Policy bucket
One unit of access. All records that belong to the same policy bucket gets granted access to simultaneously. If access is granted to one unit all others are granted implicitly. In a ABE scheme every possible combination of attribute represents one policy bucket each.

Keyfile
A file containing a number of keys. It is encrypted using another key, the result is that anyone in possession of a key to decrypt this file essentially has access to all the keys in the file and whatever they may decrypt.

Key Mapping
The collective term for a mechanism for connecting a record to the decryption key and the keyfile(s) this key can be found in. For the purposes of this chapter this mapping is a black box. The default behaviour is that a key identifier (an **Insecure Random** number) is attached to the record. The mapping could be a part of the keyfile, or it could be a central store that links to a keyfile, this is not important for this chapter.

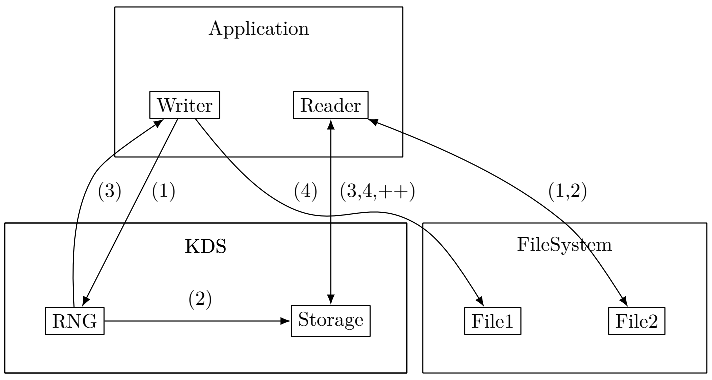


Figure 6.1: The flows of the "Everything using AES and random keys" for one record. The particular variation illustrated are the KDS housed keygen.
The writer appends a single record to the end of File1.
The reader reads File1 and retrieves the keys neccesary to decrypt the records.

6.4 Design: Everything using AES and random keys

The design for AES everywhere is one of the first ideas that are going to come up before they start looking at it. The idea is that every "record" is encrypted with AES with a random key. The key is than stored by a KDS

6.4.1 Operation

Starting conditions

To initialise the system there is a need for a key management/distribution system (Faythe). This system controls who has access to what keys. There are two key generation designs the only difference is if it is Faythe or Alice that does the key generation.

Data ingestion

When a new record arrives alice generates a key, encrypts the record and sends the key along with it's metadata to Faythe. Alternatively, Alice sends a request for a key along with the metadata to Faythe and Faythe does the key generation and sends the key to Alice. After the record is encrypted Alice Disposes o the keys.

Data consumption

To allow Bob to decrypt a record Bob needs to get the decryption keys for the records. These can be retrieved with a request to Faythe using the metadata. To mitigate the problem of the large number of key requests the requests for keys could use search syntax. After use Bob disposes of the keys.

Once a result has been produced Bob encrypts the data either by generating keys and submitting them to Faythe or by having Faythe generate new keys for him.

Authorisation

Authorisation happens by telling Faythe that Bob should be granted access to a new set of data. Then Faythe will start approving requests from Bob for those keys.

6.4.2 Attacks

Passive data attacks

For Eve to decrypt data she need to have access to each encryption key for each record. There are some data available in the metadata identifying each record.

Active compromises

Trudy could compromise Alice to get access to the records and keys passing trough while Alice is compromised. If keys are generated by Alice using an unsecure seed-vulnerable mechanism the keygen stream can be used to find future and/or past keys.

By compromising Bob Trudy would be able to access any data in a decrypted state in memory and any keys that may reside in memory. Because this is a system compromise of the compute platform, the mitigations of section 7.1 would not prevent this. These compromises assume that Trudy is able to gain memory access to the places where these kays are stored. The Java Virtual Machine (JVM) should try to prevent this.

If Trudy is able to compromise Faythe, every key is available for use. Faythe is the most valuable target in the entire system. For this system Faythe has full access to all keys and only denies/grants access based on access rules.

Privilege escalation

Wendy here refers to the user rather than the process. If the user-created part of the process has access to the decryption keys Wendy can embed them in the output data. Using these keys Wendy is able to retain access to data after her access has been removed. If the encryption system is transparent as described in section 7.1 Wendy would not have access to these keys.

If there is an escalation exploit in Faythe's systems Wendy could leverage this in order to get keys she are not authorised to access.

6.4.3 Encryption times

Variables:

n	Number of records to decrypt
r	Record length (KiB)
m	Number of keys that need to be loaded to find all the required keys for the particular set of records. For the record $m > n$.

The final time is n records with rK_{AES} decryption time each. In addition comes a $\log(m)$ search time for each key, per record.

$$n(rK_{AES} + log(m))$$

Now, it is possible to use a hashmap to negate the search times, but it will not provide a significant impact. It will still be included as a placeholder for a Application Programming Interface (API) call to a KDS for each key.

The real limiter of this solution is the memory usage. The system (or the KDS) needs to store m keys in memory. If separate API calls to the KDS is used this does not impact the available memory on the compute node, but adds the delay of the API call to the decryption times.

The encryption times are similar, except they have a key generation and storage step instead of a search step.

6.4.4 Pros

- Fairly fast
- Secure as long as the key manager is keept secure
- Can be made infinitely granular if needed.

6.4.5 Cons

- Generates a lot of keys, the key management itself might end up as “big data”
- A compromise of the key manager compromises everything.
- May exhaust the secure random key generator.

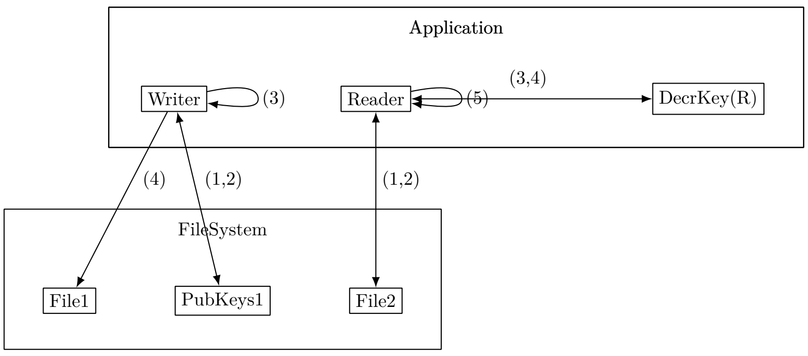


Figure 6.2: The flows for the design "Everything using ABE"
In this case the public (encryption) keys are stored openly on the file system.
The cases here are "cold start" cases, there the application does not have any keys.
The decryption key displayed is "personal" to reader.

– If a too high degree of granularity is used, the keys and metadata may exceed the actual data in size.

6.4.6 Viability

"Very small" data

Very well suited, in some respects this mechanism is what is used for most implementations that was looked at for existing data-at-rest encryption mechanisms. In their case a "record" is usually a file or a collection of files.

"Small" data

This will push the limits as the ciphertext would have to be matched up with the keys. If the metadata allows for a efficient search structure this is still viable.

"Big" data

Way too manny keys to keep track of, unless the data can be sifted using the metadata to a significantly smaller set.

6.5 Design: Everything using ABE

This solution is the same as section 6.4, but instead of using an access control system to give out the keys it has a header with ABE (section 4.3) containing the AES decryption key. This example uses a key-policy ABE, but a ciphertext-policy scheme can be used as well.

The saying “when you have a hammer, everything looks like a nail” [Mas66] is a common formulation of the law of the instrument. Some times, however, the problem seems to be that the person who just discovered the electric screwdriver forgot that there are still nails. This solution is a possible symptom of this problem. Luckily the ABE schemes seems to often be used to generate a AES key, and that does speed up the process.

In the preliminary tests for chapter 5 a benchmark was run on a MacBook Pro with a “Intel(R) Core(TM) i5-5287U CPU @ 2.90GHz” CPU while running other applications. The results for 3 attributes were around 40 milliseconds per encryption. For 18 attributes it was about 200ms. For some applications the acceptable time for processing a record is 10-20ms or faster.

As for the number of attributes to use, this will vary, depending on the level of granularity one want in the access model. It is even possible to allocate “future use” attributes on key creation to be used in the future.

6.5.1 Operation

Starting conditions

To get the system going Faythe needs to generate a master key MK and a set of public attribute keys PK. Faythe publishes the public attribute keys so that Alice has access to them and Bob privately receives a decryption key D with his access policy attached to it.

Data ingestion

Alice receives a record extracts the attributes, encrypts the record with a AES key and adds a ABE header with that key as a payload using the attributes γ and key PK. At this point Faythe is not involved in the encryption process at all and can be kept offline.

Data consumption

Bob retrieves the record, uses his D to decrypt the header and uses the AES decryption key to decrypt the rest of the record.

If implemented in a way that allows for it, it is possible to do filtering on the records based on the attributes γ before the data is even decrypted. This would reduce the processing power used for decryption of each record.

Authorisation

Faythe decrypts the MK, and uses this to construct a new D_n for Bob to use. The old decryption key D_o is disposed of. Although there are revocation-capable ABE schemes these will be ignored for the time being. After Bob receives D_n her involvement is no longer needed and her system can be taken offline.

6.5.2 Attacks

Passive data attacks

For Eve to obtain any cleartext data she needs a D capable of decrypting the subset of data she has the ciphertext to.

If some of the data is used for determining the attributes γ for a record, this will be available for Eve as a form of metadata.

Active compromises

Because there is no key management system the maximum impact Trudy is able of doing involves stealing or getting the D from Bob. She could also compromise a compute node and steal the cleartext from memory.

If Bob's system for decrypting involves him keeping the D protected (e.g. on a smart card) a compromise of a compute node would not even let Trudy get the D. The only thing available to her would be the decryption keys returned from Bob's smart card.

Privilege escalation

Wendy is only capable of obtaining data as defined by the particular ABE scheme's collusion capabilities. For the GPSW scheme this is $set(D_1) \cup set(D_2) \cup ... \cup set(D_n)$. (details in section 4.3)

6.5.3 Encryption times

Variables:

n	Number of records to decrypt
r	Record length (KiB)

This means n records, each take K_{ABE} to en/decrypt the header, and rK_{AES} to en/decrypt the payload.

$$n(K_{ABE} + rK_{AES})$$

In this case, the 50% speed point, the ratio where the throughput is halved, is when:

$$K_{ABE} = rK_{AES}$$

Given the ratio observed in chapter 5 this means that r would have to be several megabytes up to gigabytes, depending on the number of attributes.

6.5.4 Pros

– No central key manager

6.5.5 Cons

– Extremely slow

– No auditing of key usage by the manager

– ABE is still a young technology, no good tools exist yet

– Re-keying is going to be extremely difficult.

6.5.6 Viability

"Very small" data

For a user-driven application using only a few records at a time (like a medical record application [Zha14]) waiting a few seconds for the record to decrypt or encrypt is not a big problem, it would be something one got used to.

Waiting $100 * 40ms = 4s$ or $100 * 200ms = 20s$ for a 100 record retrieval for a user driven application as long as it does not have to re-encrypt for every little operation done with the data.

"Small" data

In a database application touching thousands or records at a time a $40ms$ delay on each record is unacceptable.

"Big" data

For big-data operations a $40ms$ delay per record is not something that can be easily paralleled away.

6.6 Design: Key reuse

The only real solution to the problem of having too manny keys is going to be variations of this solution. To avoid being repetitive all the variants are merged into one. Every reference to a KDS can be replaced by a ABE header on the keyfile or keyfile section. The ABE access delegation is the same as in section 6.5

Any attacks on this system is going to be equivalent to those in section 6.4 and section 6.5, so they are not going to get repeated here.

6.6.1 Operation

Starting conditions

Depending on the scheme there could be several initial conditions to the system. For a ABE based scheme, a master key must be generated and the public part distributed to Alice.

For a RSA based scheme, a key for each **policy bucket** must be generated and the public keys distributed to Alice.

For a AES based scheme, a key for each **policy bucket** must be generated and distributed to or made available for Alice.

Data ingestion

The idea here is for Alice to re-use the AES keys for as long as possible. So when Alice recieves a record and determines what **policy bucket** it belongs to she encrypts it with the key she has stored for that **policy bucket**. After encrypting the record she appends the **key mapping** ID to the file.

If she does not have a key stored for that **policy bucket** she generates a **secure random** key for that bucket, generates a **insecure random** ID and submits it to the appropriate **keyfile**. If the scheme does not involve ABE Alice could send the key to Faythe who would then encrypt the key without Alice ever touching the master key. Another option is for the **keyfile** to use a RSA based scheme.

A key is dropped from the "store" once it reaches a certain age. This age could either be a number of records it has been used for, the time since generation, or any combination. this could be implemented on retrieval, but that would leave expired keys that has not been accessed in a long time to a hacker.

If Alice has problems keeping track of all her stored keys (is running out of memory) she would have to prematurely drop a key, a good selection metric for this could be *longest time since used.*

Data consumption

when Bob starts up a data processing job, as a preparation he will read all the keys he might need. For systems using ABE this involves decrypting the ABE headers because doing that too manny times is costly. This opportunity could be used for encoding the files again and minimising the number of unique ABE headers. If Bob had malicious intentions he could do the re-encryption of the **keyfile** with fewer or more attributes and effectively grant access to more people.

Authorisation

Granting access again depends on the structure of the scheme.

For ABE a new key is generated for the user. This key either has a new policy or a different combination of attributes. This effectively grants access to new data.

For RSA the key to the new **policy bucket** is given to the user, either by directly granting it, or by adding it to a **kayfile** encrypted with the user's personal key.

For AES the key to the new **policy bucket** is given or made available to the user.

6.6.2 Attacks

Passive data attacks

This scheme has problems with vulnerability in the same way as the one described in section 6.4 and section 6.5. It is also vulnerable to data leakage trough the key to data mapping system (section 7.4).

Active compromises

If Trudy is able to compromise a ingester node she is able to compromise only the keys that node has in memory at that time. If this includes master encryption keys this is a complete compromise, if it only includes data encryption keys the compromise is limited to the lifetime of the encryption key.

If the compromise is of a compute node the attacker can compromise anything that is encrypted with a key store in memory during a processing job while the node is compromised.

Privilege escalation

This scheme has problems with vulnerability in the same way as the one described in section 6.4 and section 6.5.

6.6.3 Encryption times

Variables:

n	Number of records to decrypt
r	Record length (KiB)
m	Number of keys that need to be loaded to find all the required keys for the particular set of records. For the record $m > n$.
b	The number of records in each policy bucket in the dataset to be en/decrypted. This makes $\frac{n}{b}$ the number of buckets.
p	The number of partition in a policy bucket. (ABE blocks), this will be reset to 1 after a unification process.
f	The number of encryption keys in each bucket. May be referred to as bucket fragmentation.

Constants:

K_k	the time it takes to communicate one record to KDS.

So, in the ABE case $\frac{n}{b}p$ bucket headers need to be decrypted first. Then n records need to be decrypted.

$$\frac{np}{b}K_{ABE} + n(rK_{AES})$$

In this case we assume that the number of records that uses each key $(\frac{b}{nf} = \frac{1}{m})$ is so high that the search times and memory impact can safely be ignored. Note that if $f > p$ the ABE part is not a function of m.

For a symmetric KDS system there would be a a retrieval for each key required mK_k. Then the decryption of each record.

$$mK_k + n(rK_{AES})$$

Again, we ignore the search times for convenience. The same applies to encryption.

For the ABE encryption process the speed is still the same.

$$\frac{nf}{b}K_{ABE} + n(rK_{AES})$$

Note that here f is used rather than p, because in the encryption process $f = p$ by design.

The process of unifying a keyfile takes:

$$\left(\frac{np}{b} + \frac{n}{b}\right)K_{ABE} = \left(\frac{np+n}{b}\right)K_{ABE}$$

and it requires that the entity doing this is able/allowed to decrypt all the keyfiles in question. This will reset the variables to $p = 1$.

The process of unifying the keyfile and reduce the key space (f) is:

$$\left(\frac{np+n}{b}\right) K_{ABE} + 2n(rK_{AES})$$

This assumes that there is sufficient memory space to hold all the decryption keys while processing the data. This will reset variables so that $p = f = 1$.

The efficiency of this design in ABE mode is simply a function of p and the number of buckets ($\frac{n}{b}$). As long as it is possible to hold m keys in memory and efficiently search them.

The efficiency of the symmetric method is directly dependent on K_k (and bulk loading ti minimise delay) and that p and f are sufficiently small so that m keys can be handled and searched in negligible time.

6.6.4 Pros

– Faster

– Fewer keys

– Granular access

6.6.5 Cons

– ABE is still slow

– Speed decreases with granularity of access

– Could still have a lot of keys to keep track of

6.6.6 Viability

"Very small" data

The scheme in all modes were viable for this before optimisation were applied. It is still equally viable.

"Small" data

As long as the number of **policy buckets** are kept low both RSA and ABE is completely within reason. It will be somewhat slowed down, but for any usage outside the high-performance use cases this can be reasonable.

The AES variation the bottleneck is the key to record mapping. As long as that is implemented in a reasonable manner, this should not pose a problem.

“Big” data

As long as the number of **policy buckets** are kept low the distributed parallel nature of big data systems will overcome the disadvantage in speed. However, there is going to be problems with shipping keys around if they do not reside on the same nodes as the data they are connected to.

6.6.7 Conclusion

In combination with the results from subsection 5.3.4, it is possible to draw a conclusion. This scheme is so dependent on the access control scheme for performance, so it would require too much access control performance tuning to be cost effective. There could be done research in policy optimisation and management in order to optimise the key systems.

While the solution may be viable now it would require implementation. This implementation could be done as a larger community effort, but for a reasonably small company like Telenor Digital it is not a particularly viable undertaking.

6.7 Design: Bolt CryptDB on Top

If the particular technology satisfies some parameters it is possible to use CryptDB (section 4.2) as an encryption mechanism. CryptDB does not offer access control, but it could be extended to do so. The exact workings of how to do extend CrypdDB to allow for access control would have to be the subject for an entire research paper, but it looks possible at first look.

On the requirements [GAT+15] that need to be met in order to support CryptDB the requirements are quite strict. A major subset of the Structured Query Language (SQL) query language needs to be supported, the details require inspections of the source code. The data system must support stored procedures for the custom operators and the homomorphic operations. Lastly there must be support for bit-level operations for the stored procedures.

The data processing technology that best fits these requirements are Amazon RedShift. Because Amazon RedShift uses a PostgreSQL compatible interface it could conceivably be used without modification. It turns out that it does not support stored procedures [GAT+15], they are essential to CryptDB’s implementation [PRZB11]. It is still possible to use the original CryptDB design and adapt it to big data, but that would once again be an undertaking at the scale of the original paper.

6.8 Design: Just use Apache Ranger

The following system does not follow the established pattern, because its nature is fundamentally different. It is included to make a comparison to the non-crypto solutions available.

All solutions above require a certain amount of planning, or simply to encrypt everything. Then comes the question, with the same amount of planning is it possible to use an existing access control solution like Apache Ranger and achieve the same level of security?

6.8.1 Table-like systems

For table-like systems with a Ranger plugin (like Apache Hive) Ranger provides authentication using the Ranger Policy Manager and offers "row-level authentication for one or more table views"[horc]. How precisely this is implemented requires some code research that has not been undertaken. The assumption is going to be that it is functionally equivalent to a forced filter condition (described in section 3.1).

6.8.2 HDFS

The second way of enforcing data access policies with Ranger is on the file level. Ranger for Hadoop File System (HDFS) lives in the HDFS NameNode [Gan]. The Ranger plugin denies requests for files that the user does not have access to. Because of this design only resides in the NameNode it is possible for an attacker that knows the IDs of Data chunks to retrieve them directly from the Data nodes. This kind of attack will be regarded as unlikely, but not impossible if the target is sufficiently valuable, typically nation-state attacker.

This is the point where data-at-rest encryption comes in [horb, cloc]. There are several implementations of this, but for this the focus is going to be on the stack used by Ranger Key Management Service (KMS) [hord]. This uses HDFS' "transparent encryption" system where files are encrypted and decrypted without the application using them knowing. In the default system the keys are stored on the NameNode in a Java KeyStore file. The disadvantage of this is that it is unencrypted and any attacker capable of obtaining this file has full access. Ranger KMS encrypts the keystore and uses a different storage mechanism, supposedly "more robust" and also allows Ranger policies to be added to the keystore. The keystore should then have the same policies as HDFS and block attacks involving guessing block IDs.

How to add more granular access control

On a file system level access control mechanism it is impossible to restrict access on a more granular level than per file. The only way of allowing for partial access is to split the data into more files.

For a “field” level separation it requires a direct one-to-one mapping. This mapping could be a **insecure random** number signifying an ID that can be used to join the parts of the record together. In this case, because there is a one-to-one direct mapping the mapping is incapable of leaking sensitive information. If someone were to normalise the data to save space this security will be lost. The file containing the sensitive information can then be access restricted.

For those who are not familiar with what normalisation means, is is a process in database theory to make data more manageable. For example a record from a online store may contain (PurchaseID, Customer, Customer name, Customer Address, Customer phone number, total amount paid, payment method, date, etc.). Normalising such a database would involve storing the customer in a separate table and referencing the same record each time. Thus making the records: (PurchaseID, CustomerID, total amount paid, payment method, date, etc.), (Customer, Customer name, Customer Address, Customer phone number) because one record of the latter can be used for multiple records of the former, thus reducing the total size of the database. The privacy issue with normalising is that even if access to the customer details table is restricted, the purchase patterns available in the former table is still available with unique customer IDs. This could be used to make educated guesses on who the users may be.

Separating access on a per record level is a lot easier. One simply makes a file or directory per **policy bucket** and sorts the records into the correct file/directory. Apache Spark at least is happy to handle files or directories as one resource, but there may be some issues along the way so buckets would have to be added separately to the data pool.

Is this going to impact performance?

Could separating data into separate files impact the speed of a data query? The answer is yes, and it will. By the very nature of a data lake, the data lake does not care about the details of the contents, so there is no guarantee that the data that belong together resides on the same node. If the records do not reside on the same node it will have to be transferred over the network. This introduces delay and increases processing time greatly.

6.8.3 Pros

- Already implemented, just requires some planning and data modelling.
- Fast
- Transparent to the user
- If used with a HSM it can be made certifiably secure.
- Maturing technology, unlike a from-scratch implementation.

6.8.4 Cons

- Relies on the policy engine being fast enough
- On HDFS it can not offer granularity finer than per-file.

6.8.5 Viability

"Very small" and "Small" data

Full-disk encryption schemes like Microsoft BitLocker [Kor09] (Used by Windows OS) or Apple FileVault [CGM13] can use this for encrypting user home directories with other keys than the main system. So in a sense, this kind of access control is in use on a very coarse access level.

"Big" data

This should be obvious. As it is a solution specially developed for the big data field, it must to some degree be suitable, at least in the eyes of the designer. The advantage this system has is that it intervenes at a very early stage and does simple access control checks. The only problem that may arise is that the single policy node may become a bottleneck. On that subject the answer is that HDFS has been doing just fine with just a single active active NameNode.

6.9 Finding a balance

The problem with all of these solution is to find a balance between praticality and

The solution using AES scores high on speed, but the number of keys generated is so high it is going to make a big data problem out of the key management.

The solution using ABE scores low on speed because it takes a long time to decrypt each record. For a quick test on the Core i5 used in the preliminary tests (mentioned in section 6.5) the command "openssl speed -evp aes-128-cbc" gave "650820.84kB/s (16B blocks)". For a $40ms$ en/decryption to take up half the crypto-time each record would have to be "$650'820 \cdot 40 = 26'032'800$" bytes long.

There is no reason to grant row-level access to all data, so the easiest solution to this problem is to reduce the number of keys in use. For instance with an ABE scheme the same AES key can be used for a number of records with the same attribute combination. The same goes for an AES solution. The problem is then going to be the number of access control combinations possible, because the unit doing the encryption needs to keep track of a key for every combination it has encrypted recently.

As an example for ABE: the master key has 50 attributes, the number of possible combinations is going to be 2^{50}. This is not feasible. If we know a bit about the attribute-space it is possible to reduce this number. For instance if we knew that the number of attributes present was about three on average the number of combinations can be estimated using a binomial coefficient to be approximately:

$$\binom{50}{3} = 19600$$

Within an order of magnitude, worst case scenario. If these 50 attributes actually belong to more distinct attribute spaces it turns out that:

$$\binom{25}{3} + \binom{25}{3} = 4600$$

So it is beneficial to split up these spaces. In the case of ABE the master key size may also impact encryption/decryption performance in some schemes, even though the GPSW tests in section 5.3 did not indicate this.

In the following sections several solutions for optimising the solutions will be presented and analysed for strengths and weaknesses. For good measure the same analysis will be applied to the existing Apache Ranger; and for a lot of use cases it is going to turn out to be the best solution.

6.10 Conclusion

Although the prospect of using cryptography as an access control mechanism may be appealing in principle it would require a whole new generation of data processing tools to be developed in order to take advantage of it. It offers no tangible advantages over the tools available today apart from a higher level of security. For these reasons it is unlikely that cryptographic access control mechanisms like the ones described here will ever be commonly deployed. If they do get implemented it is going to be because of a spread of mistrust in Platform As A Service (PAAS) providers, or stricter regulations.

All access control schemes require a certain level of planning, this defies the principle/idea of a data lake, but it can't be helped. As long as the data is legally considered sensitive, there is a legal requirement for planning.

Modern cryptographic schemes like ABE offer new features that can be used in new ways, but at the cost of processing power. In a course on cryptography one might be told that RSA is a slow cipher, but when compared to the GPSW algorithm in chapter 5 it becomes extremely fast in comparison.

Table 6.1: Summary of the schemes described in this chapter

Scheme	**Pros**	**Cons**
All AES	Fast encryption/decryption.	A lot of keys, thus key management/mapping problems. Encryptor must have the decryption key.
All ABE	One decryption key per "user". Encryptor only has encryption key.	Really really slow.
Key Reuse (AES)	Fast encryption/decryption. Reasonable key management problem.	Encryptor must have the decryption key. Still a lot of keys to keep track of.
Key Reuse (RSA)	Encryptor only has encryption key for extensive time periods. Can be scaled to a good performance compromise.	Still a lot of keys to keep track of.
Key Reuse (ABE)	One decryption key per "user". Encryptor only has encryption key for extensive time periods.	Slow.
Apache Ranger	Fast. Early intervantion. Proven scalable in production. Software exist today.	Not a cryptographic access control system.

Chapter 7 Considerations for Encrypting Data

Big data is big, so even encrypted data may give up some of its information. This chapter is going to cover some ways of preventing data leaks trough meta-information. It will also cover some subjects that a designer need to keep in mind, but did not fit into any other chapters.

7.1 Trusted workers and key managers

If the system is implemented as a "transparent" plugin to the system it is possible to keep the actual decryption keys away from user accessible space. The way of doing this is by making the decryption a semi-transparent layer of code that is pre-deployed to all the compute nodes. This module will be referred to as "trusted code".

The trusted code implements isolation from the user code and the user-level code only interfaces using the users keys. Depending on the user scheme this could be a decryption key, or it could be a public key pair. In different schemes this interface might be a challenge-response mechanism or some other interactive interface. The latter is practical if the solution uses smart-card authentication.

The trusted code handles all data decryption and encryption and therefore keeps the keys isolated from user "untrusted" code. It also handles any audit logging if necessary.

This scheme only works if the worker nodes are trusted. If the worker nodes are not trusted then the scheme is rendered useless.

7.2 Multi-party authorisation agent

A common system for handling the authorisation agent is to require multiple parties for a authorisation. This ensures that at least two or more people in the trusted position would have to collude in order to compromise the system. For a key manager system the way to do it is simply to require confirmation before a change takes effect. For an ABE based system it is going to be a bit more complicated because of the decentralised nature of the system.

For this example the GPSW cipher is used. Even though GPSW has a lot of properties that are not shared with other ABE systems, it should be possible to use some of the ideas here to implement it for them as well.

The GPSW cipher is a key-policy ABE so for changing an access policy for a user one needs to issue a new user key. The system for policy-change on a ciphertext-policy system is a bit more complicated, but also doable, it depends on how the policies are built.

The central part to the GPSW cipher is the master key. The master key contains all the information to build any user key. The way of protecting it is to keep it encrypted. A preferable solution is to keep it offline and only use it when one need to craft a new key. The way to make sure that multiple people are required to access the key is by ensuring that secrets kept by multiple people are required. One way is that one person has the physical key or code to where the key is stored, and another person has access to the decryption key. This is not a very good way of doing it and it is very restrictive.

The way that's a bit more flexible is to split up the key into multiple parts. This is where secret sharing schemes (section 4.1) comes in. Note that the special case of "all keyholders need to be present" is easier to solve using an XOR key sharing scheme. The reason XOR is better in that case is purely due to the reduced complexity of the code involved. For all other cases, a scheme like Shamir [Sha79] or Blakely [Bla79] that guarantee that any partial set does not bring you closer to finding the full secret.

Why is it so important that a partial key does not bring you closer to the full key? It would reduce the key-sapce a brute-force attack can do, and if the cipher is vulnerable to a partially-known key attack the search sapce is shrunk even further.

An interesting idea is to use homeomorphic encryption to increase the security level. This would require a homeomorphic cipher that has the homeomorphic operations used in the key building of the ABE scheme in question. If such a scheme exists it is possible to have one group who generates the keys, and another group that has the keys to decrypt these keys. The only advantage gained form this is that it adds a level of more people that needs to collude in order to exploit the system. It is highly unlikely that such a scheme would have any uses outside of military applications and security levels.

7.3 ABE Policy building

How does one build the best ABE policy? The answer to the question depends on what one wants to achieve. For instance a tiered access policy is simply an attribute per tier. For more complex schemes a different approach is necessary or desired.

7.3.1 Enumerating values

If the scheme in question gets slower with the length of the master key, it is possible to enumerate different values into an encoding. This enumeration is then used instead. Some ABE schemes like GPSW has no ability to trigger on a NOT. To avoid this it is possible to add a inverted bit attribute. So for instance a country encoding $C[]$ would look like $(C_0, C_1, C_2), (\overline{C_0}, \overline{C_1}, \overline{C_2})$. This particular example is capable of encoding 8 unique countries in 6 attributes, but a check for a single country requires 3 2-way ANDs in the policy. It only requires a single 3-way AND if it is possible to make n-way ANDs.

The advantage is not in the lower number of bits, but in the higher numbers. A 8-bit encoding is capable of representing 256 values in only 16 attributes.

As shown in chapter 5, the GPSW scheme does not slow down with a larger master key, so this optimisation would not be necessary for GPSW, unless there are some hidden factors the benchmark didn't pick up on.

7.3.2 Optimising circuits

A circuit is a system of binary logic used to build a policy. In the GPSW scheme this is a tree of "nOFm" operations. The easiest policy building mechanism is to model the policy as n-way OR-gates ("1OFn") and n-way AND-gates ("nOFn"). For an advanced policy that could have three or four enumerated values of a variable in it (ORed), this would result in a lot of gates and a very complex policy circuit.

It is possible to reduce the complexity of the circuit using for instance Karnaugh maps [Kar53] or the Quine–McCluskey algorithm [Qui55]. Any optimiser implementations need to take a few thing into account when trying to optimise a circuit. The first and foremost is the lack of NOT expressions (at least for GPSW). Secondly some values are mutually exclusive. For instance a "Originated in Poland" and a "Originated in Norway" are mutually exclusive so $N \wedge P = FALSE$. This also applies to enumerated mappers where the expression $(C_0 \wedge \overline{C_1} \wedge C_2) \vee (C_0 \wedge C_1 \wedge C_2)$ (Figure 7.1) Is equivalent to $C_0 \wedge C_2$. (Figure 7.2, left) Note that these algorithms were capable of doing this by themselves, but in this case $\overline{C_1}$ and C_1 are different "inputs" so they do not know that they are mutually exclusive.

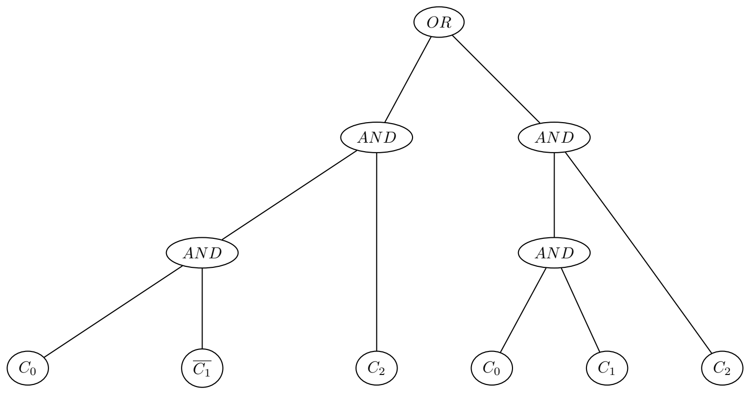


Figure 7.1: Example of an enumerated policy with 2 values, no optimisation, and only 2-way ANDs

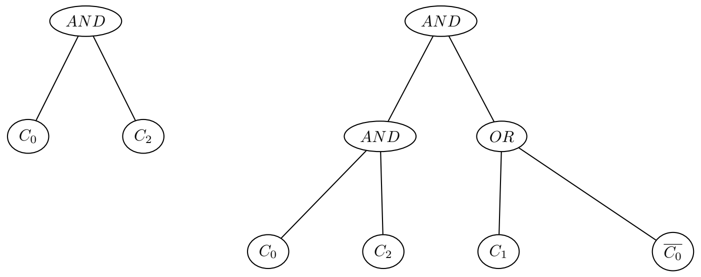


Figure 7.2: Example of an enumerated policy with 2 values (Figure 7.1), optimised. The left hand tree is mutual exclusiveness aware optimised, the right hand tree is not.

An example of a circuit minimisation that allows for a smaller circuit is the previously mentioned expression

$$(C_0 \wedge \overline{C_1} \wedge C_2) \vee (C_0 \wedge C_1 \wedge C_2) = C_0 \wedge C_2$$

This makes a circuit with 6 2-way AND and 1 2-way OR (or 2 3-way AND and a 2-way OR) into a single 2-way AND. An optimiser that does not know about the mutual exclusiveness of the attributes may only be able to reduce the expression to something like $(C_0 \wedge C_2) \wedge (C_1 \vee \overline{C_1})$ (Figure 7.2, right) an expression with 2 2-way AND and 1 2-way OR.

This also talks to strategic enumeration. If enumerations that often appear in policies together are places one fit apart it allows for a circuit granting access to both to be smaller than a circuit granting access to only one.

7.3.3 Application for GPSW

For GPSW this strategy is different. The goal would have to be to minimise the number of attributes present in each “message”, or reduce the number of messages in a “batch”. In that case enumeration is not a good idea. From the test results in chapter 5 it seems that the impact of total number of attributes are only on the length of the public key and the size it takes in memory. This assumes that the up-front cost of generating he master key is considered negligible in the long run.

7.4 The key-mapping data leak problem

The problem with using a mapping between key and ciphertext is that information is leaked trough the key mapping itself. It is not too hard to minimise as long as one knows how it works. For an ABE scheme the rule of thumb would be that all the attributes present for a data record is publicly visible. For other schemes it means that all properties that are used by the access control scheme is visible.

An example of a property could be $x > 1000$ or “Country of origin is Poland”. The problem for privacy arises only when there is enough information to identify a sufficiently small group of people/devices. For instance if two properties were “Country of origin is Poland” and “Membership level is GOLD”, and there is only one or two gold members in Poland; this is going to be a problem for privacy.

Another thing to remember is that all of these attributes/properties can be combined with all the unencrypted ones as well. This means that in some situations a data field may need encryption even though it may not be considered sensitive by itself.

In conclusion: Be aware of what information is left unencrypted, sensitive data may leak by combining these properties to isolate a user/account/device.

7.5 Deterministic encryption

Deterministic encryption was briefly discussed in subsection 4.7.2, but when it comes to big data the problems need even more attention. The term deterministic encryption is here defined as that the cleartext m_0 will always encrypt to the ciphertext C_0 with the same key. This is a problem because it means that even if an attacker does not know the cleartext m_0 the attacker can just use the ciphertext C_0 and treat it as a pseudonym. Treating data as a pseudonym opens up for de-anonymization attacks as described in section 3.1.

Even if it is not vulnerable to de-anonymisation it is vulnerable to codebook attacks. Encyclopaedia of Cryptography and Security describe it like this:

> [A] ciphertext-only attack, which would start with frequency analysis of the received blocks and attempts to guess their meaning. Ciphers with small block size are vulnerable to the Codebook attack, especially if used in the simplest Electronic Codebook mode of operation. [vTJ11, p. 216 "Codebook Attack"]

In this particular case we are not talking about block ciphers and blocks, but about entirely encrypted data values, but if the space of possible cleartext is too small the number of collisions would still allow a user to guess the value if they know the probability distribution. If they don't know the probability distribution they will be able to calculate one, even if they can't get the values.

This problem was also emphasised by Bellare, Boldyreva and O'Neill [BBO07] in their searchable encryption system.

> Our schemes only provide privacy for plaintext that have high min-entropy. (This is inherent in being deterministic or efficiently searchable, not a weakness of our particular constructs.) [BBO07]

because big data is probably bigger this is going to add an even higher requirement to the entropy/value space required. The quote above specifically refers to the attacker simply being able to encrypt al possible values and then have their ciphertexts for de-anonymisation, but this would also apply to frequency analysis or any form of guess-mapping.

If searchability and deterministic operation is still desirable it is possible to use a semi-deterministic system. For this example AES with CBC is used. The AES key is the same for all values, but the CBC initialisation vector may vary within a small space. The search algorithm would then have to go trough the entire space of possible initialisation vectors and search for each possibility. This will add entropy to the plaintext-space equivalent to the space of possible initialisation vectors. It will also introduce an additional computing cost equivalent to brute-forcing the entire initialisation vector space, and it will come at the cost of being able to do encrypted joins. A related scheme for making data more searchable (or filterable) is discussed in section 7.6

Is this sufficient? It depends on the nature of the data in question. The data designer just has be aware that the deterministic property of encryption schemes will leave the data open to analytical schemes.

7.6 Filtering data pre-decryption

In section 7.5 a semi-deterministic system for a searchable database was discussed. This section will discuss a similar system that uses a trimming mechanism rather than an entropy-adding mechanism. The trimming mechanism is an inverted version of the application of bloom filters to big data [CML14].

A Bloom filter [Blo70] is a bitmap for storing hash values. The basic idea is that in a n bit bitmap a hash with the value v will set the bit numbered $vMODn$ in the bitmap to 1. More bits are set to 1 as more values are added to the set the filter represents. It is also possible to add multiple hashes of the same data can be added to the filter. The exact mathematical reason for this is a probability of a false positive is approximately [FCAB00]:

$$\left(1 - e^{-\frac{kn}{m}}\right)^k$$

where m is the number of bits in the filter and n is the number of values in the filter and k is the number of hash functions. For a separate filters with a total of m bits the function $(k = 1)$ becomes:

$$\prod_{i=0}^{a}\left(1 - e^{-\frac{kn}{m_i}}\right)^k = \left(\left(1 - e^{\frac{kn}{m/a}}\right)^k\right)^a = \left(1 - e^{\frac{n}{m/a}}\right)^a = \left(1 - e^{\frac{an}{m}}\right)^a$$

If we insert $a = k$ for k hash functions we get the exact same expression. This is on the approximated expression, but it proves that they re approximately equivalent.

To check if a value is in the set, one simply check if the appropriate bits are flipped. The bloom filter guarantees no false negative values, but allows for false positives. The false-positive but no false-negative property is the property that is going to be re-used.

The idea is to use the false-positive vs false negative tradeoff demonstrated by Bloom [Blo70] and use this to filter out negative matches before the data is being decrypted. The other goal is to still have enough remaining false positives that an attacker trying to do analysis is unable to obtain sufficient entropy to be able to do any meaningful analysis.

The way of doing this is to store a low bit-count hash of the cleartext value. The hash will reduce the “manual” search space by 2^n for a n-bit hash. For file oriented big data where the data is not indexed or ordered and a search would probably be an exhaustive search this is a considerable improvement if the decryption process is significantly slower than the hashing process.

The resulting hash will work as an aggregation value. This means that one hash value represent more than one value, but as the same time the same value has only one hash. The security of this system is in the number of collisions. Because an attacker is unable to isolate the distinct values only the hash aggregate of multiple values the statistical distribution may bleed away in the different values. The exact number of values represented by each hash vary, but the average number for a good hash function is 2^{m-n} where n is the number of bits in the hash function, and m is the number of bits entropy in the value space. For the record, bits of entropy is $m = log_2 (count(V))$ where V is the value space.

The advantage this method has over the one described in section 7.5 is that this method will mask statistical patterns by aggregating them while the semi-deterministic method will mask it by splitting up the data into smaller groups that still has the statistical distributions, but fewer samples.

In summary: The method described here gives similar speedups to the method described in for a level of protection that is slightly higher if the data has distinct distributions that can be revealed by statistics.

7.7 Impacts of Cipher Operation modes

The different modes of operation might have different impacts on the encryption speeds. The

7.7.1 Benchmark comparing operating modes

The benchmark used utilises the same framework as used in chapter 5. Each combination of cipher modes are run 1000 times and the elapsed tie in seconds is reported as milliseconds. These particular tests were carried out on a MacBook Pro with a "Intel(R) Core(TM) i5-5287U (Broadwell) @2.9GHz" CPU. The CPU has support for AES-NI and is superscalar. It is also a mobile processor, so it is possible that there has been made compromises in favour of power consumption compared to a server processor. The JVM has been tested and reports to utilise AES-NI, but no tools are available to confirm this.

The operation modes tested are the same as those described in subsection 4.6.1, even though some of them do not seem to be proper operating modes, at least not in the way they are described.

OpenSSL speed test

OpenSSL has a benchmark suite, and it has a AES-NI implementation. To demonstrate the impact of using AES-NI a simple run of the benchmarks were made. At the beginning of each run the command for the run is stated. OpenSSL benchmarks by allocating a block and encrypting/decrypting the block repeatedly for 3 second.

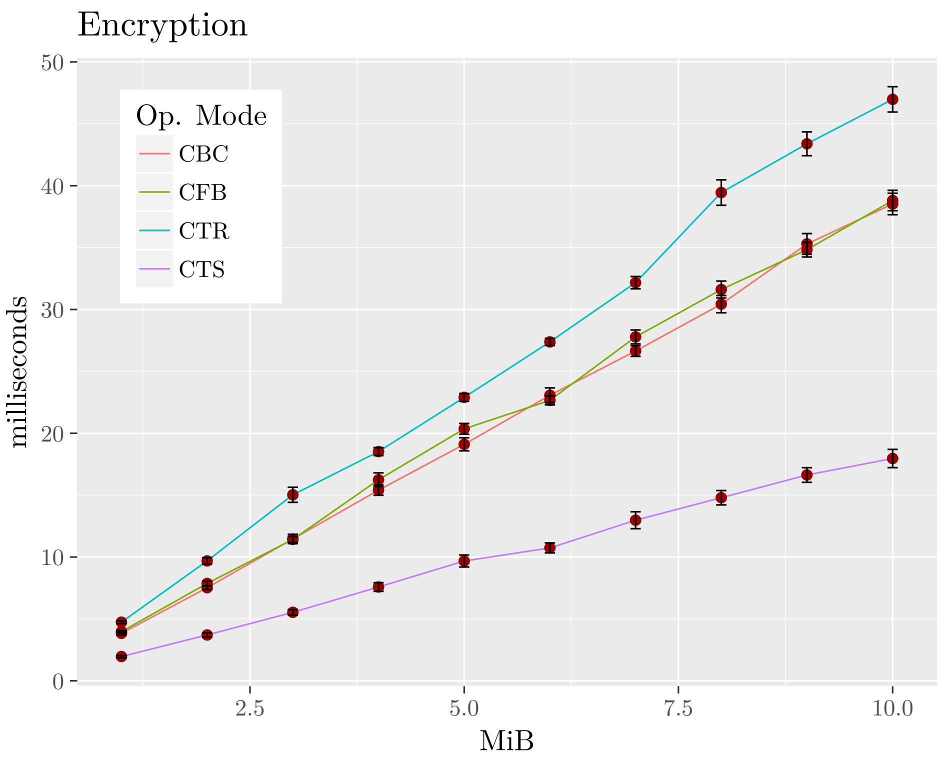


Figure 7.3: Comparison of block encryption modes 1; lower is better/faster

In the results, each column represents the data-block size in each run, the numbers are Kbytes er second, the k at the end are not part of the "number". Inverted comma thousand markers added manually.

```
CBC - encrypt with AES-NI
$ openssl speed -evp aes-128-cbc
16 bytes     64 bytes     256 bytes    1024 bytes   8192 bytes
646'811.75k  714'955.19k  729'208.22k  735'419.79k  732'106.54k
```

This result should be the result of an operation impossible to parallelise. The EVP flag uses an encryption interface that detects the optimal encryption mode, in this case that is AES-NI.

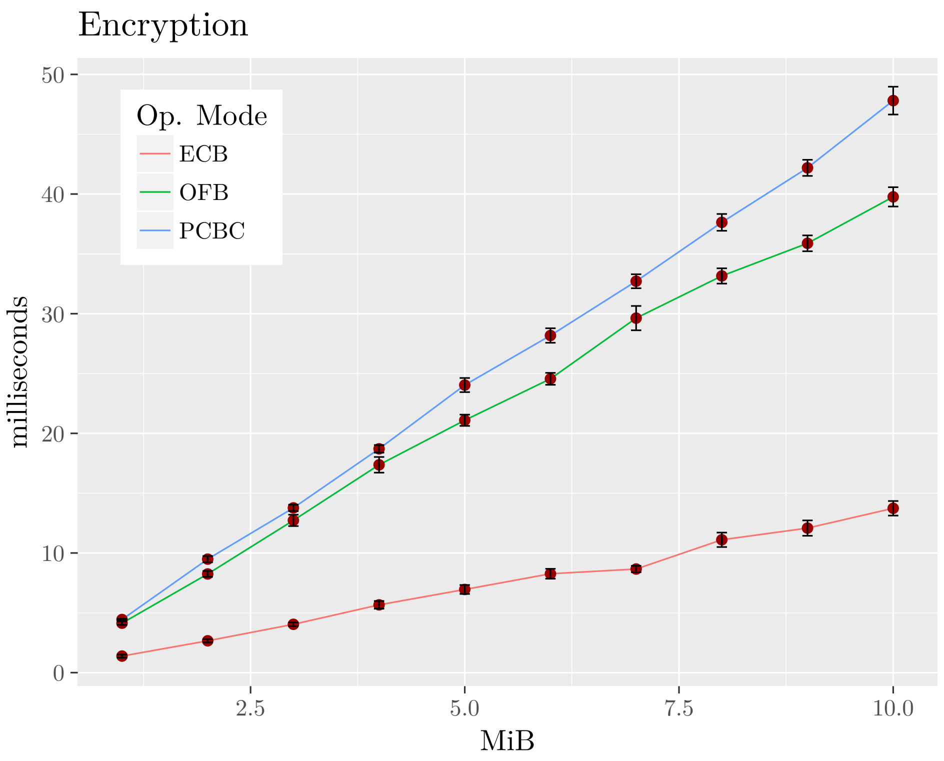


Figure 7.4: Comparison of block encryption modes 2; lower i better/faster

```
CBC - decrypt with AES-NI
$ openssl speed -evp aes-128-cbc -decrypt
16 bytes     64 bytes       256 bytes      1024 bytes
624’984.67k  2’459’025.23k  4’417’875.74k  4’933’374.44k
8192 bytes
5’066’444.11k
```

This is an improvement of $5066/732 \approx 6.9$ times on the $8192B$ data size. This is on a single threaded application on a superscalar CPU.

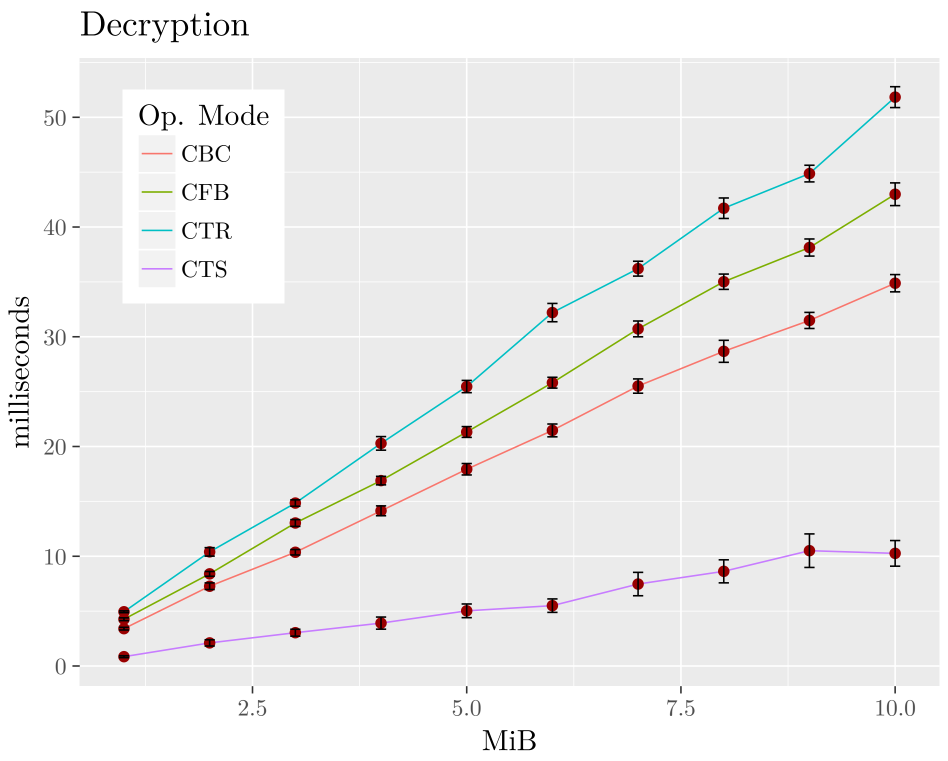


Figure 7.5: Comparison of block decryption modes 1; lower is better/faster

```
CTR - decrypt with AES-NI
$ openssl speed -evp aes-128-ctr -decrypt
16 bytes     64 bytes       256 bytes      1024 bytes
497’259.15k  1’587’017.44k  3’379’801.51k  4’495’538.00k
8192 bytes
4’904’928.28k
```

Counter mode is almost as fast as CBC decryption, but not quite.

```
CBC - encrypt without AES-NI
$ OPENSSL_ia32cap="~0x200000200000000" openssl speed -evp aes-128-cbc
16 bytes      64 bytes      256 bytes     1024 bytes    8192 bytes
279’831.76k   301’702.01k   319’134.36k   323’788.04k   322’958.43k
```

Just to do a comparison, the non-EVP mode defaults to software AES. The result is a $732/322 \approx 2.2$ times speedup by using AES-NI ($8192B$).

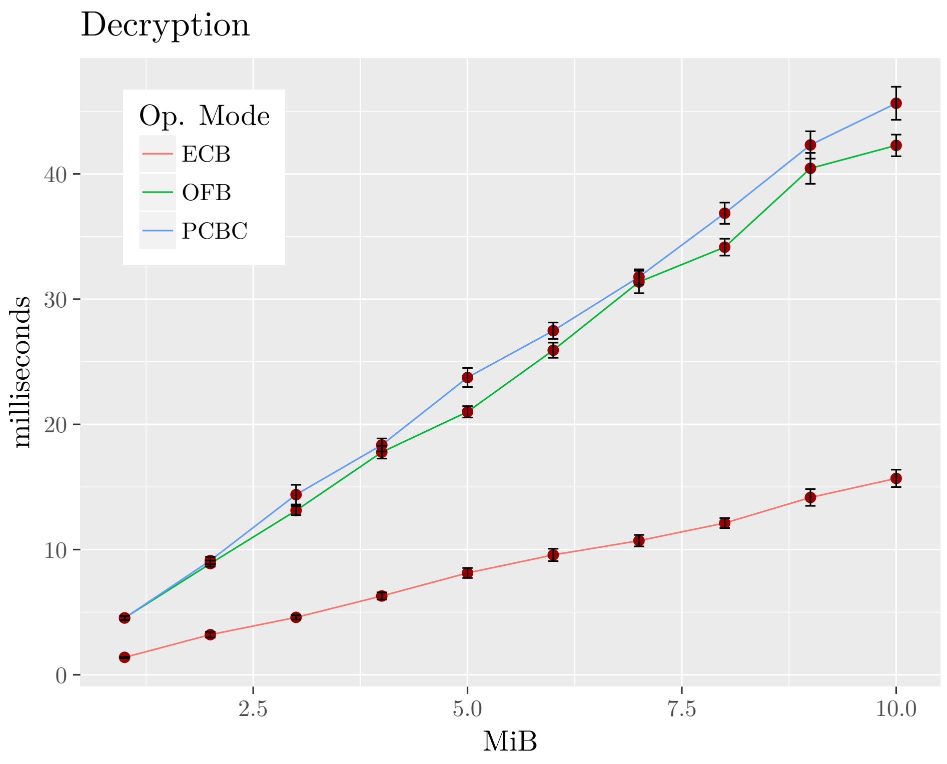


Figure 7.6: Comparison of block decryption modes 2; lower i better/faster

```
CBC - decrypt without AES-NI
$ OPENSSL_ia32cap="~0x200000200000000" openssl speed -evp \
aes-128-cbc -decrypt
16 bytes      64 bytes     256 bytes     1024 bytes    8192 bytes
83’519.80k    94’755.74k   369’015.73k   431’205.70k   442’515.68k
```

How about decryption? The result is a 5066/442 ≈ 12.0 times speedup. Note that the performance increase gained is a lot lower than the increase from the AES-NI enabled tests.

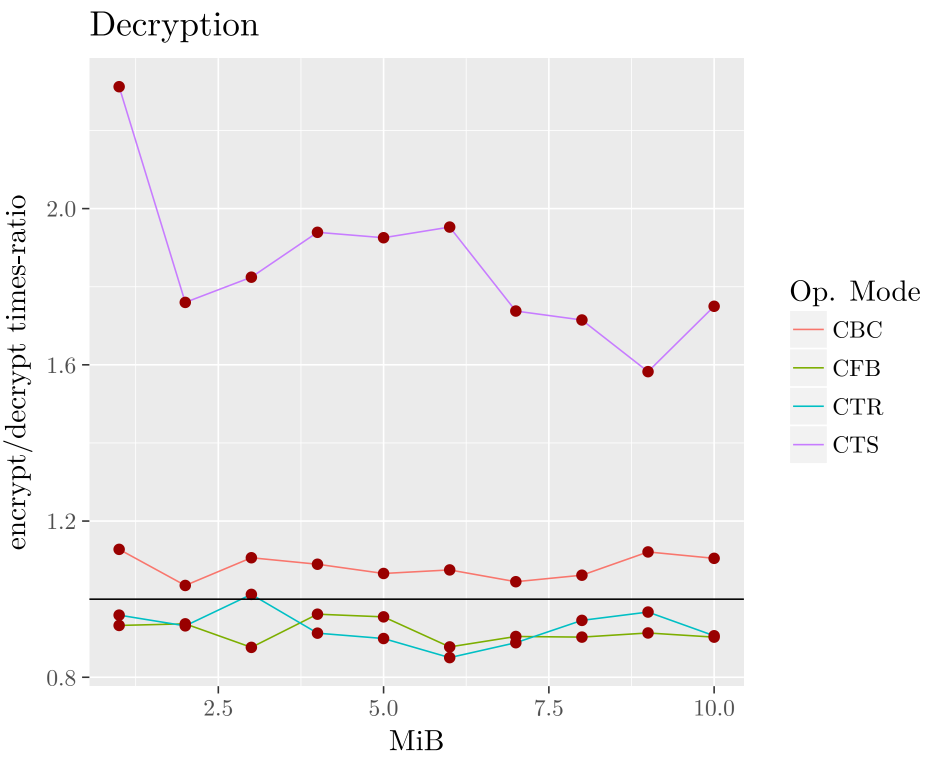


Figure 7.7: Comparison of encryption/decryption speed rates for AES
Higher number means that encryption took longer and encryption speed is slower.
1.0 is equal speed.
Should ideally be linear with data size, and even then no real growth.

```
CBC - encrypt without AES-NI
$ openssl speed -evp aes-128-cbc -multi 4 -decrypt
16 bytes       64 bytes       256 bytes      1024 bytes
1’768’846.48k  5’506’180.20k  8’373’870.76k  9’792’298.33k
8192 bytes
10’267’895.95k
```

This processor has 2 hyperthreaded cores. Hyperthreading just means that the processor uses superscalarism on two instruction streams instead of one. Logically it grants the Operating system 2 cores per physical core. Running the benchmark with 4 threads grants approximately 2 times the speed. This is a good indicator that the processor is capable of scheduling the OpenSSL to fully utilise the AES-NI resources in each core.

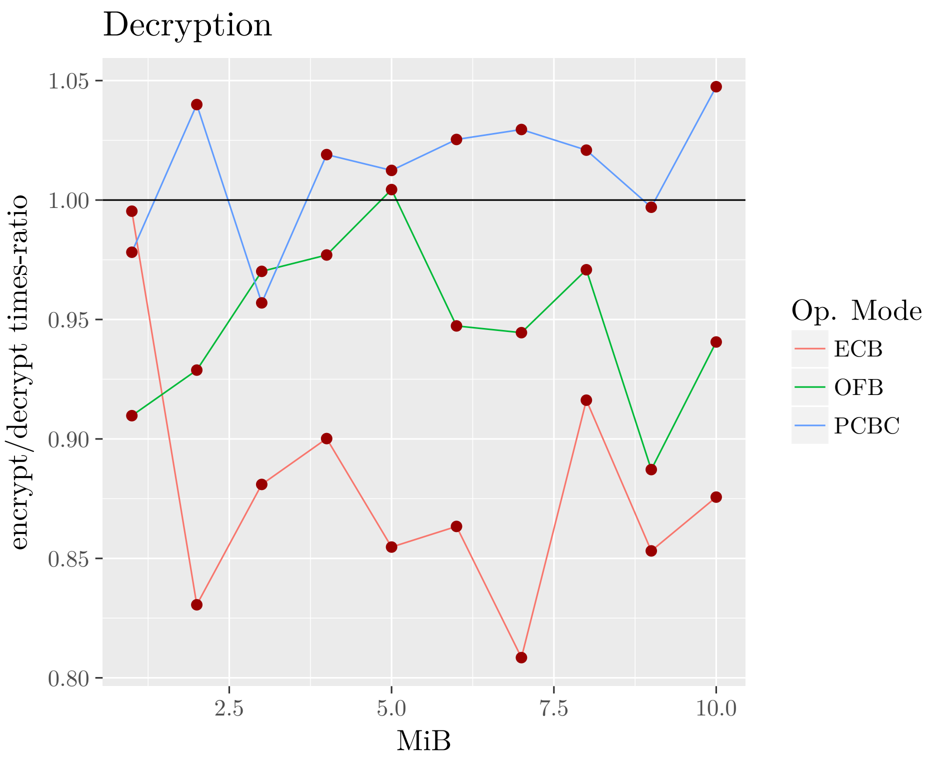


Figure 7.8: Comparison of encryption/decryption speed rates for AES
Higher number means that encryption took longer and encryption speed is slower. 1.0 is equal speed. Should ideally be linear with data size, and even then no real growth.

```
RSA4096
$ openssl speed rsa4096
sign       verify       sign/s   verify/s
0.004728s  0.000068s    211.5    14721.7
```

Just to have a result to compare to those in chapter 5 a 4096 bit RSA test. The results here are almost identical to the preliminary results for chapter 5 run on the same machine, not included in final report.

Results

These results are quite surprising. In Figure 7.3 and Figure 7.5 CounTeR mode (CTR) mode is the slowest operating mode, even though it is the mode that is best suited for superscalar parallelism. Secondly in Figure 7.7 CBC showed nothing close to the level of parallelism acceleration seen in the OpenSSL test. The speeds achieved are also somewhat disappointing. CBC decryption decrypted a $10MB$ block in $35ms$ or about $285MB/s$, compared to $5GB/s$ for OpenSSL's implementation.

Why are these results so slow? It is tempting to blame Java for not using the AES-NI instructions. An early test with OpenSSL (not using the evp bindings) had half shis speed; an updated result that used the evp bindings showed higher speeds than what Java offered. There is no way of testing this theory now.
Another possibility could be that the implementation only uses AES-NI for AES-rounds, and all the multiplexing takes place inside the JVM. If this is the case the performance impact is due to the JVM instruction stream being so long or unpredictable. The lack of difference in results form the software implementation of AES supports this theory.
The third possibility is that the JVM garbage collector performs garbage collection during the benchmark and skews the results. If this is the case than the result still represents real world expected performance. Because the variance is so low it also means that this is a consistent, reproducible result.

Considering that Java seem to handle RSA at speeds comparable to what OpenSSL does, these results are concerning. There could be a lot of reasons whu it has performance issues, but discovering these will have to be the work of someone else. [oraa]

The patch implementing this feature speaks of a conversion of the "expanded key" from a 32-bit big-endian to 128-bit words in little endian. If this conversion happens every time a AES cipher operation is performed, this could account for the difficulty of out-of-ordering the instruction stream. Or it could not, that would have to be someone else's research project.

Conclusions

These results say that at least on "Java HotSpot(TM) 64-Bit Server VM (1.8)" running on a "Intel(R) Core(TM) i5-5287U"; the choice for operating mode for block ciphers in Java (javax.crypto.cipher) is of little consequence to encryptions and decryption speeds. It also turns out that all ciphers are within $\pm 20\%$ on encryption/decryption ratio. This probably translates into other versions of the HotSpot JVM and other Intel processors, but it is also possible that desktop or server CPUs are better at optimising JVM at the cost of more power consumption.

The results also show that javax.crypto.cipher is quite slow compared to OpenSSL's implementation of AES, so it might be worthwhile to develop native middleware for an encryption heavy protection system.

7.8 Data padding

For some data the length may add some data leaks, examples of this is in attacks against compression enabled crypto [CVE12]. Another case is variable bit-rate Voice over IP (VoIP) that uses the same kinds of properties [WBC$^+$08].

In the cases above a lot of the data is leaked trough the compressibility of the data and the size of it after encryption. The same can be applied if the data simply has different sizes. The smallest resolution available to a malicious analyst is going to be the block size of the encryption algorithm.

If the data is padded to 64 byte multiples the entropy in the record length is reduced to about a quarter compared to 16 Byte AES blocks. This number is if we assume a uniform distribution of data sizes. If the data already fell mostly into 64 Byte groupings (see Figure 7.9) this particular masking would be practically useless. Even though a uniform distribution in total would make a frequency attack impossible, any variable that changes the distribution of content length is guessable. On the other hand if the interesting data samples resided in the low zones (see Figure 7.9) the aggregation would be very useful. There is no scenario where increasing the size steps would be useful to an attacker looking at data sizes.

7.9 Hardware acceleration

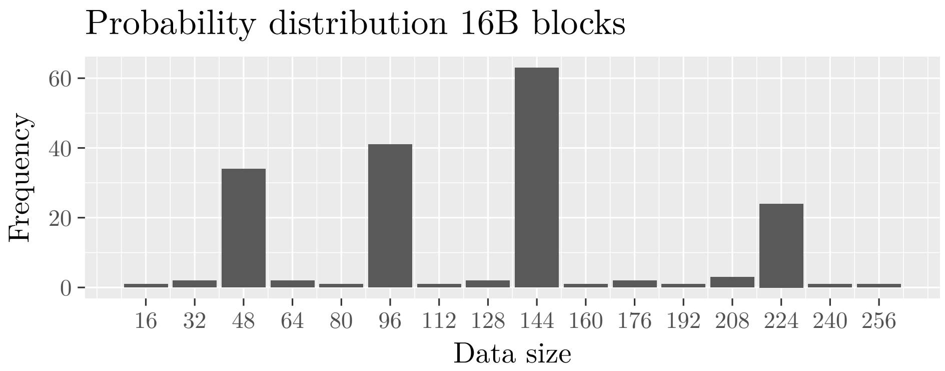
In section 7.7 it was demonstrated that hardware acceleration is effective for some cryptographic systems. Is this feasible to use for a discrete logarithm-based cipher? In this section it will be made an attempt to calculate how expensive it would be to implement this on a chip. This calculation has probably been done before, but it would be interesting to look at the numbers.

7.9.1 Basic blocks

For the simplicity two logical building blocks are used. These are used in some Field-Programmable Gate Arrays (FPGAs) the numbers denote the number of input bits and the number of output bits. Each of these logic units can fulfil any truth diagram. The first is a 2-to-1 gate the second is a 4-to-2 gate. To reduce the number of constants the number of gates are just represented as a complex number, with 2-to-1 on the real axis and 4-to-2 on the imaginary axis. To avoid confusion all functions are written as an upper case letter, and all variables are lower case.

7.9.2 Binary Multipliers

There are several ways of making efficient multipliers, one is Wallace trees [Wal64], another is Dadda multipliers [Dad65]. Because it is not easy to get a gate count



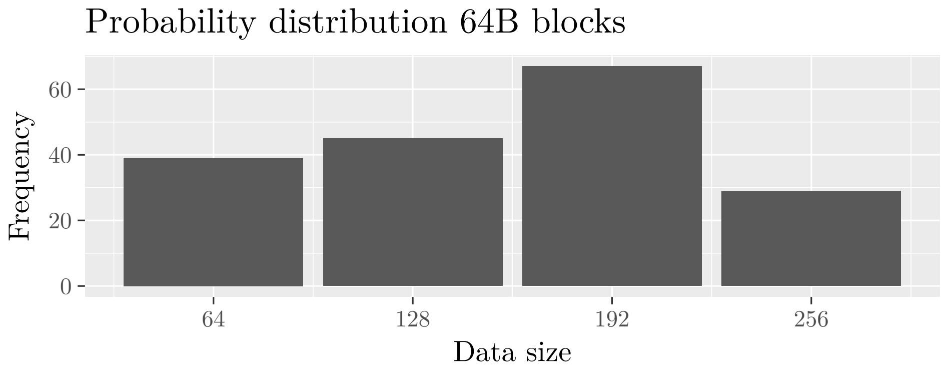


Figure 7.9: Example of a distribution that is resistant to padding.
Because 3 out of 4 16B blocks in each 64B block is negligible one could just assume them to be zero.
this is a constructed worst case scenario.

from these, the assumption is going to be that the number is $m * n$ 2-way ANDs and $m * n$ full adders. The formula then becomes:

$$M(m, n) = m * n + i(m * n)$$

A multiplier always has $m + n$ output bits, the proof for the case $m = n$ is $\left\lceil \frac{log_2\left((2^n-1)^2\right)}{n} \right\rceil = 2$. A quick script to check that $\lceil log_2\left((2^n - 1)\left(2^m - 1\right)\right)\rceil = m + n$ holds true, shows that it is true for $m, n > 1 \wedge m, n < 10000$.

7.9.3 1 Bit chain exponensiator

A basic (and possibly naive) implementation of a modular exponensiator is described in Figure 7.10. The 1-bit modules are shown in Figure 7.11 For this the original base

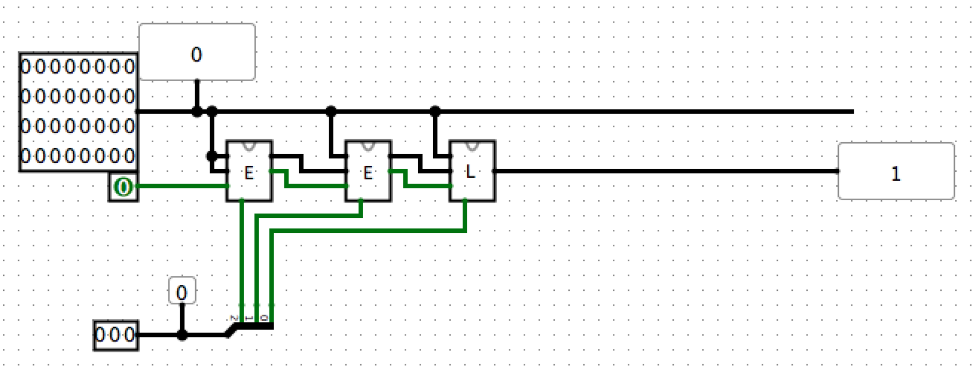


Figure 7.10: Chain of 1-bit exponensiator chains. The E-blocks are described in Figure 7.11, while the L-block is described in Figure 7.12

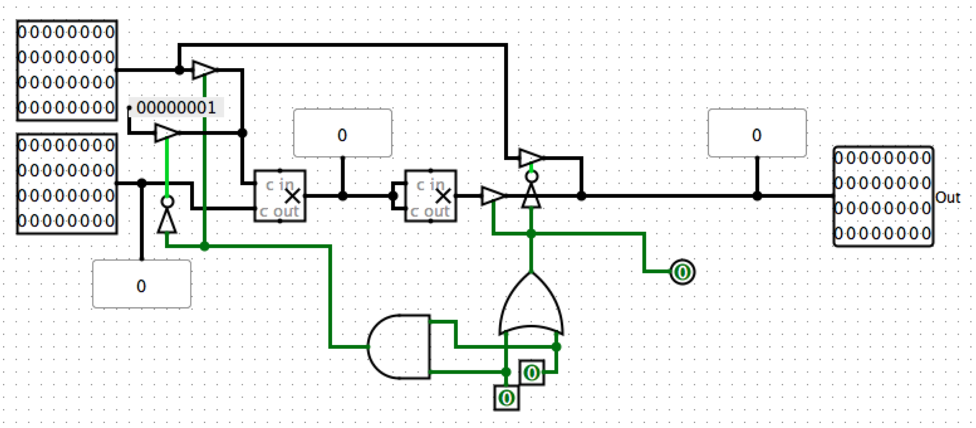


Figure 7.11: 1 bit of an exponensiator. This has a squarer at the output to chain to another cell.
Because this is intended to handle different size exponents it has a bypass feature and a “MSB has happened” output.

is n bits, the chain input is m bits. This design requires $2n$ selectors (all other bits an be hard coded to 0), a $m * n$ bit multiplier, a $m + n$ bit squarer (multiplier in this case), and $2(m + n)$ selectors. The 2 not gates depicted can be implemented in a truth diagram change for the selectors so the additional logic is 2 gates. The final function becomes:

$$E_b(m, n) = 3m + 4n + 2 + M(m, n) + M(m + n, m + n)$$

The output here is $2(m + n)$ bits wide.

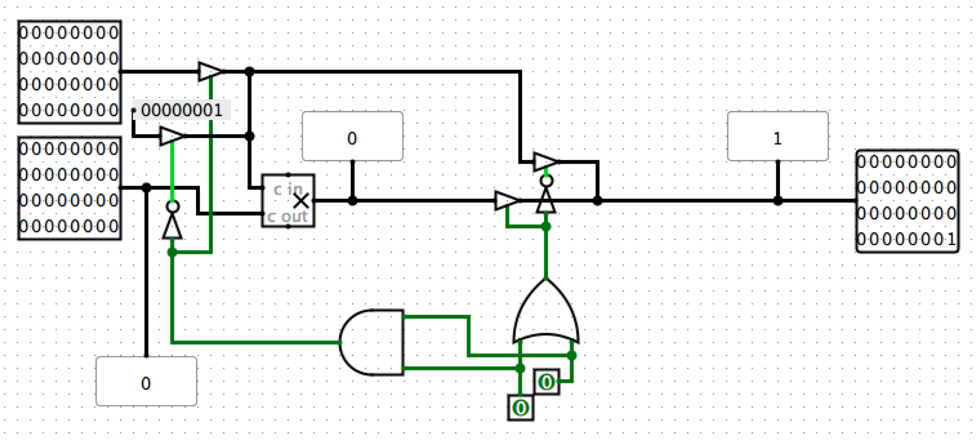


Figure 7.12: 1 bit of an exponensiator last bit. No squarer on the output. There will be no bits chained after this module. The module has been wired to handle the special case of the exponent being zero.

The least significant bit does not need the squarer at the end so it requires fewer bits (see Figure 7.12). This means that it requires $2n$ selectors for input, a $m * n$ multiplier, $m + n$ output selectors, and 2 regular gates. In total:

$$E_l(m,n) = 3n + m + 2 + M(m,n)$$

The output is then $m + n$ bits wide.

7.9.4 Adding the modules together

The difficulty is going to be calculating m_i, so the lazy way is to define it as a recursive function. Because for the first iteration $m = n$ there is a few simplifications to make:

$$W(n,i) \begin{cases} i = 1 & n \\ i = 2 & 3n \\ i > 2 & 2W(m,n,i-1) + n \end{cases}$$

The total for a n^a exponensiator (n, a in bits) would be:

$$\sum_{i=1}^{a-1} \left[E_l\left(W\left(n,i\right),n\right)\right] + E_l\left(W\left(n,a\right)\right)$$

7.9.5 Gates in an exponensiator

Because of how simple this is to implement programatically, we try to apply this to a simple system, a 256 bit AES key encrypted using RSA. The public exponent are often $2^{16} + 1$. As a result we would need a 256^{16} exponensiator. Plug this into the program (section C.9) and the number that comes out is: $93'833'648'016'160 + 93'833'580'904'448j$, or approximately 186 trillion gates.

According to Intel the Intel Core [intb] (the quad core version) had ≈ 1.5 billion transistors. Going by Moore's law from 2011; 6 years makes for 4 doublings, so the number should be approximately $1.5 * 2^4 = 24$ billion, or smaller by approximately a factor of 7700. Thus the hardware logarithm accelerator is unfeasible using this naive implementation.

7.9.6 FPGA sizes for reference

An Intel FPGA (formerly known as Altera) in the Arria 10 family [inta] can have between $160K$ and $1.5M$ Logic units in addition to $60K$ and $427K$ so called Adaptive Logic Modules. For the purposes here, we are treating a ALM as an adder. From reading the specifications it is definitely possible to implement this more efficiently by making multi-bit adders on ALMs, but for simplicity this possibility is ignored. The LUs are 4-input so 2 can make a 2-to-4 unit.

7.9.7 Wide multipliers

Implementing a $512 * 512$ bit multiplier would require $(262'144 + 262'144j)$ elements that is feasible on a mid to high end unit of the Arria 10 FPGA family. This number is something that could be used in a acceleration unit. This is 16 times wider than the 32 bit multiplications available in x86 using something like the PMULUDQ instruction [intc, p.5].

Using the PMULUDQ instruction this would take $16 * 16 = 256$ multiplications and 256 64-bit adds with carry. Roughly speaking this means that if the 32-bit multiplier is 256 times faster, the wide multiplier would still be slightly faster.

Even if a less extreme case is used, a 64-bit multiplier instruction would reduce the number of instructions needed to do large multiplications by a quarter, and thus improving the performance accordingly.

7.9.8 Multiplication algorithms

How beneficial is it to implement a wider multiplier?

The traditional long multiplication algorithm taught in grade schools have a complexity of n^2 multiplications, where n is the number of blocks the factors would have to be broken down to.

The Kartasuba algorithm [KO62] is a 2-way divide and conquer algorithm that has the complexity of $n^{log_2(3)} \approx n^{1.585}$ multiplications.

This is actually a special case of the Toom-Cook algorithm described in it's current form in Cook's Ph.D. thesis [CA69]. The complexity is $C(k)n^{\frac{log(2k-1)}{log(k)}}$ where k is the number of partitions. There is several pages in Knuth's The Art of Computer Programming, Volume 2 [Knu97, (p294)4.3.3] dedicated to the multiplication problem.

There are other algorithms that uses Fourier transforms like the Schönhage–Strassen [SS71]. Large number libraries like GNU MP [gnu] set the threshold of the advantage to somewhere between 3000 and 10000 times the multiplier width.

The Montgomery modular multiplication algorithm [Mon85] is an algorithm that is used to do exponentiation with modulus and keep the number of bits down during the process. There are claims of it gaining from using Kartasuba with sufficiently large modulus, but these claims have not been verified.

Chapter 8
Discussions and Conclusions

Back in section 1.3 the goals of this thesis was explained. This chapter will try to sum up those goals and their answers.

8.1 Viable cryptographic access control

> Is cryptography a good solution for access control?
> If there is a cryptographic access control solution does this scale to big data applications?

If the access control policy is simple it is as simple as giving the user the key to each dataset they have access to. As the policies get more complex the number of keys goes up. While the hierarchical approach [AT83] offers more flexibility it is still pretty restricted.

The approach of using ABE to make policies grants a lot of flexibility, but at the cost of computing power. For GPSW the scaling seems to be equivalent to n times RSA for somewhere between 3072 and 4096 bits in the modulus; (section 5.3, Figure 5.4, and Figure 5.5) where n is the number of attributes in the message. A system and a key policy would have to maximise the data per GPSW decryption and minimise the number of attributes in a block.

Even though it would take a lot of planning and work to implement, a ABE based solution would be possible to implement using a strategy similar to the one described in section 6.6. The disadvantage is the need for regular “vacuuming” to join ABE blocks together for efficiency (described briefly in subsection 6.6.3).

8.2 More “traditional” alternatives

If there are no cryptographic mechanisms that allow practically usable access control are there any other ways of achieving General Data Protection Regulation (GDPR) compliance?

One possibility for access control is to do as described in section 6.8, and simply use the existing access control systems long with integrated at-rest encryption systems. While this report does not look into the integration between Apache Ranger’s access policy and the key management module, an integration between the two would make up a partially crypto-enforced access control system.

When it comes to GDPR compliance there are other concerns. For instance the “right to be forgotten” provision (section B.2) that enforces a users right to tell any organisation to delete all data they have on that individual. Because this requires a certain level of control over individual records a file-based data lake may be sub-optimal for that purpose. Therefore it may be beneficial to store sensitive data in a table-like storage system like Apache Hive or Apache HBase where individual records can be deleted on individual basis.

The advantage of using a table system is that it allows for more granularity in the access control policy. As described in section 6.8 table-like systems advertise with more granularity of access control than HDFS does. This is purely due to the fact that he table systems have a deeper understanding of what data they are handling and what the queries look like.

The final conclusion would be that the due to the control requirements of GDPR nullifies the advantage of using a data lake to store raw files. This conclusion is subject to lawyers and may be completely wrong. Due to this advantage-nullification assumption it is possible to declare a table-based storage method as clearly advantageous over a raw-file data lake. Had this not been the case, the difference would be dependent on how good the strategy for deleting records of a user is.

8.3 Surprising/Interesting results

Are there other avenues that could be interesting to look into into?

One of the most interesting and surprising results of this project was the advantage of hardware based acceleration shown in section 7.7.1. While there were expected some speed advantages a 15.8 (software CBC encryption against hardware CBC decryption) times speedup from different hardware acceleration in combination is stunning. Also this number assumes that the the software implementation did not have any advantages from superscalarism. This assumption is false, because the cipher was specifically designed to be able to use parallelism.

8.4 Future research

Are there other avenues that could be interesting to look into into? A basis for further research.

A proper threat analysis should be done of the table system approach. It may be that for some reason there are disadvantages that outweigh the advantages seen on a first look.

Schemes for improving ABE should be looked into. Algorithms for optimising access control and attributes would be extremely beneficial to have. If the policy system could automatically optimise the attributes to the policies in such a way that the number of attributes applied to a message is kept to a minimum, the solution may still be viable.

A wide multiplier co-processor for handling discrete logarithm cryptography could be desirable as a solution to the speed issues faced by public-key cryptography based on discrete logarithms.

References

[ABKN16] Alex Aiken, Utpal Banerjee, Arun Kejariwal, and Alexandru Nicolau. *Instruction Level Parallelism.* Springer, 2016.

[aes] Cnss policy no. 15, fact sheet no. 1 national policy on the use of the advanced encryption standard (aes) to protect national security systems and national security information http://csrc.nist.gov/groups/STM/cmvp/documents/CNSS15FS.pdf.

[amaa] Cloudhsm - amazon web services. Retrieved on 2017.04 - Website at https://aws.amazon.com/cloudhsm/.

[amab] Elastic mapreduce - amazon web services. Retrieved on 2017.05 - Website at https://aws.amazon.com/articles/Elastic-MapReduce/4926593393724923.

[arm] "arm with built-in aes cryptographic module" - arm community forums. Retrieved on 2017.04 - Website at https://community.arm.com/processors/f/discussions/1424/arm-with-built-in-aes-cryptographic-module.

[AT83] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 535–552, 2007.

[BD13] Gérard Blanchet and Bertrand Dupouy. *Computer Architecture.* Wiley-ISTE, 2013.

[Ber15] Elisa Bertino. Big data - security and privacy. In *2015 IEEE International Congress on Big Data, New York City, NY, USA, June 27 - July 2, 2015*, pages 757–761, 2015.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. pages 313–317, 1979.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[CA69] Stephen A Cook and Stål O Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[CDG$^+$06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A distributed storage system for structured data (awarded best paper!). In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 205–218, 2006.

[cel] Celery project. Retrieved on 2017.05 - Website at http://www.celeryproject.org/.

[CGM13] Omar Choudary, Felix Gröbert, and Joachim Metz. Security analysis and decryption of filevault 2. In *Advances in Digital Forensics IX - 9th IFIP WG 11.9 International Conference on Digital Forensics, Orlando, FL, USA, January 28-30, 2013, Revised Selected Papers*, pages 349–363, 2013.

[CJL$^+$15] Mandy Chessell, Nigel L Jones, Jay Limburn, David Radley, and Kevin Shank. *Designing and Operating a Data Reservoir.* IBM Redbooks, 2015.

[cloa] Apache sentry - cloudera. Retrieved on 2017.04 - Website at https://www.cloudera.com/products/open-source/apache-hadoop/apache-sentry.html.

[clob] Cloudera navigator - cloudera. Retrieved on 2017.04 - Website at https://www.cloudera.com/products/product-components/cloudera-navigator.html.

[cloc] Hdfs encryption - cloudera. Retrieved 2017.03 - Website at https://www.cloudera.com/documentation/enterprise/5-4-x/topics/cdh_sg_hdfs_encryption.html.

[CML14] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *MONET*, 19(2):171–209, 2014.

[Com13a] Wikimedia Commons. Cbc decryption. https://commons.wikimedia.org/wiki/CBC_decryption.svg, 2013. File: CBC_decryption.svg; Licence: Public Domain.

[Com13b] Wikimedia Commons. Cbc encryption. https://commons.wikimedia.org/wiki/CBC_encryption.svg, 2013. File: CBC_encryption.svg; Licence: Public Domain.

[Com13c] Wikimedia Commons. Ctr decryption. https://commons.wikimedia.org/wiki/CTR_decryption_2.svg, 2013. File: CTR_decryption_2.svg; Licence: Public Domain.

[Com13d] Wikimedia Commons. Ctr encryption. https://commons.wikimedia.org/wiki/CTR_encryption_2.svg, 2013. File: CTR_encryption_2.svg; Licence: Public Domain.

[Com13e] Wikimedia Commons. Ecb decryption. https://commons.wikimedia.org/wiki/ECB_decryption.svg, 2013. File: ECB_decryption.svg; Licence: Public Domain.

[Com13f] Wikimedia Commons. Ecb encryption. https://commons.wikimedia.org/wiki/ECB_encryption.svg, 2013. File: ECB_encryption.svg; Licence: Public Domain.

[CSN$^+$14] Mandy Chessell, Ferd Scheepers, Nhan Nguyen, Ruud van Kessel, and Ron van der Starre. Governing and managing big data for analytics and decision makers. *IBM Redguides for Business Leaders*, 2014.

[CVE12] CVE-2012-4929 - crime. Available from MITRE, CVE-ID CVE-2012-4929., December 3 2012.

[CVE16a] CVE-2016-5195 - dirty cow. Available from MITRE, CVE-ID CVE-2016-5195., May 31 2016.

[CVE16b] CVE-2016-7545 - selinux bypass. Available from MITRE, CVE-ID CVE-2016-7545., September 9 2016.

[CVE17] CVE-2016-7545 - net/dccp/input.c bypass. Available from MITRE, CVE-ID CVE-2017-6074., February 17 2017.

[Dad65] Luigi Dadda. Some schemes for parallel multipliers. *Alta frequenza*, 34(5):349–356, 1965.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[DI11] Angelo De Caro and Vincenzo Iovino. jpbc: Java pairing based cryptography. In *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pages 850–855, Kerkyra, Corfu, Greece, June 28 - July 1, 2011. IEEE.

[Don24] John Donne. *Devotions upon Emergent Occasions*. 1624 1624 1624.

[DR98] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, pages 277–284, 1998.

[Dra93] The journal of supercomputing - special issue on instruction-level parallelism. *J. Supercomput.*, 7(1-2), 1993.

[FCAB00] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[Gan] Balaji Ganesan. Best practices in hdfs authorization with apache ranger - hortonworks. Retrieved on 2017.03 - Website at https://hortonworks.com/blog/best-practices-in-hdfs-authorization-with-apache-ranger/.

[GAT$^+$15] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1917–1923, 2015.

[gdp] The general data protection regulation. http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf.

[Geo11] Lars George. *HBase - The Definitive Guide: Random Access to Your Planet-Size Data.* O'Reilly, 2011.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, pages 29–43, 2003.

[GHMJ14] Albana Gaba, Yeb Havinga, Henk-Jan Meijer, and Evert Jan. Privacy and security for analytics on healthcare data. 2014.

[gnu] Fft multiplication - gnu mp. Retrieved on 2017.05 - Website at https://gmplib.org/manual/FFT-Multiplication.html#FFT-Multiplication.

[God14] Matt Godbolt. x86 internals for fun and profit. Presentation at goto; conference: https://gotocon.com/chicago-2014/presentation/x86%20Internals%20for%20Fun%20and%20Profit; Video of presentation avialable on YouTube: https://www.youtube.com/watch?v=hgcNM-6wr34, May 21 2014.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.

[Gro] The PostgreSQL Global Development Group. Postgresql documentation chapter 2.6. - joins between tables.

[Gue12] Shay Gueron. *Intel® Advanced Encryption Standard (Intel® AES) Instructions Set.* Intel, 2012.

[HD14] Xueli Huang and Xiaojiang Du. Achieving big data privacy via hybrid cloud. In *2014 Proceedings IEEE INFOCOM Workshops, Toronto, ON, Canada, April 27 - May 2, 2014*, pages 512–517, 2014.

[HKF15] Vincent C. Hu, D. Richard Kuhn, and David F. Ferraiolo. Attribute-based access control. *IEEE Computer*, 48(2):85–88, 2015.

[HKN$^+$16a] Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 795–806, New York, NY, USA, 2016. ACM.

[HKN$^+$16b] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Managing google's data lake: an overview of the goods system. *IEEE Data Eng. Bull.*, 39(3):5–14, 2016.

[hora] Aapache rranger - hortonworks. Retrieved on 2017.03 - Website at: https://hortonworks.com/apache/ranger/.

[horb] Configuring and using hdfs data at rest encryption - hortonworks. Retrieved on 2017.03 - Website at https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.4/bk_hdfs_admin_tools/content/config-use-hdfs-encr.html.

[horc] Ranger-hive integration - hortonworks. Retrieved on 2017.03 - Website at https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.4/bk_Sys_Admin_Guides/content/ref-746ce51a-9bdc-4fef-85a6-69564089a8a6.1.html.

[hord] Ranger key management service - hortonworks. Retrieved on 2017.03 - Website at https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.4/bk_Ranger_KMS_Admin_Guide/content/ch_ranger_kms_overview.html.

[hore] Security and governance - hortonworks. Retrieved on 2017.04 - Website at https://hortonworks.com/solutions/security-and-governance/.

[inta] Arria 10 fpga - intel fpga / altera. Retrieved on 2017.05 - Website at https://www.altera.com/products/fpga/arria-series/arria-10/overview.html.

[intb] Fun facts exactly how small (and cool) is 22 nanometers? - intel promotion sheet. Retrieved on 2017.05 - Website at http://www.intel.com/content/dam/www/public/us/en/documents/corporate-information/history-moores-law-fun-facts-factsheet.pdf.

[intc] Intel vector functions instructions reference - intel. Retrieved on 2017.05 - Website at https://software.intel.com/sites/default/files/4f/5b/36945.

[Kar53] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

[KC13] Mohammed Korayem and David J. Crandall. De-anonymizing users across heterogeneous social computing platforms. In *Proceedings of the Seventh International Conference on Weblogs and Social Media, ICWSM 2013, Cambridge, Massachusetts, USA, July 8-11, 2013.*, 2013.

[KNR$^+$11] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[KO62] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962.

[Kor09] Jesse D. Kornblum. Implementing bitlocker drive encryption for forensic analysis. *Digital Investigation*, 5(3-4):75–84, 2009.

[Koz] Vladimir Kozlov. add intrinsics to use aes instructions https://bugs.openjdk.java.net/browse/JDK-8004381.

[KSW02] Günter Karjoth, Matthias Schunter, and Michael Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. In *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*, pages 69–84, 2002.

[Lab09] RSA Laboratories. Pkcs 11: Cryptographic token interface standard. Available from multiple sources, EMC listed https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm, 2009.

[LS16] Alice LaPlante and Ben Sharma. *Architecting Data Lakes.* O'Reilly Media, 2016.

[LZGP15] Shuyu Li, Tao Zhang, Jerry Gao, and Younghee Park. A sticky policy framework for big data security. In *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*, pages 130–137, 2015.

[Mas66] Abraham H. Maslow. *The Psychology of Science.* New York: Harper Row, 1966.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[MR12] Ashwin Machanavajjhala and Jerome P. Reiter. Big privacy: protecting confidentiality in big data. *ACM Crossroads*, 19(1):20–23, 2012.

[MSTZ14] Natalia G. Miloslavskaya, Mikhail Senatorov, Alexander I. Tolstoy, and Sergey Zapechnikov. Information security maintenance issues for big security-related data. In *2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014, Barcelona, Spain, August 27-29, 2014*, pages 361–366, 2014.

[NIS] Fips pub 81. Retrieved on 2017.04 - Website at http://csrc.nist.gov/publications/fips/fips81/fips81change3.pdf.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 111–125, 2008.

[NS09] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 173–187, 2009.

[ODS13] Aisling O'Driscoll, Jurate Daugelaite, and Roy D. Sleator. 'big data', hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, 46(5):774 – 781, 2013.

[oraa] "add intrinsics to use aes instructions" - openjdk bug tracker. Retrieved on 2017.04 - Website at https://bugs.openjdk.java.net/browse/JDK-7184394.

[orab] Java cryptography architecture standard algorithm name documentation - oracle. Retrieved on 2017.04 - Website at http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html.

[pci] Pci-dss security standards. Retrieved on 2017.04 - Website at https://www.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss.

[PM11] Siani Pearson and Marco Casassa Mont. Sticky policies: An approach for managing privacy across multiple parties. *IEEE Computer*, 44(9):60–68, 2011.

[PRZB11] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 85–100, 2011.

[Qui55] Willard V Quine. A way to simplify truth functions. *The American Mathematical Monthly*, 62(9):627–631, 1955.

[RBF] Reuters, Liana B. Baker, and Jim Finkle. Sony playstation suffers massive data breach. http://www.reuters.com/article/us-sony-stoldendata-idUSTRE73P6WB20110427.

[RFa] Reuters and Jim Finkle. Adobe data breach more extensive than previously disclosed. http://www.reuters.com/article/us-adobe-cyberattack-idUSBRE99S1DJ20131029.

[RFb] Reuters and Julia Fioretti. Google under fire from regulators on eu privacy ruling. http://www.reuters.com/article/us-google-eu-privacy-idUSKBN0FT1AZ20140724.

[Rpia] Raspberry pi model 2b - specifications. Retrieved on 2017.04 - Website at https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[Rpib] Raspgerry pi, raspbian os (debian variation). Retrieved on 2017.04 - Website at https://www.raspberrypi.org/downloads/raspbian/.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RSK$^+$10] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*, pages 297–312, 2010.

[Sch95] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., New York, NY, USA, 1995.

[SD15] James G. Shanahan and Laing Dai. Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 2323–2324, 2015.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3):281–292, 1971.

[Sti92] Douglas R. Stinson. An explication of secret sharing schemes. *Des. Codes Cryptography*, 2(4):357–390, 1992.

[Tay10] Ronald C. Taylor. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(S-12):S1, 2010.

[UN13] Jayakrishnan Unnikrishnan and Farid Movahedi Naini. De-anonymizing private data by matching statistics. In *51st Annual Allerton Conference on Communication, Control, and Computing, Allerton 2013, Allerton Park & Retreat Center, Monticello, IL, USA, October 2-4, 2013*, pages 1616–1623, 2013.

[VMD+13] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop YARN: yet another resource negotiator. In *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 5:1–5:16, 2013.

[vTJ11] Henk C. A. van Tilborg and Sushil Jajodia, editors. *Encyclopedia of Cryptography and Security, 2nd Ed.* Springer, 2011.

[WAE+15] Samuel Fosso Wambaa, Shahriar Akterc, Andrew Edwardsd, Geoffrey Chopine, and Denis Gnanzouf. How 'big data' can make big impact: Findings from a systematic review and a longitudinal case study. *International Journal of Production Economics, Forthcoming*, 2015.

[Wal64] Christopher S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. Electronic Computers*, 13(1):14–17, 1964.

[Wat12] Brent Waters. Functional encryption for regular languages. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 218–235, 2012.

[WBC+08] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 35–49, 2008.

[YJR+14] Kan Yang, Xiaohua Jia, Kui Ren, Ruitao Xie, and Liusheng Huang. Enabling efficient access control with dynamic policy updating for big data in the cloud. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, pages 2013–2021, 2014.

[Zha14] Liang Zhang. Enhancing data security in cloud computing: Build an encrypted personally controlled health records platform on indivo x https://sites.google.com/a/ualr.edu/reu-project-by-liang-zhang/home, 2014.

[ZYL13] Wenrong Zeng, Yuhao Yang, and Bo Luo. Access control for big data using data content. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 45–47, 2013.

Appendix A
Preliminary Result Graphs

Because some preliminary results were referenced it is necessary to include them as well, just for reference. This chapter is dedicated to preliminary graphs and results from unreliable test rigs that still turned out to be interesting or useful.

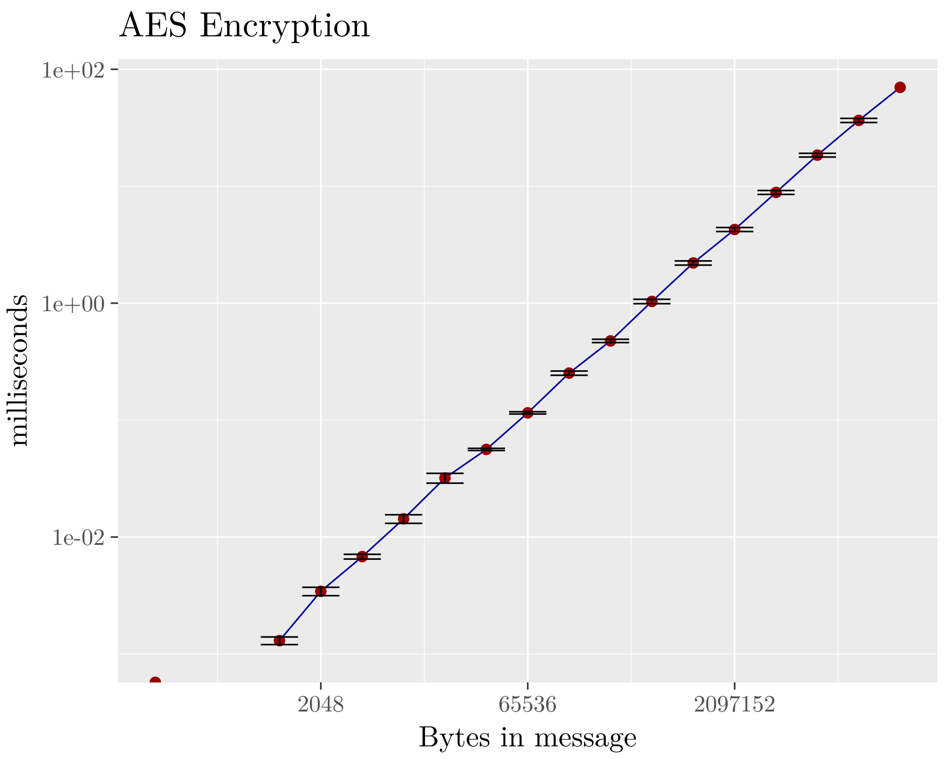


Figure A.1: AES Encryption times, Variable: Bytes of data, Scales: Logarithmic along x-axis and y-axis
Run on Intel Core i5, under shared computer conditions

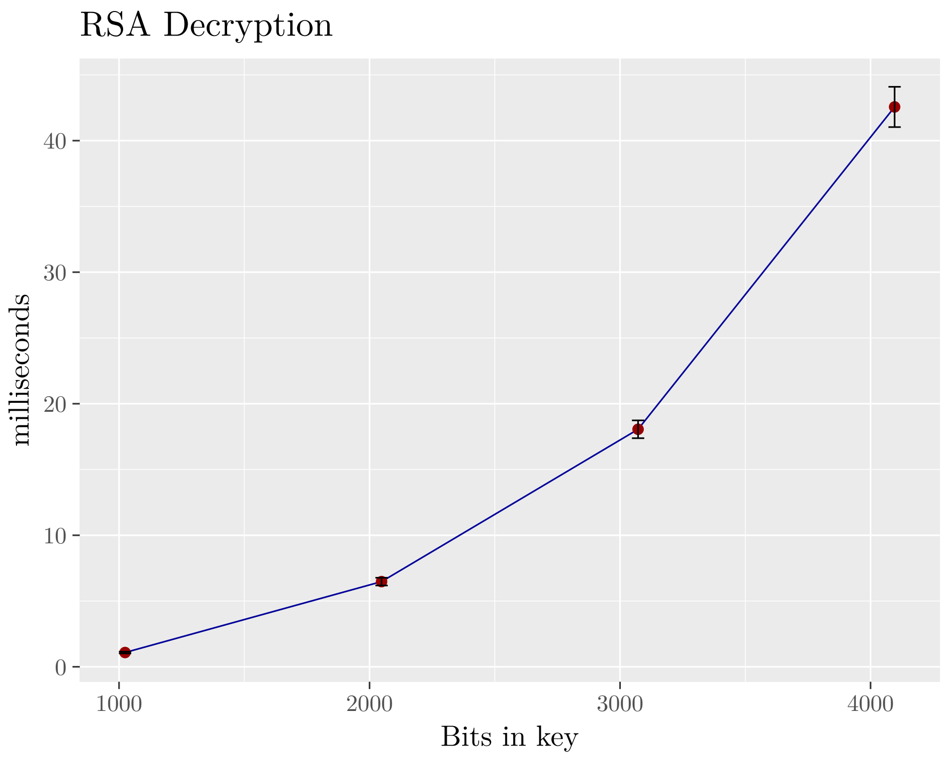


Figure A.2: RSA Decryption times, Variable: Bits in key.
Run on Intel Core i5, under shared computer conditions

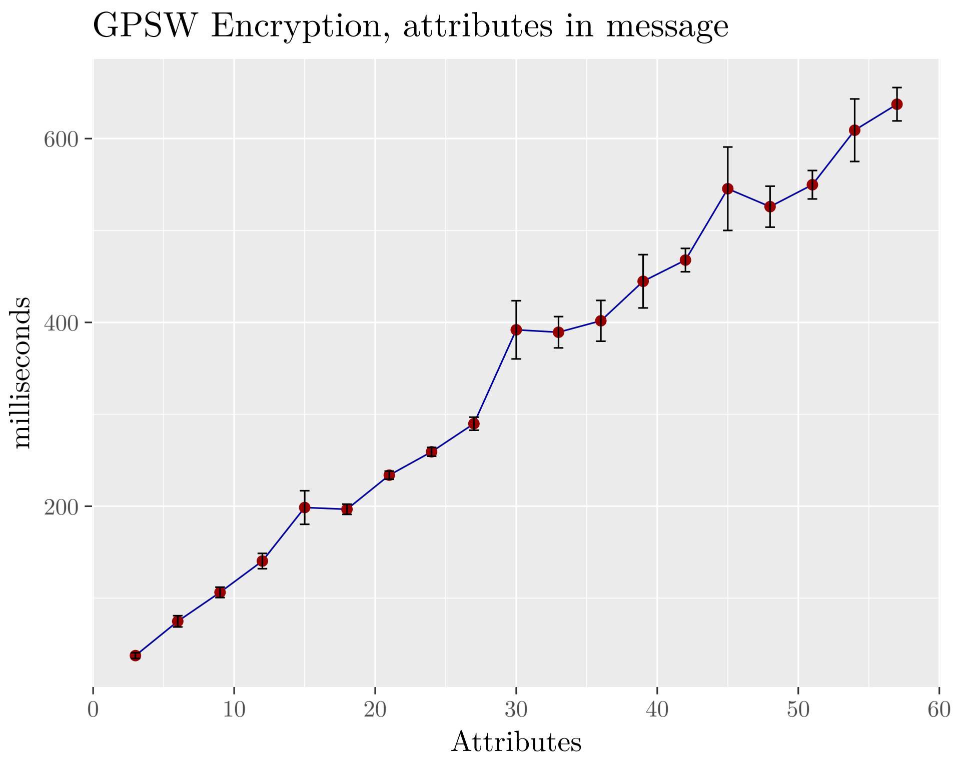


Figure A.3: GPSW Encryption times, Variable: Total number of attributes on message.
Run on Intel Core i5, under shared computer conditions

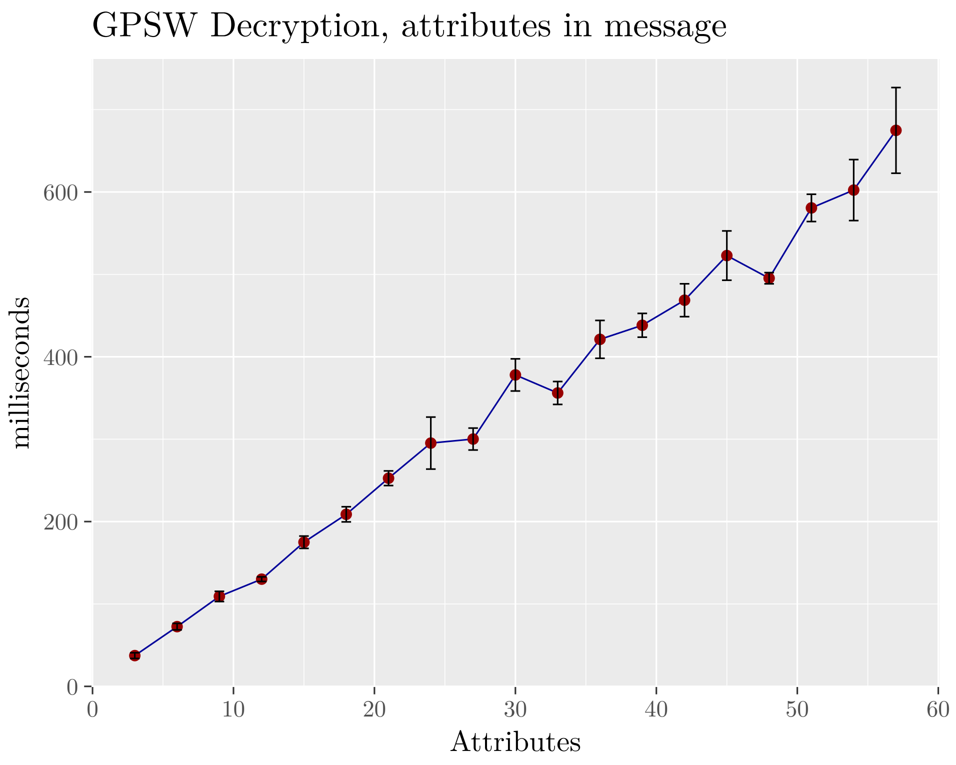


Figure A.4: GPSW Decryption times, Variable: Total number of attributes on message.
Run on Intel Core i5, under shared computer conditions

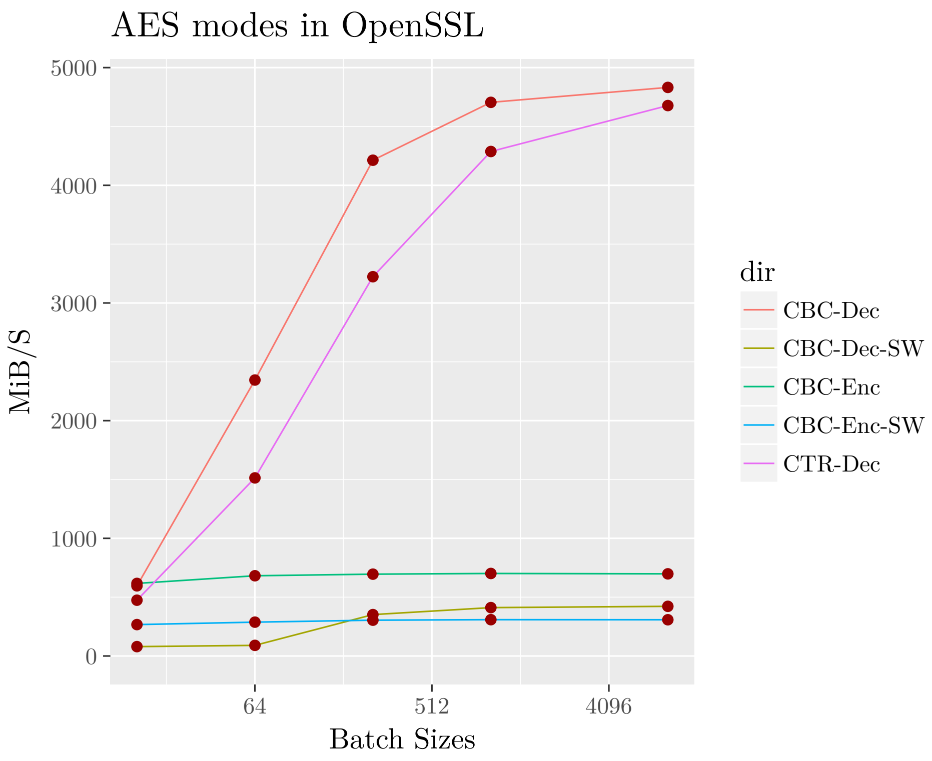


Figure A.5: Different operating modes of AES under OpenSSL. SW means OpenSSL was forced to use the software implementation.
Run on Intel Core i5, under shared computer conditions

Appendix B
General Data Protection Regulation (GDPR)

This chapter of the appendix quotes the parts of the GDPR that are deemed relevant for the work done [gdp].

> (22) Any processing of personal data in the context of the activities of an establishment of a controller or a processor in the Union should be carried out in accordance with this Regulation, regardless of whether the processing itself takes place within the Union. Establishment implies the effective and real exercise of activity through stable arrangements. The legal form of such arrangements, whether through a branch or a subsidiary with a legal personality, is not the determining factor in that respect.

> The application of pseudonymisation to personal data can reduce the risks to the data subjects concerned and help controllers and processors to meet their data-protection obligations. The explicit introduction of 'pseudonymisation' in this Regulation is not intended to preclude any other measures of data protection.

B.1 (32)

> [...] Personal data should be processed in a manner that ensures appropriate security and confidentiality of the personal data, including for preventing unauthorised access to or use of personal data and the equipment used for the processing.

B.2 (65)

A data subject should have the right to have personal data concerning him or her rectified and a ‘right to be forgotten’ where the retention of such data infringes this Regulation or Union or Member State law to which the controller is subject. In particular, a data subject should have the right to have his or her personal data erased and no longer processed where the personal data are no longer necessary in relation to the purposes for which they are collected or otherwise processed, where a data subject has withdrawn his or her consent or objects to the processing of personal data concerning him or her, or where the processing of his or her personal data does not otherwise comply with this Regulation. That right is relevant in particular where the data subject has given L 119/12 EN Official Journal of the European Union 4.5.2016
his or her consent as a child and is not fully aware of the risks involved by the processing, and later wants to remove such personal data, especially on the internet. The data subject should be able to exercise that right notwithstanding the fact that he or she is no longer a child. However, the further retention of the personal data should be lawful where it is necessary, for exercising the right of freedom of expression and information, for compliance with a legal obligation, for the performance of a task carried out in the public interest or in the exercise of official authority vested in the controller, on the grounds of public interest in the area of public health, for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes, or for the establishment, exercise or defence of legal claims.

B.3 (83)

In order to maintain security and to prevent processing in infringement of this Regulation, the controller or processor should evaluate the risks inherent in the processing and implement measures to mitigate those risks, such as encryption. Those measures should ensure an appropriate level of security, including confidentiality, taking into account the state of the art and the costs of implementation in relation to the risks and the nature of the personal data to be protected. In assessing data security risk, consideration should be given to the risks that are presented by personal data processing, such as accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to, personal data transmitted, stored or otherwise processed which may in particular lead to physical, material or non-material damage.

B.4 Article 4

(1) 'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;
(2) 'processing' means any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means, such as collection, recording, organisation, structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, restriction, erasure or destruction;
(3) 'restriction of processing' means the marking of stored personal data with the aim of limiting their processing in the future;
(4) 'profiling' means any form of automated processing of personal data consisting of the use of personal data to evaluate certain personal aspects relating to a natural person, in particular to analyse or predict aspects concerning that natural person's performance at work, economic situation, health, personal preferences, interests, reliability, behaviour, location or movements;
(5) 'pseudonymisation' means the processing of personal data in such a manner that the personal data can no longer be attributed to a specific data subject without the use of additional information, provided that such additional information is kept separately and is subject to technical and organisational measures to ensure that the personal data are not attributed to an identified or identifiable natural person; (6) 'filing system' means any structured set of personal data which are accessible according to specific criteria, whether centralised, decentralised or dispersed on a functional or geographical basis;

..

(8) 'processor' means a natural or legal person, public authority, agency or other body which processes personal data on behalf of the controller;

..

(12) 'personal data breach' means a breach of security leading to the accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to, personal data transmitted, stored or otherwise processed;

..

B.5 Article 23: 2.

1. Taking into account the state of the art, the costs of implementation and the nature, scope, context and purposes of processing as well as the risk of varying likelihood and severity for the rights and freedoms of natural persons, the controller and the processor shall implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk, including inter alia as appropriate:
(a) the pseudonymisation and encryption of personal data;
4.5.2016 EN Official Journal of the European Union L 119/51
(b) the ability to ensure the ongoing confidentiality, integrity, availability and resilience of processing systems and services;
(c) the ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident;
(d) a process for regularly testing, assessing and evaluating the effectiveness of technical and organisational measures for ensuring the security of the processing.

In particular, any legislative measure referred to in paragraph 1 shall contain specific provisions at least, where relevant, as to:
(a) the purposes of the processing or categories of processing;
(b) the categories of personal data;
(c) the scope of the restrictions introduced;
(d) the safeguards to prevent abuse or unlawful access or transfer;
(e) the specification of the controller or categories of controllers;
(f) the storage periods and the applicable safeguards taking into account the nature, scope and purposes of the processing or categories of processing;
(g) the risks to the rights and freedoms of data subjects; and
(h) the right of data subjects to be informed about the restriction, unless that may be prejudicial to the purpose of the restriction.

B.6 Article 32

1. Taking into account the state of the art, the costs of implementation and the nature, scope, context and purposes of processing as well as

the risk of varying likelihood and severity for the rights and freedoms of natural persons, the controller and the processor shall implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk, including inter alia as appropriate:
(a) the pseudonymisation and encryption of personal data;
4.5.2016 EN Official Journal of the European Union L 119/51
(b) the ability to ensure the ongoing confidentiality, integrity, availability and resilience of processing systems and services;
(c) the ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident;
(d) a process for regularly testing, assessing and evaluating the effectiveness of technical and organisational measures for ensuring the security of the processing.

B.7 Article 33

1. Taking into account the state of the art, the costs of implementation and the nature, scope, context and purposes of processing as well as the risk of varying likelihood and severity for the rights and freedoms of natural persons, the controller and the processor shall implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk, including inter alia as appropriate:
(a) the pseudonymisation and encryption of personal data;
4.5.2016 EN Official Journal of the European Union L 119/51
(b) the ability to ensure the ongoing confidentiality, integrity, availability and resilience of processing systems and services;
(c) the ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident;
(d) a process for regularly testing, assessing and evaluating the effectiveness of technical and organisational measures for ensuring the security of the processing.
2. In assessing the appropriate level of security account shall be taken in particular of the risks that are presented by processing, in particular from accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to personal data transmitted, stored or otherwise processed.
3. Adherence to an approved code of conduct as referred to in Article 40 or an approved certification mechanism as referred to in Article 42 may be used as an element by which to demonstrate compliance with the requirements set out in paragraph 1 of this Article.

4. The controller and processor shall take steps to ensure that any natural person acting under the authority of the controller or the processor who has access to personal data does not process them except on instructions from the controller, unless he or she is required to do so by Union or Member State law.

Appendix C
Code

this chapter contains the relevant parts of the code used during the project.

C.1 Benchmark for GPSW encryption

```
@Override
public boolean run() {
    return instance.encrypt(iterations, attribs_use);
}

@Override
public boolean setup() {
    instance = new KpAbe();
    try {
        attribs_all = KpAbe.generate_attribs(attribs);
        instance.setup(attribs_all);
        attribs_use = KpAbe.get_random_attributes(attrib_used,
            attribs_all);
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
        return false;
    }
    return true;

}

@Override
public boolean cleanup() {
    instance = null;
    attribs_all = null;
    attribs_use = null;
    return true;

}
```

C.2 Benchmark for GPSW decryption

```
@Override
public boolean run() {
    return instance.encrypt(iterations, attribs_use);
}

@Override
public boolean setup() {
    instance = new KpAbe();
    boolean works = true;
    try {
        attribs_all = KpAbe.generate_attribs(attribs);
        instance.setup(attribs_all);
        attribs_use = KpAbe.get_random_attributes(attrib_used,
            attribs_all);
        works = works && instance.do_iteration_enc(attribs_use);
        attribs_policy = KpAbe.get_random_attributes(attrib_key,
            attribs_use);
        instance.setPolicy(KpAbe.make_policystring(attribs_policy))
            ;
        works = works && instance.do_key_generation();
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return works;
}

@Override
public boolean cleanup() {
    instance = null;
    attribs_all = null;
    attribs_use = null;
    return true;

}
```

C.3 Benchmark for RSA encryption

```
@Override
public boolean run() {
    for(int i=0; i<iterations; i++){
        if(!iteration()){
            return false;
        }
    }
    return true;
}

private boolean iteration() {
    try {
        encrypted = engine.doFinal(data);
    } catch (IllegalBlockSizeException | BadPaddingException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

@Override
public boolean setup() {
    data = new byte[32];
    Random random = new Random();
    random.nextBytes(data);

    try {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(keysize);
        keys = kpg.generateKeyPair();
        engine = Cipher.getInstance("RSA");
        engine.init(Cipher.ENCRYPT_MODE, keys.getPublic());
    } catch (NoSuchAlgorithmException | InvalidKeyException |
        NoSuchPaddingException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

@Override
public boolean cleanup() {
    this.keys = null;
    this.engine = null;
    this.data = null;
    this.encrypted = null;
    return true;
}
```

C.4 Benchmark for RSA decryption

```
@Override
public boolean run() {
    for(int i=0; i<iterations; i++){
        if(!iteration()){
            return false;
        }
    }
    return true;
}

private boolean iteration() {
    try {
        outdata = engine.doFinal(encrypted);
    } catch (IllegalBlockSizeException | BadPaddingException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

@Override
public boolean setup() {
    data = new byte[32];
    Random random = new Random();
    random.nextBytes(data);
    try {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(keysize);
        keys = kpg.generateKeyPair();
        engine = Cipher.getInstance("RSA");
        engine.init(Cipher.ENCRYPT_MODE, keys.getPublic());
        encrypted = engine.doFinal(data);
        engine.init(Cipher.DECRYPT_MODE, keys.getPrivate());
    } catch (NoSuchAlgorithmException | InvalidKeyException |
        NoSuchPaddingException | BadPaddingException |
        IllegalBlockSizeException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

@Override
public boolean cleanup() {
    this.keys = null;
    this.engine = null;
    this.data = null;
    this.encrypted = null;
    return true;
}
```

C.5 Benchmark for AES encryption

```
boolean iteration() {
    try {
        engine.init(Cipher.ENCRYPT_MODE, key, ivp);
        outdata = engine.doFinal(data);
    } catch (InvalidKeyException |
        InvalidAlgorithmParameterException | BadPaddingException |
        IllegalBlockSizeException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

@Override
public boolean run() {
    for(int i=0; i<iterations; i++){
        if(!iteration()){
            return false;
        }
    }
    return true;
}

@Override
public boolean setup() {
    data = new byte[length];
    iv = new byte[16];
    Random random = new Random();
    random.nextBytes(iv);
    random.nextBytes(data);

    try {
        KeyGenerator KeyGen=KeyGenerator.getInstance("AES");
        KeyGen.init(128);
        key=KeyGen.generateKey();
        engine = Cipher.getInstance("AES/CBC/PKCS5Padding");
        ivp = new IvParameterSpec(iv);
    } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

C.6 Benchmark for AES decryption

```
private boolean iteration() {
    try {
        engine.init(Cipher.DECRYPT_MODE, key, ivp);
        outdata = engine.doFinal(ciphertext);
    } catch (InvalidKeyException |
        InvalidAlgorithmParameterException | BadPaddingException |
        IllegalBlockSizeException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

@Override
public boolean run() {
    for(int i=0; i<iterations; i++){
        if(!iteration()){
            return false;
        }
    }
    return true;
}

@Override
public boolean setup() {
    data = new byte[length];
    iv = new byte[16];
    Random random = new Random();
    random.nextBytes(iv);
    random.nextBytes(data);

    try {
        KeyGenerator KeyGen=KeyGenerator.getInstance("AES");
        KeyGen.init(128);
        key=KeyGen.generateKey();
        engine = Cipher.getInstance("AES/CBC/PKCS5Padding");
        ivp = new IvParameterSpec(iv);
        engine.init(Cipher.ENCRYPT_MODE, key, ivp);
        ciphertext = engine.doFinal(data);
    } catch (NoSuchAlgorithmException | NoSuchPaddingException |
        BadPaddingException | InvalidKeyException |
            InvalidAlgorithmParameterException |
                IllegalBlockSizeException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

C.7 GPSW Wrapper

Liang's original implementation required some wrangling to work properly this is the re-worked outer wrapper.

```
public class KpAbe {
    private gpswabePub pub;
    private gpswabeMsk msk;
    private gpswabePrv prv;
    private gpswabeCphKey cphKey;
    private gpswabeCph cph;
    private byte[] simulated_key = {
            (byte) 165, (byte) 165, (byte) 241, (byte) 218, (byte) 155,
                 (byte) 193, (byte) 4, (byte) 237,
            (byte) 33, (byte) 163, (byte) 171, (byte) 245, (byte) 92, (
                byte) 52, (byte) 159, (byte) 191,
            (byte) 87, (byte) 212, (byte) 225, (byte) 4, (byte) 80, (
                byte) 80, (byte) 244, (byte) 82,
            (byte) 89, (byte) 82, (byte) 19, (byte) 146, (byte) 192, (
                byte) 247, (byte) 115, (byte) 52
    };

    private byte[] ciphertext;
    private String policy;
    private String[] attrs_univ;

    public void setup(String[] attrs_univ) throws IOException,
        ClassNotFoundException {
        pub = new gpswabePub();
        msk = new gpswabeMsk();
        gpswabe.setup(pub, msk, attrs_univ);
        this.attrs_univ = attrs_univ;
    }

    public void keygen(String policy) throws Exception {
        prv = gpswabe.keygen(pub, msk, policy);
    }

    public byte[] enc(String[] attrs, byte[] cleartext) throws
        Exception {
        byte[] aesBuf;
        Element m;

        cphKey = gpswabe.enc(pub, attrs);
        m=cphKey.key;
        cph=cphKey.cph;
        //System.err.println("m = "+m.toString());

        if (cph == null) {
            System.out.println("Error␣happed␣in␣enc");
            System.exit(0);
        }
        aesBuf = AESCoder.encrypt(m.toBytes(), cleartext);
        // PrintArr("element: ", m.toBytes());
        return aesBuf;
    }

    public byte[] dec(byte[] ciphertext) throws Exception {
        byte[] plt;

        Element m=gpswabe.dec(pub, prv, cph);
        if (m!=null) {
            plt = AESCoder.decrypt(m.toBytes(), ciphertext);
            return plt;
        } else {
            return null;
        }
    }

    public boolean do_iteration_enc(String[] use_attrib){
        try {
            this.ciphertext = this.enc(use_attrib, simulated_key);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;

    }

    public boolean encrypt(int iterations, String[] use_attrib){
        assert iterations > 0;
        for (int i = 0; i < iterations; i++) {
            if( !this.do_iteration_enc(use_attrib)){
                return false;
            }
        }
        return true;
    }

    public boolean do_key_generation(){
        try {
            this.keygen(this.policy);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }

    public boolean do_iteration_dec(){
        try {
            byte[] out = this.dec(this.ciphertext);
            if (Arrays.equals(out, this.simulated_key)){
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean decrypt(int iterations){
        assert iterations > 0;
        for (int i = 0; i < iterations; i++) {
            if( !this.do_iteration_dec()){
                return false;
            }
        }
        return true;
    }

    public static String[] generate_attribs(int num){
        String[] a = new String[num];
        for(int i =0; i<num; i++){
            a[i] = String.format("%04X", i);
        }
        return a;
    }

    public static String make_policystring(String[] attribs){
        String[] inp = attribs;
        String[] iter;
        int l;
        while(inp.length>1) {
            l = inp.length / 2 + inp.length % 2;
            iter = new String[l];
            for (int i = 0; i < l; i++) {
                if (inp.length-(2*i) > 1) {
                    iter[i] = inp[2 * i] + "␣" + inp[(2 * i) + 1] + "␣2
                        of2";
                } else {
                    iter[i] = inp[2 * i];
                }
            }
            inp = iter;
        }
        return inp[0];
    }

    public static String[] get_random_attributes(int n, String[]
        candidates){
        if(candidates == null || candidates.length<n){
            return null;
        }
        Set<String> set = new HashSet<>();
        int r;
        while(set.size() < n) {
            r = (int)(Math.random() * n);
            set.add(candidates[r]); //if its the same, it won't be
                added to the set and the size won't increase
        }
        return set.toArray(new String[set.size()]);
    }

    public String[] get_random_attributes(int n){
        return get_random_attributes(n, attrs_univ);
    }

    public void setPolicy(String policy){
        this.policy = policy;
    }

    public static void main(String[] args){
        KpAbe kp = new KpAbe();

        String[] attribs = generate_attribs(10);
        try {
            kp.setup(attribs);
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(200);
        }
        String[] use_attrib = kp.get_random_attributes(5);

        long startTime = System.nanoTime();
        kp.encrypt(1000, use_attrib);
        long endTime = System.nanoTime();
        long duration = (endTime - startTime);
        System.out.println(duration / 1000000);

        kp.policy = make_policystring(get_random_attributes(2,
            use_attrib));
        kp.do_key_generation();

        startTime = System.nanoTime();
        kp.decrypt(1000);
        endTime = System.nanoTime();
        duration = (endTime - startTime);
        System.out.println(duration / 1000000);

        kp.policy = make_policystring(get_random_attributes(4,
            use_attrib));
        kp.do_key_generation();

        startTime = System.nanoTime();
        kp.decrypt(1000);
        endTime = System.nanoTime();
        duration = (endTime - startTime);
        System.out.println(duration / 1000000);


    }
```

C.8 Main test loop

```
    void runtests(){
        Collections.shuffle(entries);
        int iterations = entries.size()*2;
        BenchmarkEntry e;

        long startTime;
        long endTime;
        long duration;
        boolean result;

        while(entries.size() > 0 && iterations > 0){
            e = entries.remove(0);

            if(!e.setup()){
                System.err.println("Error!");
                entries.add(e);
            } else {
                startTime = System.nanoTime();
                result = e.run();
                endTime = System.nanoTime();
                duration = (endTime - startTime) / 1000000;
                if (result) {
                    System.out.print(e.name());
                    System.out.print("\t");
                    System.out.print(e.var());
                    System.out.print("\t");
                    System.out.println(((float) duration) / 1000.0);
                } else {
                    System.err.println("Error!");
                    entries.add(e);
                }
            }
            e.cleanup();
            iterations--;
        }
    }
```

C.9 Calculator for logic element count

```python
from math import log2, ceil


"""
Squaring in-bits vs out bits
"""
for x in range(1, 4096):
    o = float(ceil(log2((2**x-1)**2)))/float(x)
    if o != 2.0:
        print(x, o)

"""
Multiplying uneven numbers, in bits vs out bits
"""
for a in range(2, 100):
    for b in range(a, 100):
        o = ceil(log2((2**a-1)*(2**b-1)))
        if o != a+b:
            print(a, b, o)

def mult(m, n):
    """
    The number of Logic elements in aa m*n multiplier
    """
    return m*n + (m*n)*1j

def exp_b(m, n):
    """
    One exponensiation bit m is bits carried, n is base (bits)
    """
    return 3*m + 4*n + 2 + mult(m, n) + mult(m+n, m+n)

def exp_l(m, n):
    """
    The last exponensiation bit m is bits carried,
    n is base (bits)
    """
    return 3*n + m + 2 + mult(m, n)

def w(n, i):
    """
    In bits form ith exponent bit with base n (bits)
    """
    if i == 1:
        return n
    if i == 2:
        return 3*n
    else:
        return 2*w(n, i-1) + n

def exp_t(n, a):
```

```
"""
cost of n^a bit exponentiator
"""
o = 0
for i in range(1, a):
    o += exp_b(w(n, i), n)
o += exp_l(w(n, a), n)
return o
```