DEGREE PROJECT IN ELECTRICAL ENGINEERING, SECOND LEVEL
STOCKHOLM, SWEDEN 2017

Video Integrity through Blockchain Technology

ADAM HEMLIN BILLSTRÖM and FABIAN HUSS

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Video Integrity through Blockchain Technology

Adam Hemlin Billström and Fabian Huss

2017-08-02

Master's Thesis

Examiner
Gerald Q. Maguire Jr.

Supervisor
Anders Västberg

Industrial adviser
Ben Wilmot

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

The increasing capabilities of today's smartphones enables users to live stream video directly from their mobile device. One increasing concern regarding videos found online is their authenticity and integrity. From a consumer standpoint, it is very hard to distinguish and discern whether or not a video found on online can be trusted, if it was the original version, or if has been taken out of context. This thesis will investigate a method which tries to apply video integrity to live streamed media.

The main purpose of this thesis was to design and evaluate a proof of concept prototype which will apply data integrity while simultaneously recording videos through an Android device. Additionally, the prototype has an online verification platform which verifies the integrity of the recorded video. Blockchain is a technology with the inherent ability to store data in a chronological chained link of events: establishing an irrefutable database. Using cryptographic hashes together with blockchain: an Android device can generate cryptographic hashes of the data content from a video recording, and consequently transmit these hashes to a blockchain. The same video is deconstructed in the web client creating hashes that can subsequently be compared with the ones found in the blockchain.

A resulting prototype system provides some of the desired functions. However, the prototype is limited in that it does not have the ability to sign the hashes produced. It has also been limited in that it does not employ HTTPS for communication, and the verification process needs to be optimized to make it usable for real applications.

Keywords— Hashing, live stream, video integrity, video verification, blockchain

Sammanfattning

Den ökande funktionaliteten hos dagens smarta mobiltelefoner ger användare möjligheten att direktsända video. Det förekommer en ökande oro när det kommer till videors äkthet och huruvida en video är original eller inte. Ur en konsumentsynpunkt är det nämligen väldigt svårt att bedöma huruvida det går att lita på videon, om det är originalvideon eller om det bara är så att videon är tagen ur sitt sammanhang. Detta examensarbete på Master-nivå kommer att undersöka en metod för att verifiera att direktsänd media är oförändrad.

Huvudsyftet med arbetet var att ta fram och utvärdera en prototyp som kan säkerställa oföränderlighet inom direktsänd video samtidigt som videon spelas in på mobilenheten. Prototypen har dessutom en webbaserad verifieringsplattform som kan verifiera och säkerställa huruvida videon (media) är oförändrad. Blockkedjeteknologin har den inbyggda egenskapen att kunna spara data i en kronologisk sammanlänkad ordning av händelser. Den skapar databas som inte kan ifrågasättas. Genom att använda kryptografisk hashning tillsammans med blockkedjetekniken kan en Android mobilenhet skapa kryptografiska hashar av videodata under tiden som videon spelas in och simultant skicka dessa hashar till en blockkedja. Samma video tas sedan isär i prototypens verifieringsfunktion. Verifieringsfunktionen skapar sedan hashar på samma sätt som i mobilenheten för att kunna jämföra dessa hashar mot de hashar som kan hämtas från blockkedjan.

Prototypen är fungerande men saknar viss eftersträvad funktionalitet. Prototypen är begränsad på det sätt att mobilenheten inte kan signera de hashar som genereras. Den saknar även möjligheten att kommunicera över HTTPS protokollet samt att processen för att verifiera videomaterial är alldeles för långsam för att kunna användas i en verklig produkt.

Nyckelord— direksändning, hash, videointegritet, videoverifiering, blockkedja

Acknowledgement

We would like to extend our greatest gratitude towards Gerald Q. Maguire Jr. for being our examiner in this Master's Thesis.

We would also like to extend our gratitude towards Ben Wilmot for being our industry supervisor at Ericsson. We would also like to thank Timur Alimbayev, Johan Kristiansson, Mads Becker, and Mala Chakraborti at Ericsson whom all have been very helpful.

Stockholm, August, 2017
Adam Hemlin Billström, Fabian Huss

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programmable Interface
app	Application
AVC	Advanced Video Coding
B-Frame	Bidirectional-Frame
BFT	Byzantine Fault Tolerance
BTC	Bitcoin (currency)
CA	Certificate Authority
CSS	Cascading Style Sheet
DoS	Denial of Service
ETH	Ether
EVM	Ethereum Vitrual Machine
GOP	Group of Pictures
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I-Frame	Intra-Frame
ID	Identity
IDE	Interactive Development Environment
MD5	Message Digest 5
MPEG	Moving Picture Experts Group
OS	Operating System
P-Frame	Predicted-Frame
PoS	Proof of Stake
PoW	Proof of Work
REST	Representational State Transfer
SAC	Strict Avalanche Criterion
SDK	Software Development Kit
SHA-1	Secure Hash Algorithm 1
SHA-2	Secure Hash Algorithm 2
SSL	Secure Socket Layer
TCP	Transfer Control Protocol
TSA	Time Stamping Authority
unit8	8-bit Unsigned Integers
XML	Extensible Markup Language

Chapter 1

Introduction

We live in an era where mobile phones become more and more common and mobile internet connections are getting faster and more reliable. The amount of streamed media content around the world is at a record high, and will only increase[2]. Media content originating on smartphones spreads rapidly between users and with limited restrictions via the social media platsforms[3]. Today video media content is being viewed by millions via social media platforms and sometimes even used by traditional broadcast media companies despite the fact that there is little or no source verification. This makes it increasingly important to be able to verify online video content, specifically: the integrity of the video file and the time it was taken (and possibly even by whom). Additionally, there is a question of has the media been manipulated or cut? The answer to these questions may be very important and may completely alter the viewers' reliance on the video.

In the context of unaccredited news, some videos have been manipulated or the video taken out of its context. This Master's thesis investigates methods to verify video content based on a specific criteria: integrity (i.e., non-modification) of the recording. This would not only flag as unverified a lot of fake news videos but might also be used to give increased credibility to videos which can be verified.

This chapter gives a brief introduction to this Master's thesis. Section 1.1 gives a general background to the reader. Section 1.2 further discusses the problems that this thesis will investigate. Section 1.3 gives a brief overview of our ideas regarding how the problem will be solved and why. Section 1.4 succinctly breaks down and quantizes the project into measurable goals. Section 1.5 presents a general framework and approach to the complete project. Section 1.6 explains the initial idea regarding what the final proof of concept prototype will look like. In Section 1.7 the delimitations are discussed and why certain areas have been excluded. Finally, Section 1.8 outlines the entirety of this thesis.

1.1 Background

Mobile phone cameras have a plethora of applications and uses in the connected society. The ability to share cherished moments and experiences are increasingly important and highly appreciated by the citizen of the modern world. Videos are found

on almost every social media platform and users are streaming more and more content live from their phone camera or other internet connected device(s). The number of videos which can be found online is immense and it is often very difficult to discern the authenticity and the integrity of the media content, to distinguish between original content and falsified and/or copied content. People and news agencies are using media content to drive their own agenda and these videos may be manipulated, cut, or taken out of their original context without the viewer's knowledge. Therefore, it is of great interest to develop a tool which makes it easy to, in a fast manner, distinguish between the modified content and the original content. This tool would enable viewers to themselves be able to discern and verify which videos to trust and which videos to discard regarding their integrity.

The topic of this Master's thesis is, as previously mentioned, highly relevant and widely discussed but usually not in the context of blockchains. Blockchain technology has the potential to provide data security and enable validation of the integrity of the data.

1.2 Problem

More and more mobile phones are equipped with cameras, internet connections, and various sensors. The increasing capabilities of today's smartphones enable the end user to produce and distribute media content more easily than before. One issue that has arisen from this is the inability to be able to distinguish between a modified video and the original video. Additionally, videos are taken out of context or claimed to be from a place and/or time different from what might actually be the truth. It is getting increasingly difficult for the viewer to know what to believe, who to believe, and tools to help counter these issues are scarce and limited.

A service as proposed by this Master's thesis will need to be trusted by its users. It is therefore of paramount importance to be able to ensure the integrity of the service and its software. The user needs to be able to trust the software they are downloading, installing, and using on their smartphone.

1.3 Purpose

The purpose of this Master's thesis is to set up and evaluate the usage of a blockchain in order to verify and store signature hashes from live-streamed video content from an Android device. The creation of the signature hashes should be done as near to real-time as possible, leaving as little time as possible for data manipulation by anyone and enabling the establishment of the precedence of this signed hash (i.e., it can be shown that it was signed at a particular period in time (based upon the entries before and after it in the blockchain), therefore the hash had to be calculated before this, and hence the media over which the hash is computed has to be even earlier than this). Storing the signatures in the blockchain will enable signature verification at any later time. In the thesis, the "Good cop/Bad cop" [4] the thesis project addes the signed hashes to the stream and stores the final signed hash with one or more escrow agents - (noting that Shamir secret sharing can be used - so that n of m escrow agents are sufficient to recover the escrowed value). In contrast, in this thesis project, the signature hash is stored in a blockchain. Verification methods for these hashes are implemented

and accessible to users via a web-based interface. This proof of concept prototype showcases the potential of combining blockchains with hashes over chunks of media content and meets the requirements of the evolving world of streaming media content.

1.4 Goals of this Master's Thesis

The measurable goals for this Master's Thesis can be enumerate as three goals through which we aim to develop and evaluate a proof of concept prototype. The prototype when complete should be able (in no particular order) to:

1. Determine whether a video has been taken within a specified time interval of 1 day.
2. Determine which parts of the video whose integrity can/cannot be verified.
3. The process of verifying the video should require less than 60 seconds for a 300 second video recording*.

1.5 Research Methodology

The qualitative hypothetico-deductive method is used for this Master's thesis. We have selected a deductive method since it uses previous theory and the main goal of the thesis is to prove a theory; specifically, to prove that blockchain technology is a suitable tool to ensure data integrity for video streaming. This is the hypothesis we wish to either confirm or dismiss.

We have chosen a qualitative method as this project is primary exploratory research within the combined areas of data integrity and blockchains. We seek to provide insights into the problem being investigated. The data collection method in this Master's thesis will be unstructured and dependent upon the actual progress toward the project's goals.

The data collected during the investigation will be highly reliable since it will be collected by software without any human interaction. The data is replicable as long as the same hardware and software are used. This is due to the fact that different hardware has different characteristics (such as processing power and read/write memory speed).

1.6 The setup

The verification service (web server) runs on a Macbook Pro (see Table 1.1) with El Capitan (10.11.6) version of the MacOS operating system. The mobile client software is installed on a Oneplus One (see Table 1.1), running the Android Operating System (OS) version 6.0.1. The system can be seen in Figure 1.1. The Android application is built in Android studio 2.3† using Java and Extensible Markup Language (XML). The phone sends the hashes over the Hypertext Transfer Protocol (HTTP) protocol to the server which is running on the Macbook Pro seen in Figure 1.2. The server computes the blockchain. The verification service seen in Figure 1.3 takes videos as

*The 300 second recording translates to 225Mb of data given the parameters set in Table 4.1

†https://android-developers.googleblog.com/2017/03/android-studio-2-3.html

an input and verifies the integrity of the video by hashing the videos and comparing these hashes to the signed hashes found in the blockchain. The web app is built in JavaScript/HTML/Cascading Style Sheet (CSS) and is further explained in Section 2.5. The web browser used for testing is Google Chrome version 58.0.3029.110 (64-bit).

Table 1.1: Device details

	Macbook	**Android device**
Model	Macbook Pro Retina 13" Late 2013	Oneplus One
OS	OSX El Capitan (10.11.6)	Android OS version 6.0.1
Processor	2.4GHz core i5	Snapdragon 801 Quad-core 2.5 GHz
RAM	8GB 1600MHz DDR3	3GB 1866MHz LPDDR3
Graphics	Iris 5100 Free	Adreno330 GPU
Camera sensor	N/A	Sony Exmor RS IMX214 (4K@30fps)

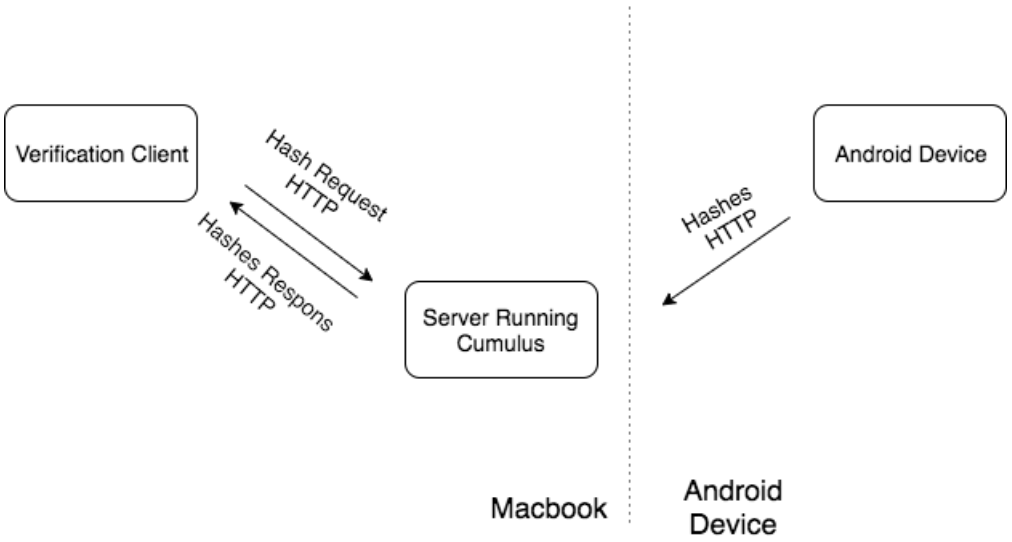

Figure 1.1: The proposed system model.

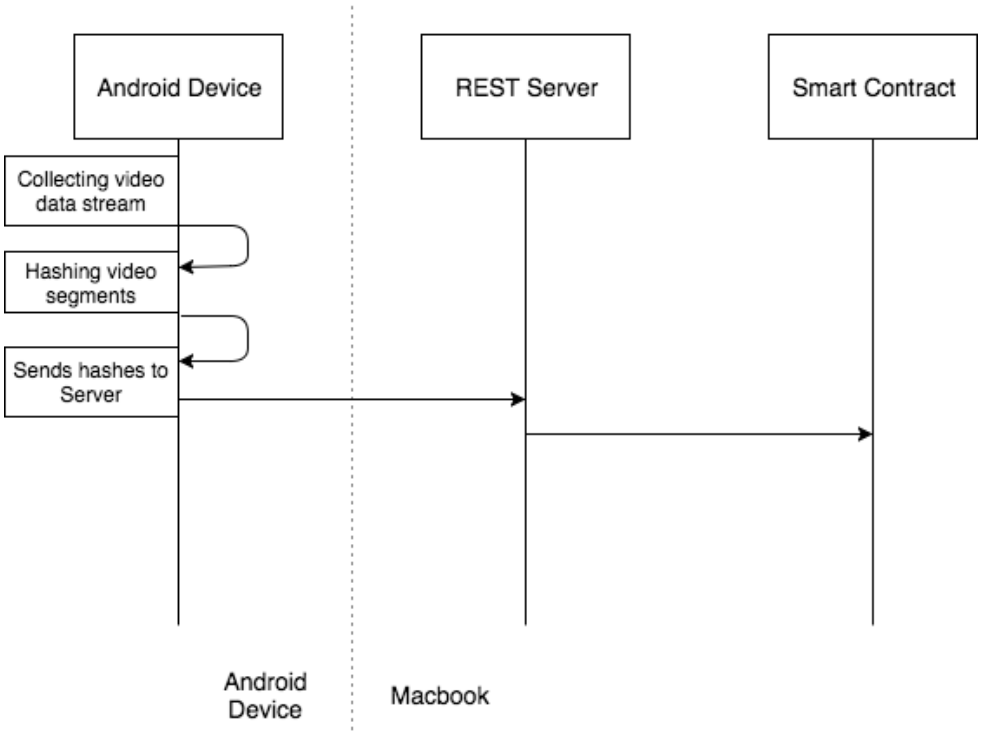

Figure 1.2: A process diagram for the process of creating and adding hashes to the smart contract.

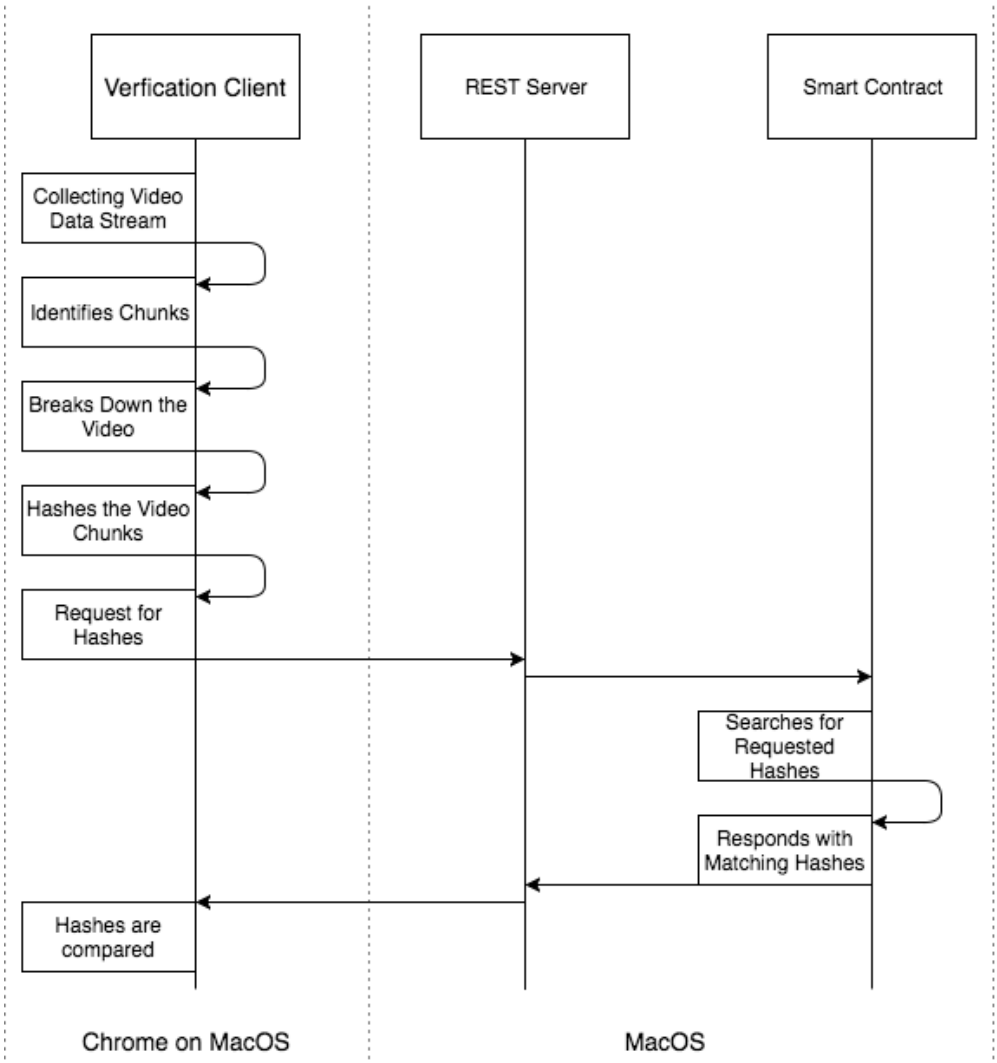

Figure 1.3: A process diagram for the process of creating, requesting and verifying the hashes.

1.7 Delimitations

This Master's thesis will investigate and develop a proof of concept prototype showcasing the merger of an app running on an Android device with a backend security system using blockchain verification through smart contracts. However, the thesis will not examine what type of manipulation or tampering has been done to the recorded data, but rather will simply verify if the hash of the video content is the same as that of the signed hash as recorded in the blockchain. There already exists a plethora of methods for detecting data tampering and detecting what has actually been manipulated (see Section 2.6.1). The planned proof of concept prototype will simply determine whether or not the hash of the video subjected to verification matches the original video's hash saved by the Android device in the blockchain. Additionally, the use of

a blockchain is limited to setting up a working blockchain which can handle and store unique hashes from connected Android devices.

Other delimitations are the assumption of a perfect connection between the Android device and the server, i.e., without any packet loss. This loss free communication is required for both the upload of the video to be compared and the upload of hashes (with the meta data) to the blockchain. The issue of handling packet-loss in these scenarios are outside of the scope of this Master's thesis.

1.8 Structure of the Thesis

This Master's thesis is organized into six chapters. Chapter 2 details all the discrete building blocks that the thesis will be constructed from. Chapter 2 also provides all of the relevant background that the reader will need to understand the rest of this thesis. Chapter 3 details proof of concept prototype and the fundamental building blocks of the project in which the prototype will be implemented, also the limitation of each major part of the prototype. Chapter 4 details the research process, building the structure of the research process, solution framework evaluation, and describes the tests used to evaluate the prototype. Chapter 5 introduces the test results evaluating the final product and discussing the results. Finally, Chapter 6 summarizes the conclusions of this project and proposes what should be done as future work.

Chapter 2

Background

Proving the integrity of data has become more and more important. Security breaches that have leaked passwords and user emails are commonplace and regularly appear in the news. When a system is hacked, the first issue that usually comes to mind is the compromise of confidentiality, specifically what confidential information is now in the hands of the perpetrator. Another aspect, which is getting increasingly important, is to know the integrity of data, has anything been changed in order to address the higher level question: "Can we trust that this data is the actual data?". The need to verify the integrity of the data varies between different organizations. For example, in health care, incorrect information could be life threatening, while in industries such as finance in authentic data could be associated with great cost. Social media platforms are presenting live videos on their platform with only a very low (if any) level of verification, often next to paid content and advertising. Companies paying for distribution of content and advertising have an inherent interest to know that their content is being displayed reliably and in the case of advertisements that the content displayed next to their content is reliable, i.e., that it has not been compromised. This Master's thesis will investigate the possibility of verifying the integrity of video content (such as surveillance videos) using blockchains. The resulting system should be capable of being implemented on various platforms, such as social media platforms.

This chapter introduces the reader to all the relevant background knowledge needed to grasp all of this Master's thesis. Section 2.1 details what a video stream is composed of and how it can be decomposed. In Section 2.2 describes the intricacies of the main underlying concepts that comprises a blockchain. Section 2.3 talks about the security of our system with regards to device to device communication. Section 2.4 details the Android framework and what components & Application Programming Interfaces (APIs) will be used to create the prototype mobile client. Section 2.5 explains the front end development done for the web server, focusing mainly on the necessary JavaScript development. Section 2.6 discusses related projects that have been done both in academia and in the industry. Lastly, Section 2.7 summarizes the background information and lays a theoretical foundation for this Master's thesis.

2.1 Video Stream

A video stream can be either a combination of audio & video, only video, or video and other content, depending on the format. A camera outputs uncompressed video at a constant data rate. The data rate for uncompressed video is fixed and given as the frame rate multiplied by the video resolution, and the color depth. Compression of video tries to minimize the data rate in order to transmit the data within a given bandwidth-constrained channel or to reduce the time it takes to transfer the compressed data. Compression can be done using two techniques: lossless or lossy. As the words imply, lossless compression compresses the data *without* the loss of information. In contrast, lossy compression encodes the data by removal of some information in order to further minimize or bound the resulting data rate or required storage space.

Uncompressed data from the camera is encoded and compressed using video encoding formats (such as AVC [5], HEVC [6], VP8 [7], VP9 [8], etc.), often in combination with audio encoding formats (MP3 [9], AAC [10], Opus [11], etc.). Container formats (such as AVI [12], QTFF [13], Matroska [14], mp4 [5], etc.) combines the audio and video together with additional related files and optional features (header files, chapters, subtitles, etc.), and creates one single file.

Feature extraction from a video can be done frame by frame, using short video sequences, or entire video files. The thesis will investigate what the signed hashed should be done on, i.e., either frame by frame or on video segments. Additionally, the time to compute the hash needs to be taken into consideration when generating the signed hashes. Finally, the number of hashes being created should also be considered, this is discussed in Section 6.1. As we will work with compressed video we need to ensure that the code that does the compression has not been modified, hence in Section 2.2.4 we will describe how we also compute hashes over the code in the device. The computing power of a smartphone needs to be considered with respect to a real-time hashing algorithm when computing hashes over video frames or video segments. This is tested in test “Hashing” in Section 5.2

2.1.1 Advanced Video Coding

Advanced Video Coding (AVC) or simply H.264 is a standard video encoding format from the Moving Picture Experts Group (MPEG). It is one of the most common video encoding formats for recording, compression, and streaming. H.264 encoded videos are constructed from chunks of frames called a Group of Pictures (GOP). These GOPs consists of three different frame types:

1. Intra-frame (I-Frame)
2. Predicted-frame (P-Frame)
3. Bidirectional-frame (B-Frame)

An I-frame is a complete standalone frame which can construct a full image by itself. P-frame uses the information from a previous frame together with new information in order to construct the full image. A B-frame as the name implies takes information from both the previous frame and the coming frame to construct a full image. Concretely, this means that I-frames have little or no compression difference from a

normal compressed still image. A P-frames can further compress the data by predicting the content when constructing the image. And lastly, B-frames can compress the data even further by having even less information by looking at both past and future frames [15]. Each chunk (GOP) must start with an I-frame and can thereafter contain P-frames and B-frames interchangeably depending on the encoded data. There can be multiple I-frames in a chunk. For example, in fast-paced video content such as rapid sports games, there is a greater need for more I-frames. Whereas in more static cases such as video conference calls, the CODEC does not need to use as many I-frames and can thus reduce the bitrate by using more P- and B-frames to construct the video.

2.1.2 Muxer/Demuxer

The process of multiplexing/demultiplexing the content will not change the source data itself but can change the way in which the data is packaged. The H.264 stream will add with other data componets and meta data that together make up a new mp4 file. Several streams of data becomes a new package with new labels (headers). The process of demuxing the data is the reversed process of muxing. The process of taking the mp4 file apart extracts its data, stream by stream.

2.2 Blockchain

Blockchain technology is a ledger containing all of the executed transactions for devices in a logical network. The ledger grows with each block appended in a linear chronological fashion. Each node in the logical network maintains a copy of the blockchain which together with cryptographic hashes (see Section 2.2.4) and a consensus algorithm prevent undetected tampering with the blockchain[16]. The ledger is tamper proof, hence all of the information in the ledger is locked in and can not be changed by anyone without completely altering the blockchain, see Section 2.2.10. There is a great variety of blockchains and each serves a different purpose. A blockchain can be distributed, for everyone to use, thus everyone will keep a copy of the blockchain. Alternatively, a permissioned blockchain requires that a user has permission in order to gain access to the blockchain. Both distributed and centralized blockchains have advantages and disadvantages. These different blockchain solutions will be further elaborated in Sections 2.2.10 through 2.2.15.

2.2.1 Blockchain Background

The different techniques used by blockchains are not new. The first research on cryptographically connected blocks was proposed by Haber and Stornetta as early as 1990 [17]. In 1998 a mechanism for decentralized digital currency called bit gold was proposed. But it was not until 2008 when a white paper was written under the pseudonym Satoshi Nakamoto told the world about Bitcoin [18]. The area is still maturing but products, solutions, and collaborations with many high profile members are being developed all the time all over the globe, including Hyperledger [19] and R3 [20] (Richard Branson's COALA) to name two of the blockchain initiatives [21]. Blockchain has evolved from a technology revolving around tokens (Bitcoin) to a technology more adapted and suited for a specific purpose. Moreover, this purpose changes with every application of the technology.

2.2.2 Permissioned/Permission Free

As stated earlier, there are two main blockchain types: permissionless blockchain and permissioned blockchain. The former are public for everyone to see, such as Bitcoin. This type of blockchain can also be used in a closed environment where only users with access to the environment have the possibility to read and write data in the blockchain. In a permissioned blockchain the users need to trust a third party, i.e. the provider of the blockchain. The advantage of using a trusted third party with a centralized blockchain is that the system can use faster consensus algorithms and as a result transactions can be done in close to real-time. In contrast, decentralized distributed blockhains makes use of consensus algorithms such as proof of work (PoW), which is further described in Section 2.2.6. In the case of Bitcoin, the decentralized blockchain verification of a transaction could take hours. The advantages and disadvantages of the two proposed models will be discussed in Sections 2.2.5 and 2.2.6.

2.2.3 Blockchain and Trust

Blockchain technology is reinventing trust. Trust in a third party is unnecessary when using a permissionless decentralized blockchain network. In this setting, a blockchain makes use of middlemen unnecessary. A decentralized distributed blockchain does not require any trust between parties since each transaction is inserted via a consensus algorithm and subsequently everything can always be proven and verified, i.e. every transaction is stored in the blockchain and no one will be able to claim subsequently that a transaction did not take place. Due to the indisputable nature of the blockchain, every piece of information stored in the blockchain can be verified by anyone at anytime. The indisputable nature of the blockchain makes everything verifiable and very easy to track, all they way back to the genesis (original) block.

2.2.4 Cryptographic hashing

The general idea of hash algorithms is that they act as one-way deterministic functions, i.e. you cannot use the output of a hash function to determine the original input. This deterministic nature means that the same input will always reproduce the same output. The secure hash algorithm 2 (SHA-2) is the third generation of the family of SHA algorithms. Note that the previous generations have been deemed insecure for use in practical applications [22, 23]. The SHA-2 family consists of SHA224, SHA256, SHA384, and SHA512. SHA224 and SHA384 are slightly modified and truncated versions of SHA256 and SHA512 respectively. All of these algorithms break the input data into subsets, then these subsets are used to calculate the output bits. The operations in this computation consist of logical bit operations, such as modulo addition, bit rotation, etc.

The SHA256 algorithm accepts arbitrary length input sequences but always produces a fixed length output containing (256 bits generally written as a 64 hexadecimal characters), hence there exists $2^{256} \approx 1.15 \times 10^{77}$ unique outputs. Additionally, for an ideal hash function any change in the input sequence should produce a very different output value. This is due to the Strict avalanche criterion (SAC) which states that given a change of one input bit, the probability of a change in each output bit must exceed 50% probability [24].

In Examples 2.2.1 and 2.2.2, the second input sequence "hello!"* has one extra character, yet yields a completely different hash satisfying the SAC. One concern regarding the calculation of hashes is the usage of too few characters in the input sequence in order to produce a hash. A related problem is the collision risk, i.e., two different input sequences produce the same output hash. Currently there are no publicly known ways of hacking the SHA-2 family of algorithms. Attacks include brute forcing the input birthday attack, or collision attack [25]. One recent blog post by the security team at Google exposed vulnerabilities in the SHA-1 algorithm using a collision attack to produce the same output hash from two different PDF files. The attack named "shattered" is 100 000 times faster than normal brute force [23]. The complexity of brute forcing a short input sequence is directly related to the length of the input sequence. For example, if an input is a three character long lower case alphabetic sequence, then the number of possibilities for that sequence is $26^3 = 17576$. Whereas if the input sequence consists of long arrays of data (for example, binary data or data containing upper and lower case characters, arabic numerals, special characters, etc.) vastly increases the number of possibilities as: C^L, where C is the number of elements in the set of characters used and L is the length of the input sequence.

Example 2.2.1: SHA256 input: "hello"

2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Example 2.2.2: SHA256 input: "hello!"

ce06092fb948d9ffac7d1a376e404b26b7575bcc11ee05a4615fef4fec3a308b

2.2.5 Consensus

Blockchain sets out to solve the problem of coming to agreements across multiple nodes/devices. Disparate consensus opinions can be caused for a couple of reasons. First and foremost, when multiple nodes must agree on a common block they are influenced by neighboring nodes in the network. Assuming that neighbors can easily spread, share, and agree on the same block. But problems arise when the nodes cannot easily communicate with each other (for example, due to long delays or lack of network connectivity).

In addition, there may be hostile nodes, i.e., nodes that purposely disrupt with the intent to cause harm to the system. The blockchain must provide a solution which can facilitate consensus while managing misbehaving nodes.

2.2.6 Proof of Work

Proof of Work (PoW) was initially proposed as a method of deterring spam/denial of service (DoS) attacks on services. The concept relies upon the fact that by requiring users/services to compute a small (mathematical) problem in order for the service to approve the user/ service, this "work" deters casual attackers. In a

*This can be tested by using "echo -n hello! | sha256sum" in a linux terminal.

blockchain, these proof of work problems are typically small cryptographic puzzles that help validate blocks in the blockchain. The most distributed block will become the next block, thus coming to a consensus regarding which is the next block in the blockchain. Additionally, there is an incentive to solve these small cryptographic puzzles, as the mining node could be compensated with a fraction of a unit of cryptocurrency. For this reason PoW solvers are called miners, as they mine for cryptocurrency by helping to validate transactions.

2.2.7 Proof of Stake

Proof of Stake (PoS) is an alternative method of proof of work. The fundamental difference between the two lies in the blockchain participants not competing for the next block; but rather a block creator is determined (pseudo-randomly) by how invested (in the stake) in the blockchain the person/entity is. Additionally, there is no compensation given to the stakeholders. PoS does not "mine" in the sense of PoW, hence there is no financial cost associated with appending blocks to the blockchain.

2.2.8 Byzantine Fault Tolerance

The Byzantine Generals Problem [26] describes the situation where a number of generals have to agree on a common order to execute with regards to the enemy army. Having a common strategy in a battlefield is essential when striving for victory. The decision to either retreat or attack is vital information that can determine victory or defeat. These generals only have access to couriers who carry messages amongst them. If one of the generals is in league with the enemy force, then that general can sow discord by sending false messages to the remaining generals. These false message will influence the remaining generals and thus the common plan for the entire army will be in disarray; for example, with some generals preparing for an attack, while others prepare for retreat. An illustration of this problem can be seen in Figure 2.1. The example of Byzantine Generals Problems relates to blockchain in terms of *which* blockchain ledger is to be approved and appended, despite one of the generals being suborned. The blockchain together with PoW (see Section 2.2.6) is one of the first digital solutions to the Byzantine Generals Problem. However, there are other solutions (see for example PoS in Section 2.2.7).

The Byzantine fault tolerance (BFT) model can be described as a method for coming to agreements among the generals. In order for a common decision to be agreed upon (i.e reaching a consensus) a certain percentage (67%) of these generals needs to agree in order for a consensus to be reached.

In the blockchain approach to PoW consensus, the BFT model lies on the opposite side of the spectrum when compared to the Bitcoin PoW model [27]. The BFT model is built upon a small set of trusted nodes, instead of trying to reach consensus with the complete set of untrusted nodes, these trusted nodes must obtain a 67% consensus.

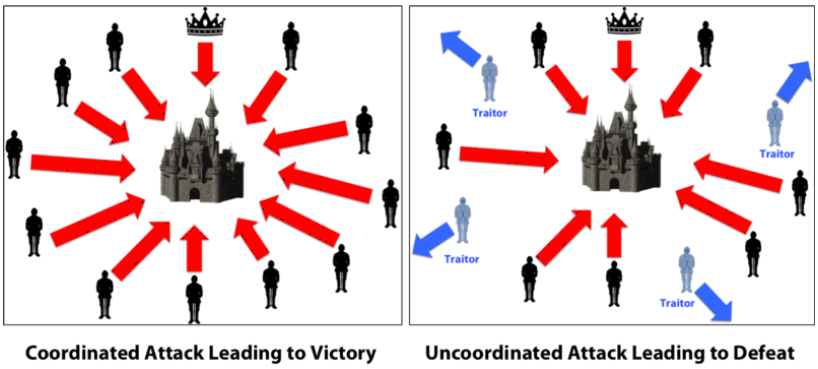

Figure 2.1: Byzantine Generals Problem [1]

2.2.9 Smart Contracts

Smart contacts are small frameworks or scripts built on top of a blockchain in order to control functions. Smart contracts are the intelligence of a blockchain, applications stored on the blockchain. This application brings new functionality and since it is stored in the blockchain it is impossible to change (without altering the entire blockchain). Smart contracts may be written in several Turing complete scripting languages, e.g. Solidity see Section 2.2.13. Initially, blockchains (such as Bitcoin) were able to perform simple tasks connected to cryptocurrency and the management of the transfer of those currencies. Since then, smart contracts have evolved to manage more demanding tasks. Smart contracts are sometimes referred to as smart agents since the word “contracts” is associated with a single use case. Once a smart contract is created, it can act autonomously when called on to perform an action [28]. A smart contract can act as a third party or middleman when exchanging money, property shares, or anything else that can be represented digitally. Smart contracts are able to do this in a transparent and conflict-free way. The agreement will be defined in the code of the smart contracts and when all parties have fulfilled their part the exchange will be made according to the contract[29].

2.2.10 How Does Blockchain Work?

A blockchain is a chain of blocks, such as shown in Figure 2.2.

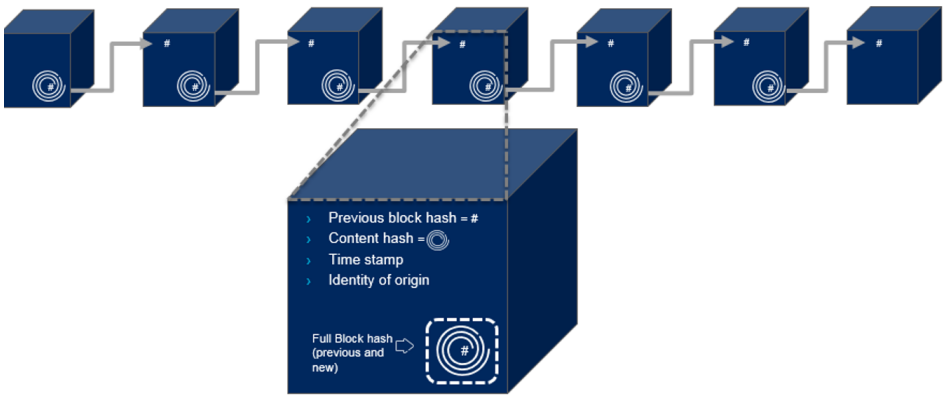

Figure 2.2: Blockchaining

The new block is created by hashing the hash of the previous block together with the hashed content of the present block and a timestamp. This way the new block will always be dependent upon the previous block and interlink them in an indisputable manner, as seen in Figure 2.2. If a block in the blockchain is interchanged with another block, then the chain is broken and a new shorter blockchain is created (whose length depends on which block is replaced). Every block beyond the interchanged block will be "thrown away"(i.e., is no longer valid). Depending on which protocol is used by the surrounding blockchain nodes, the protocol may discard the shorter blockchain and keeps the longest, according to the consensus algorithm (see Figure 2.3).

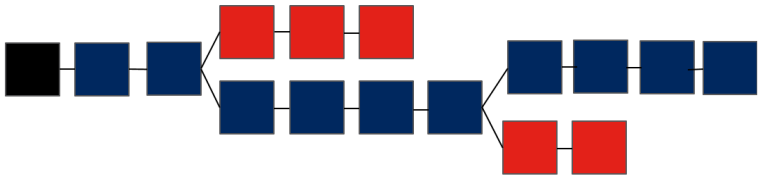

Figure 2.3: Shows the path of the longest chain

The blockchain will keep the blockchain that the majority of the users consider to be the correct one. This could be the majority of the network if the network is permissionless or if the blockchain is permissioned it will be the majority of trusted nodes. This means that a consensus-based blockchains will keep the unaltered blockchain. The only way to alter the blockchain is to have the majority of nodes in the system believe the altered blockchain is the correct one. This would occur if an entity controls more than half of the computing power of the network (in the case of PoW) or if the party owns more than half of the assets (in the case of PoS). To have 100% control requires 100% of the assets of the system.

The fact that every transaction is stored and that nothing is ever deleted or taken out of the blockchain means that the blockchain will constantly grow. Therefore, the blockchain will grow with time and usage. This constant growth can create a problem of scalability for which there is no clear solution. One approach to limit the growth of

the blockchain is to use hashes instead of the actual information. Any amount of digital information can be hashed and the resulting size of the hash will be no more than 32 bytes (with the use of SHA-256).

2.2.11 Application of Blockchain

Following Satoshi Nakamoto's 2008 white paper on blockchain for cryptocurrency, in 2009 the same author released the first source code for the Bitcoin (BTC) application. Bitcoin has since grown rapidly to a market footprint of around 39 billion U.S. dollars (as of 12 Jul 2017)[30]. BTC is a decentralized cryptocurrency using a permissionless blockchain that every user has the ability to download and thereby check. The users of the network use their personal computational power to verify transactions through hashes, i.e., mining. The miners are given a computationally difficult problem to solve. Furthermore, this task increases in difficulty as the computational power of the network increases. The Bitcoin protocol aims to append a new block to the Bitcoin blockchain every 10 minutes. This distributed PoW is a huge waste of energy but the strategy makes Bitcoin very hard to control in a centralized manner. The total computational power of the Bitcoin network is estimated to be more than 256 times the power of the world's 500 top supercomputers [31]. The entire Bitcoin network has a transaction speed of less than 7 transactions per second[32], which is a major concern regarding scalability. In comparison, leading credit-card payment companies average 2000 transactions per second, with the capability of sustaining 10000 transactions per second [33].

2.2.12 Hyperledger

Hyperledger is a project under the Linux Foundation which seeks to provide a unified blockchain architecture for industrial applications. Hyperledger is a collaboration of more than 100 organizations focused on banking, supply chain, and other transaction networks [34].

The traditional blockchain used in Bitcoin relies on a PoW model for determining how blocks and transactions are verified. In contrast, Hyperledger uses the BFT system which relies on a few nodes working together to reach consensus (see Section 2.2.5). The advantages are the fact that it requires less distribution of ledgers to reach consensus. Fewer decision nodes result in lower latency, hence blocks can consensually be locked within milliseconds instead of the 10 minutes needed for the Bitcoin blockchain. As less time is needed to reach consensus, the performance in transactions per seconds increases manyfold.

2.2.13 Ethereum

Ethereum is a permissionless blockchain made as a smart contracts platform run by Ether (ETH)[35]. The platform is highly distributed with, compared to Bitcoin, faster transaction times of seconds instead of minutes [36]. However, Ethereum is still very slow compared to permissioned (non-distributed) blockchains such as Eris and Hyperledger. Ethereum makes it possible to create and use smart contracts and distributed applications to be run distributed over the network without any downtime, any third part interference, or fraud, as well as creating, handling, and holding cryptocurrencies [37]. This is done through the peer to peer network which constitutes

Ethereum together with ETH which fuels the network, the consensus algorithm to share the state of the network, and the Turing complete scripting language which enables users to write complex scripts (smart contracts). It is the consensus engine which sets the speed of the Ethereum network by updating the state of the network at specific time intervals (shorter than for the Bitcoin network). The Ethereum network is fueled by gas which is a unit connected to ETH which is used to pay for the storage and computations an application (smart contract) needs.

Solidity is a programing language build specifically to target the Ethereum virtual Machine (EVM) which in turn is the protocol to access the blockchain[38]. Solidity has a syntax similar to JavaScript and is the language in which many smart contracts are written. The smart contracts are then compiled into bytecode and fed into to EVM and thus they can operate on the blockchain. In Ethereum are every transaction and smart contract is saved on the blockchain. The blockchain is in turn distributed and the states are handled by the consensus algorithm.

2.2.14 Tendermint

Tendermint [39] is software for securely and consistently replicating an application on many machines [40, 41]. Tendermint is not a blockchain in itself but rather a general purpose blockchain consensus engine with a consensus layer for blockchaining. Tendermint can be used as a replacement for the consensus engines of other blockchains [41, 42]. Moreover, Tendermint is a platform built for creating and developing blockchain applications. The platform does not provide any functionality and needs an application to run on top of it. Tendermint runs a BFT algorithm and requires at least 3 out of 4 validating nodes in order to validate transactions. If the Byzantine criteria are unfulfilled, then transactions will be stored in the memory pool of the system to wait for verification. A node connects to the network as an observer and is not required to be a validating node [40]. Tendermint is a permissioned blockchain consensus manager which gives it the ability to perform transactions very frequently (in the order of 1 transaction per second and up to 10 000 transactions per second) [43]. Tendermint is also highly flexible since it uses Docker (a container platform) to launch applications.

2.2.15 Eris:db

Eris:db is a controllable (permissible), smart contract-enabled, PoS based blockchain design with the PoS based on Tendermint. Eris:db is an application layer for blockchain applications. Eris is the backbone for deploying and interacting with the application logic [42]. Eris builds on both Tendermint, for consensus, and Ethereum for smart contracts. Compared to Tendermint, Eris has more functionality already built in. Eris is an implementation of Ethereum and uses the Ethereum transaction mechanics, with additional features for a name-registry, permissions, contracts, and keys for signing transactions [44]. Eris and Tendermint both use Docker to launch applications. This makes it highly flexible, since it can run on almost every computer [45]. Compared to Hyperledger, Eris is less focused on financial transactions and more focused on smart contracts and the functionality they offer, both hyperledger and Eris have the same advantages due to being a permissioned blockchain with a high transaction rate.

2.2.16 Cumulus

Cumulus is a blockchain solution developed by Ericsson*. Cumulus builds on and works with the Ethereum Virtual Machine (EVM) but separates the consensus algorithm and the blockchain from the EVM. Cumulus works similar to Ethereum; however, it is a private blockchain which means it does not run on *gas*. This brings Cumulus a lot of flexibility, while still having the benefit of the smart contracts mentioned in Section 2.2.9. A client running Cumulus will be able to call (use) and deploy (install on the blockchain) smart contracts with remote access to the EVM, the consensus mechanisms and the storage server (which could be a blockchain). Depending on the use of the blockchain different needs arise. Cumulus may use a distributed blockchain with a suitable consensus mechanism, similair to Bitcoin in Section 2.2.11, Ethereum in Section 2.2.13, or it may be equipped with permissioned blockchain running on a number of trusted servers. It is possible to run Cumulus with a regular database but still utilize the blockchain for consensus and smart contracts though the consensus client. Compared to the other blockchain technologies investigated, Cumulus is the most flexible.

The consensus hash server may be seen as a state manager for the system. The consensus server saves the hashes of all transactions in a long linked hash list, similar to a blockchain but without blocks. Depending on usage the system may utilize one manager, several managers, or even one manager for each node in the system, all depending on the level of trust needed for the system and the distribution of the system. The separation of consensus and storage gives the possibility to store and access information in a more efficient and more flexible manner compared to regular blockchain storage since the storage may be on a different storage unit.

The separation of the EVM, Storage server and Consensus server, is very clear in Figure 2.4. All communication with the world goes through the smart contract which is deployed in the EVM, the storage server, and the consensus server. The consensus server is a lightweight blockchain in which no data can be saved but it keeps track of the transaction and the chronological order of the transaction (much as how a blockchain works, as described previously). The separation makes the consensus mechanism very light weight and the information stored by the smart contract on the storage server may be stored in a database and yet maintain the same level of integrity as a regular blockchain.

Cumulus has a number of already developed smart contracts, one of them is an RSA manager. RSA is a public key cryptosystem used to secure and sign communications [46]. The RSA manager provides the sender the ability to sign and encrypt the message, the receiver is then able to verify the sender and decrypt the message.

*Specifically, Ericsson R&D Luleå

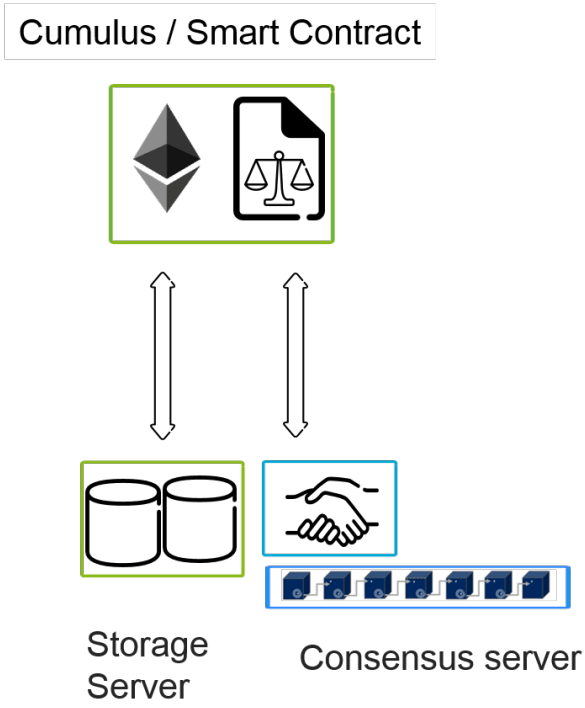

Figure 2.4: Diagram detailing the proposed blockchain solution

2.2.17 Which Blockchain Solutions to Use?

There are a number of blockchain solutions to choose from, some of the most important ones are described in the previous subsections. For this Master's thesis project we chose to work with Cumulus because of it flexibility and due to the availability of in-house knowledge of Cumulus at Ericsson. Moreover, Ericsson has developed useful APIs which are used by the web server.

Table 2.1 provides an overview of the previously discussed blockchains and how they differ. Bitcoin is added for comparison due to its popularity and people's general knowledge about it.

Table 2.1: Comparison of Described Blockchains

	Distribution	**Transaction speed (per second)**	**Smart Contracts**	**Consensus Algorithm**	**API**
Hyperledger	Permissioned	Fast (10k)	Yes	BFT	No
Eris:db	Permissioned	Fast (10k)	Yes	Tendermint(PoS)	Yes
Ethereum	Permissioned Free	Slow (20)	Yes	PoS	Yes
Tendermint	Permissioned	Fast (10k)	Yes	Tendermint BFT	No
Bitcoin	Permssioned Free	Very slow (7)	No	PoW	No
Cumulus	Permissioned	(untested)	Yes	Yes (unknown)	Yes

2.3 Secure Connection

In this Master's thesis project data integrity is of paramount importance. In order to be able to trust the data stored within the blockchain, a reliable secure connection is created between the application (app) running on an Android device and the computer that is computing the blockchain (as shown in Figure 2.5. Only meta data and hashes are sent between the device and the server where the blockchain computations are done, therefore it is impossible for a third party to know what the plaintext of the video is. As a result encryption of this hash might be of less importance. However, it is crucial for the system to know that the integrity of the hash is intact. The video captured and stored at a device can later be verified as to what is received by another device by using the blockchain. An HTTP connection is used between the device and the server. The HTTP connection will in future implementation (see Section 6.2) be replaced with a HTTPS connection to secure the communication.

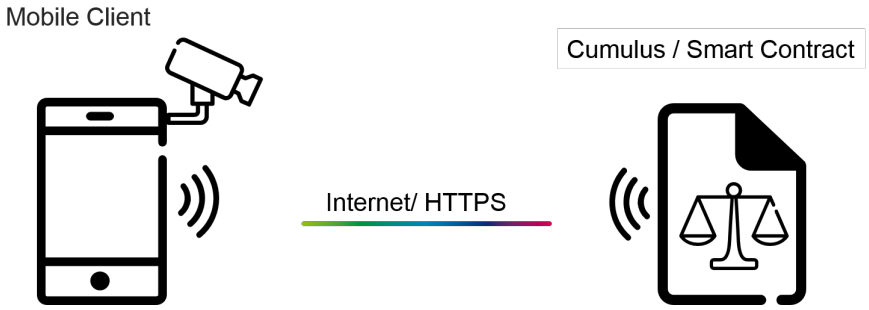

Figure 2.5: System diagram of connection between client app and server

2.4 Android

The Android platform is a mobile operating system based on the Linux kernel and developed by Google Inc. Application software development for Android is done through a Software Development Kit (SDK) and preferably through the official Interactive Development Environment (IDE) Android Studio. Android app development is done with a Java back-end and a front-end interface layer built on XML. For this Master's thesis project the minimum SDK API level has been set to 21 (Android 5.0 Lollipop) in order to gain access to the features in the Camera2 API. The Camera2 API can handle more complex requests and can be customized more when processing images and videos than the older camera API. The new API is described in further detailed in Section 2.4.1.

2.4.1 Camera2 API

The Camera2 API is the latest suite of camera APIs provided to Android developers by Google [47]. Introduced for API level 21 (Android 5.0 Lollipop) and above. The previous camera API has since been deprecated and is not recommended for use in app development. Although the old API is used in backward compatible apps, the new API provides a plethora of new features to developers. Camera2 models the camera device as a pipeline which can send and receive requests. This API can be used to change

requests and alter settings on the fly, while still maintaining the same preview session seen by the user. The API does not differentiate between still pictures and video, i.e. unlike the deprecated camera API, a video is simply treated as a rapid succession of images and thus they are processed in the same way.

2.4.2 MediaCodec

MediaCodec is a class in Android which can access low-level media APIs [48]. The class is used to specify encoding/decoding schemes, frame rate, bit rate, resolution, etc. The MediaCodec can be used in conjunction with the Camera2 API to setup and handle specific requests. The MediaCodec class in the context of this set up is according to the flowchart shown in Figure 2.6. Empty input buffers are initialized, transmitted from the CODEC, and handed to the input client. In this context, the input client is the camera. These input buffers are filled by the camera data in the Raw file format and transported back to the CODEC. The CODEC empties the input buffers and encodes the input buffer's data into a specified video format and places the output into output buffers. These output buffers can be used to encode video files, send the encoded video over streaming protocols, etc. When an output buffer is filled, it will be transported to the receiving client (in the figure this is shown as a Media Multiplexer (Muxer)). Once the output buffer is emptied, it is transported back to the CODEC to serve new requests. The transport between CODEC and client on both the input and output side is done asynchronously. Asynchronous buffer handling means that the CODEC does not have to wait for one task to finish before starting a new one. Thus input buffers can be emptied and filled simultaneously without the need to wait in line when encoding an input buffer to fill output buffers in the CODEC.

The Android MediaCodec API uses the H.264 baseline profile*. The baseline profile is designed for low-cost applications with low processing power. The more limited baseline profile (compared to more complex profiles in the H264 standard) excludes support for more complex colour spaces (YUV4:2:2†, and YUV4:4:4‡). The profile also limits the frame types to I-frames and P-frames (as described earlier in Section 2.1.1).

*Support for H264 Main Profile was added in Android 6.0 and onwards.
†The color depth corresponds to 4 bytes of data per 2 pixels.
‡The color depth corresponds to 3 bytes of data per pixel.

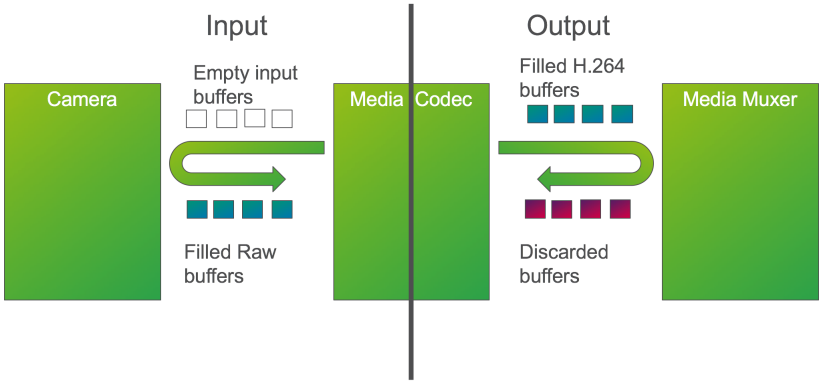

Figure 2.6: Input/output buffer flowchart of the Android MediaCodec Class

2.4.3 Media Muxer

The Media Muxer class was introduced in Android 4.3 (API level 18). The class multiplexes different streams and media data (audio, video, etc.) [49]. The Media Muxer class can be used together with the MediaCodec class (described in Section 2.4.2). As shown in Figure 2.7, the Media Muxer accepts video, audio, and other meta data and multiplexes them into a single container file. When using the MPEG-4 (.mp4) container format, the muxer is able to include meta data requests such as geolocation and device orientation. This geolocation (location data) can potentially be used when generating the unique output hash. Given that the location data could be read from the mp4 container file on the receiving side it would be possible to know where the device claims it was (assuming that the location data is trusted in the original video).

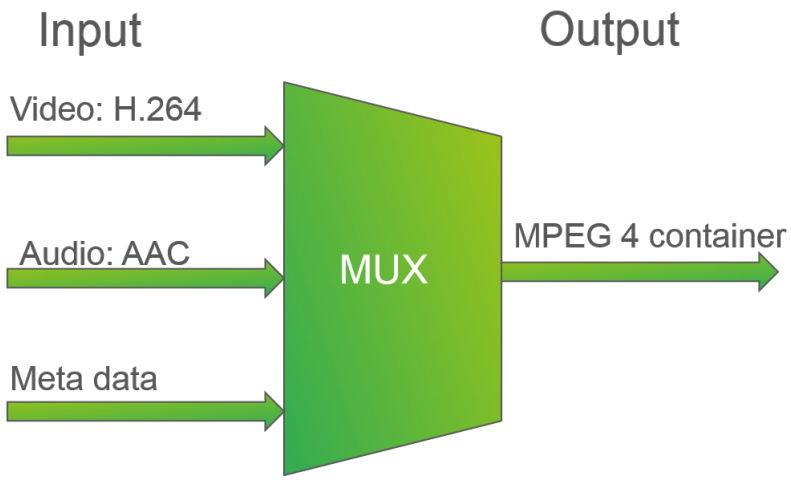

Figure 2.7: Example of a setup using the Media Muxer

2.4.4 MessageDigest

MessageDigest is a built in Android class that has been included since the very first public version Android 1.0 (API level 1) [50]. It is included in the "java.security" package. It should be noted that java is the programming language that Android is built on top of. The MessageDigest class provides methods for calculating various kinds of hash functions. By default the MessageDigest class can digest data and produce Message Digest 5 (MD5), SHA-1, SHA-2 (SHA-224 all through SHA-512) hashes. Section 2.2.4 gives more information regarding these functions.

2.4.5 Android security

Each app running on the Android OS runs in a secure sandbox. The components that together provide a secure environment include* [51]:

- The Android operating system is a multi-user Linux system in which each app is considered a different user.
- By default, the system assigns each app a unique Linux user Identity (ID) (this ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine, so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it is no longer needed or when the system must recover memory for other apps† [51].

Using the principles of least privileges [52], each app is isolated so that it can only access its own resources. By default, an app only has access to the most basic resources necessary to run the app. Additionally, an app needs permission in order to gain access to resources and modules from the OS, e.g. Input/Output write permission, camera services, GPS, etc. Furthermore, as of Android Marshmallow (6.0), an app also requires the user to explicitly grant permission to the app to use the requested permission upon the first launch of the app. All of these features together creates a secure environment around each app built on top of the Android platform.

2.5 Web Verification client

The verification of the video is done via a web page hosted on a web server. For the purposes of testing in this thesis project, the web server runs locally on the host computer (Macbook, see Section 1.6 and the web page is accessible by our web browser via the local host on port: 8080, i.e., http://localhost:8080/. The web server is built in JavaScript. JavaScript was chosen as it is a very common front-end development language. The JavaScript together with HTML (content of web page) and

*Portions of Section 2.4.5 are reproduced from work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.

†Which is not an issue since our app does not run in the background.

CSS (defining the web page's design) constitutes the web page retrieved from the web server.

2.5.1 JavaScript

In this Master's thesis, Node.js* is used as the run-time environment together with the jQuery† library installed on the server. Node.js runs there servers, i.e., the consensus server (see Section 2.2.16, the storage server (see Section 2.2.16), and the web server (see Section 2.5). Note that all three servers run within the node.js environment on the computer hosting the web server and blockchain. JavaScript is the front end development language which realizes the functionality of the web page. JavaScript works together with HTML and CSS in order to deliver a nice looking functional web page.

2.5.2 HTTP Server

In order to easy communicate with the blockchain from both the mobile Android device and the server running the verification client Representational States Transfer (REST, also called RESTful)[53] APIs used over the HTTP protocol, usually to the server's (TCP) port (80). A RESTful API is basically a web server architecture which enables clients to make a request of servers. The HTTP protocol builds upon a request/response relationship between a client and a server and in this case used within the RESTful API architecture. In HTTP a string is sent to a specific port on the server with a specific method predefined by the HTTP standard. The most common methods are: GET, HEAD, POST, and PUT[54]. The POST method is used by both the Android device and the web client of this Master's thesis project to give both the ability to add and via the reply to this POST request to get information from the server.

2.5.2.1 POST Method

The POST method is a standard method in the HTTP protocol and gives the user the ability to add and receive data from the server[54]. A HTTP transaction session is established through a TCP connection to a specific port on the server. After the TCP connection is established, of plain text HTTP message is sent to the other who either confirms the transaction or responds with an error message. It is in the body of this message that the response to the sender will be sent. Within the scope of this Master's thesis the response will either be a boolean variable showing a successful transmission or a string of hashes requested from the blockchain.

Example 2.5.1 illustrates a regular HTTP POST together with the cumulus specific request and the actual input arguments used by the prototype. The first line constitutes the header of the message. Below the header is the body of the message. All of the Cumulus data is written in Base64 encoding. Base64 is used to encode binary data into 8-bits ASCII string format. It utilizes a base of 64 and is able to write "a-z", "0-9", and "- _" or "+ /"[55].

*https://nodejs.org/en/
†https://jquery.com/

Example 2.5.1: Client Side- HTTP POST with body

POST /index.html HTTP/1.0
Address: example@example.com
Username: username
Password: password
Function: call/deploy
EVM: 3a66203a35a176540802cf5c51dd6e842d880d3a
Contract: 53684f50367ad35480bd330224a75b6430fd985c
Arguments: input

Another approach to sending a HTTP POST is to write all of the information in the header and skip the body of the message, such a message is shown in Example 2.5.2. This is the type of message used in this prototype. The design choice of the REST server is made by the developer of Cumulus and can therefore not be answered for by this Master's thesis project.

Example 2.5.2: Client Side- HTTP POST Without Body

POST http://address:password@example.com/call/EVM/Contract/input

2.5.2.2 Hypertext Transfer Protocol Secure

Hypertext Transfer Protocol Secure (HTTPS) is an encrypted version of the HTTP protocol[56]. HTTPS is usually used for sensitive information such as banking, private information, or classified information. HTTPS usually uses port 443 rather than HTTP port 80. In order for a user to be able to trust that a server is who it says that it is-signed, third party certificates are used, usually in conjuction with the use of the Secure Socket Layer (SSL) as a secure transport protocol. A third party provides the server with a signed certificate which the user of the service verifies. Now that the user has verified the identity of the client and the server has verified the identity of the client - and encryption between the two parties may be set up. The security depends on that the certification chain work, that the user knows which server to use and that the user (client) notices when the certificate is incorrect or missing. Furthermore, in order to provide a high level of security the keys and the algorithms used need to be strong.

2.6 Related Work

This section presents related work done both in the academic world and the industry.

2.6.1 Academic Research

Three different academic papers have been identified as relevant to this research. The following subsections highlight their main points and relevance to this Master's thesis project.

2.6.1.1 Timestamping video footage in traffic incidents

In the context of the model proposed in this Master's thesis, the conference paper "Securing Video Integrity Using Decentralized Trusted Timestamping on the Bitcoin Blockchain" [57] is particularly relevant as the scenario is close to what this Master's thesis investigates.

In this paper Gipp, Krosti, and, Breitinger investigated the use of the Bitcoin network to timestamp a video feed from a smartphone camera placed in a car in the event of an accident. They describe the use of an algorithm using the accelerometer of the smartphone to detect traffic accidents. When the magnitude of the accelerometer readings exceeds a certain threshold, the app initiates the video recording and storing functionality of the smartphone. The camera's output is processed by the smartphone, hashed, and this hash is sent as a transaction on the Bitcoin network. The hash will be stored in the Bitcoin network and subsequently be accessible to anyone but temperable by none. The stored video can be verified as the original video content as captured at approximately the time of the timestamp.

2.6.1.2 Trusted Timestamping

With regards to trusted timestamping two reports were identified: "Trusted Timestamping" [58] and Commitcoin [59]. Both solutions leverage the time stamp made using the Bitcoin protocol when creating a transaction together with the carbon dating nature of the blockchain (i.e., one can tell the rough date of an entry by looking at the sequence of time stamps). These two solutions are slightly different in terms of their execution, but the basic theory of using the existing Bitcoin blockchain is the same. The paper "Trusted time stamping" uses a Time Stamping Authority (TSA)[60] on top of the Bitcoin functionality. The plain text time stamp from the TSA is added to the hashed information of the transaction and hashed again before being added to the Bitcoin blockchain. Commitcoin does not use a third party to create a time stamp, but rather relies entirely on the blockchain to do this. Both solutions are based upon a block associated with the transaction being appended to the blockchain to which subsequent blocks will be appended.

2.6.1.3 Forensics Investigations of Multimedia Data

In [61], R. Poisel and S. Tjoa review the latest trends in forensic investigations of multimedia data, i.e. images, videos, and audio files. They describe different methods for determining what has been done to images and expose fabrications down to which details within a picture have been tampered with. Being able to tell what parts of an image are inconsistent with the rest of the image is often of particular interest. However, the ability to distinguish such disparate elements in a picture is outside the scope of this project, as we simply wish to prove that a video sequence has or has not been tampered with. However, it is worth mentioning that applying state-of-the-art research done in video forensics will likely be the next logical step when investigating source material deemed to have been manipulated.

2.6.1.4 Digital Watermarking

I. Echizen, et al.[62] insert digital watermarks into video files to detect data tampering. They begin by breaking a video file into its composonents: the video, the audio, the

timecodes, and the header. The header is used together with the timecodes to separately watermark the audio and video. After watermarking, the parts are then combined and sent over a given channel. By separating audio and video, the watermarking can prove which form of data tampering has been done and whether the video or audio has been altered (such as shifted, replaced, or deleted, or if the header has been manipulated).

2.6.2 Industrial Research

Three different industry solutions have been identified as being relevant. The following subsections highlight their main points and relevance to this Master's thesis project.

2.6.2.1 Nexan - Assureon Archive Storage

Nexan Assureon™"Assureon Archive Storage"[63] took an approach similar to that to be used in this Master's thesis project. Fingerprints are created from the data in order to prove the integrity of files within a data archive system. The original files are stored on at least two different disks or at two different geographical locations. The fingerprints are later used for verification of the files. However, they have not described a method for safe storage of the fingerprints. In contrast, this safe storage of the fingerprints is a central goal of our solution.

2.6.2.2 Enigio - time:beat

The product series "time:beat" includes: time:shot, time:stamp, time:grab, and time:mail: by Enigio. This series shares similarities with the proposed solution in this Master's thesis. For example, "time:beat" is a blockchain solution for the purpose of archiving time stamped fingerprints of integrity sensitive materials: email, pictures, documents, and websites. These fingerprints are stored in a permissioned blockchain controlled and owned by Enigio. The fingerprints are accessible from their webpage for verification[64].

2.6.2.3 Ascribe

Ascribe helps artists to create a digital copy of their work and time-stamp it in the Bitcoin blockchain. When a file is uploaded, Ascribe creates a digital certificate which can be traded, tracked, or loaned via the blockchain. Ascribe uses an open source protocol called "SPOOL"[65] to interact with the Bitcoin blockchain[66].

2.7 Background Summary

Table 2.1 summarized the differences and similarities of the blockchain platforms discussed in Sections 2.2.12 through 2.2.17. The different parts and aspects of the proof of concept prototype were broken down and analyzed. From the smallest components of the blockchain consensus algorithms, to which existing blockchain platform should be utilized. Section 2.3 talked about the communication between an Android device and a server. Furthermore, while there exists some related work, there seems to be a gap when it comes to solutions combining blockchains with video integrity. The closest related work we have discovered was discussed in Section 2.6.1.1, where blockchains are used to enable Android devices to be used as dash-cameras to create verifiable video

clips for use in conjunction with traffic accidents. Chapter 3 provides an overview of the proposed proof of concept prototype.

Chapter 3

Implementation

The main objective of the thesis is to realize a proof of concept prototype of the proposed system using a laptop computer and an Android device.

The live video content is hashed in real-time in chunks of predetermined size(s) by the Android device. The size depends on the buffer sizes used by the Android device when filming and encoding raw footage, the speed of the Android system on the device, and speed with which the blockchain is able to receive hashes. When the video content is hashed it will be signed by the Android device and sent to the blockchain. These hashes are appended with a time stamp upon arrival in the blockchain. Later, the video may then be verified by using the web client part of the system. The web client will emulate what the Android device did when filming the video by breaking the video into the predetermined size chunks with the same time alignment as the original video. The web client hashes the content of the video starting with a specific time frame and ending with another specific time frame and then compares the hash with the hash found in the blockchain. If the two hashes do not match, then the video chunk has been manipulated.

The proposed system is meant to simulate the live streaming feature of social media platform apps such as the Facebook app. When a video is uploaded through the system proposed in this Master's thesis it will be an easy task for the user to later determine if the video was produced (approximately) when claimed by the publisher of the video content (in this case the party that uploads the video using the proposed system).

This chapter introduces the design of the prototype in Section 3.1. In Section 3.2 the the functionality of the mobile client is described step by step manner together with a screenshot. This is followed by a description of how the prototype utilizes the hashing function in Section 3.3. The web client is described in Section 3.4 visualized with a screenshot. The description of the web client will identify the specific data sets to be hashed. This is follow by a description of how the smart contracts work.

This chapter details the proof of concept prototype. The chapter both presents all of the discrete components that will be combined and provides details of the parameters used to fine-tune and analyze the complete system.

3.1 System Design

The system is built in three parts: the web client, the mobile client, and the smart contract (see Figure 3.1). For the mobile client running on the Android device, there are a lot of parameters that need to be tuned in order to achieve the best configuration and still have the performance required for the prototype. The parameters that will be investigated for the entire system are:

1. Buffer chunk size (these define the size of the chunks are used for hashing),
2. Hashing algorithm(s), and
3. Encoding settings (frame rate, bit rate, resolution, etc.).

The smart contract is the only part which interacts with the blockchain.

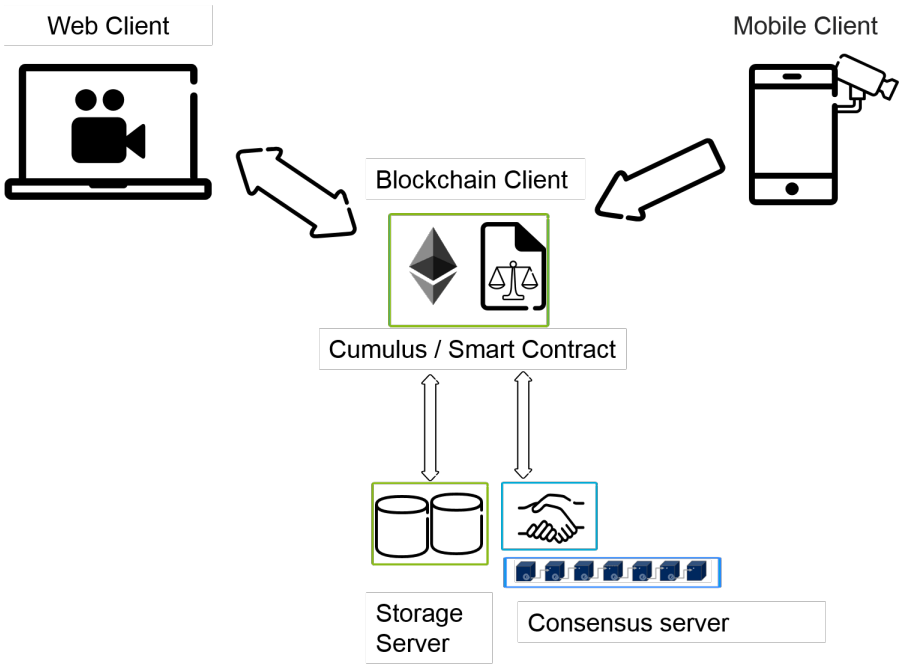

Figure 3.1: Prototype overview

3.2 Mobile Client

The mobile client running on the Android device executes several concurrent tasks using multiple threads. These tasks are:

1. Record video produced by the camera,
2. Encode raw camera feed in to H.264,
3. Acquire meta data (e.g. geolocation),
4. Multiplex video, (audio), and potential meta data into a mp4 container,

5. Fetch output buffers, append meta data, and run through a SHA-2 hashing thread,
6. Send the hash from the SHA-2 thread to HTTP client running on the mobile client, and
7. Transmit the hash from the HTTP client to server client.

As mentioned in Section 2.4.2 the MediaCodec handles input buffers and output buffers asynchronously. This means that the buffers can be used concurrently without needing to wait for earlier buffers to be processed as shown in Figure 3.2. The MediaCodec will send the H.264 filled output buffers to the MediaMuxer. Simultaneously, the SHA-2 thread will intercept the output buffer of the encoded video data. The SHA-2 thread also processes the acquired meta data, if there is any. The hash produced by the SHA-2 thread is sent to the Mobile HTTP client and sent to the smart contract through a HTTP POST request.

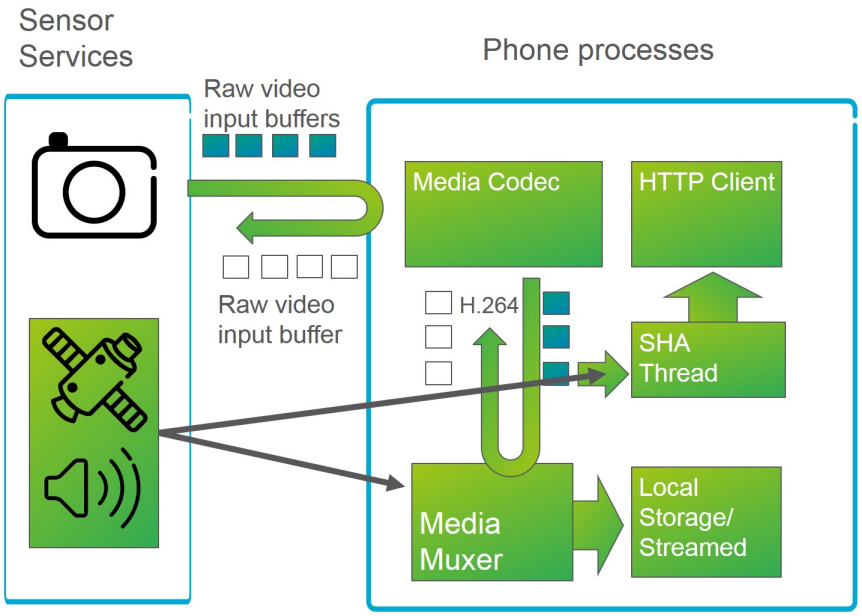

Figure 3.2: Mobile client system

Simultaneously with the SHA-2 thread, the MediaMuxer combines video, (audio), and meta data to create a combined container file in .mp4 file format. When the phone stops recording, an .mp4 file will have been generated and the app will stop sending hashes to the HTTP client (and by extension the smart contract) once all the output buffers have been emptied and the last block of the file written.

An Android app has been developed targeting Android devices running 5.0 Lollipop (API level 21) and above. The app interface can be seen in Figure 3.3. Running Android 5.0 gives developers access to the Camera2 API as described in Section 2.4.1 and the MediaCodec class introduced in Android 4.1 as described in Section 2.4.2. The interface of the Android app can be seen in Figure 3.3. The left picture in Figure 3.3

shows the start screen of the app. If the button “Recording Session” is pressed the right hand picture appears and the app is ready to record.

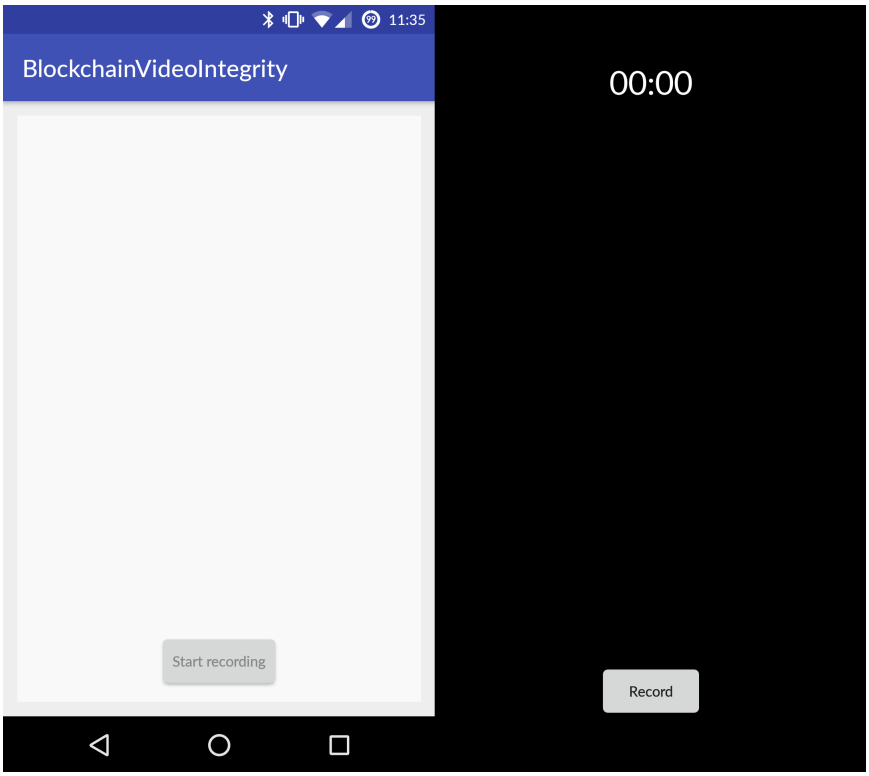

Figure 3.3: Screenshots from the mobile client interface (left: Initial view, right: after pushing the Recording Session button)

3.3 Hashing

The security of the blockchain is to a great extent derived from the hashing algorithm used by the system. Longer and more complex hashing outputs makes brute forcing much harder (as described in Section 2.2.4), but it will also require more resources from the devices running both the mobile client and the web client. In real-time the mobile phone will hash the video which is computationally demanding (but less demanding than video encoding) and the web client will need to do the same in a short amount of time (otherwise the service might not be used by an end user) potentially over very large video files. The potential usage of both of these clients requires a well thought through and well balanced hashing algorithm, as will be described in Chapter 4.

The hashing algorithm will operate on chunks of data from the output buffer from the MediaCodec (as described in Section 2.4.2). These chunks consist of GOPs as decribed in Section 2.1.1. These chunks will be used to compute the hashes to be incorporated into the blockchain. The chunk size will be evaluated in Chapter 4. A private-public key-pair will be used to sign these hashes. A smart contract is used as a trusted third party and acts as the Certificate Authority (CA) to store all the public keys

of the system. The public keys are stored in the blockchain or on the consensus and storage sever (see Section 2.2.16), therefore the public keys are accessible by anyone connected to Cumulus with the correct access rights.

3.4 The Web Client

The verification of the video is done by the web client and the interface is shown in Figure 3.4. The web client performs the same computations as the original mobile client on the media content and uses the hashes from the blockchain (accessible online) in order to verify the integrity of the video clip.

The web application is simple and features a drag and drop window to upload the video. Any video may be uploaded with a size limit of 1 GB. The video is then divided into the same sets of frames of the video and each set of frames is hashed with the same hashing algorithm used on the mobile device. The hashes are then compared with the hashes stored in the blockchain. If the same hash produced by the web client can be found in the specified time span of video content, then, the hash is considered verified.

In order to communicate with the smart contract there are two parameters needed: the Virtual Machine address of the EVM and the contract Address (for details see Section 2.2.16). Cumulus runs within an Ethereum Virtual Machine and several of these virtual machine can be operated on the same time within the cumulus runtime environment. The virtual machine address is the address used in order to separate the machines from each other. The virtual machine address may be arbitrary but uniquely chosen by the deployer of the contract (not done by the verification client). The Contract address in generated by Cumulus and given as a response to the deployment of the contract. The end-user of a system such as this proposed prototype will not be in contact with either the virtual machine contract nor the contract address. This should be handled by the system. The date span seen in Figure 3.4 added is the span of the dates the the verification client calls upon the smart contract to get hashes for. Thus the client will be able to verify the integrity of videos for all of the dates within the date span.

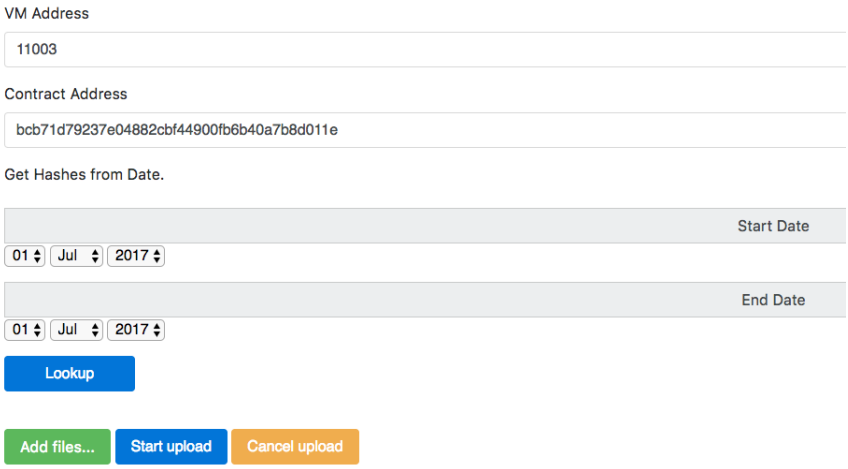

Figure 3.4: Screenshot from the Web client interface

3.4.1 Demultiplexing & Hashing

It is of utmost importance to be able to tell exactly where to divide the video into sets of frames at a frame by frame level and to know exactly the correct size of chunk to use. If the specification of the start and end frames is done with only one bit of offset the resulting hashs will be completely different (see Section 3.3) and therefore it will be impossible to match with the hashes received from the blockchain (for verification). In order to correctly determine where in the video input one should start and end (i.e. how long the input bitstream should be) we will specify the specific I-frames of the bitstream that are to be processed. Each chunk of input starts with an I-frame and then depending on the preset length of the input chunks, either one or more chunks are processed (hashes) until the ending I-frame appears. These I-frames are the foundation to determine where the hashing processing must start and end. If the preset input size of the chunks is less than the amount of data between I-frames, then several hashes will be computed over that data set.

The process of identifying I-frames is done using the headers from the H.264 stream which have been parsed through the mp4 muxer. The byte sequency idicating an I-frame in the H.264 stream is “00 00 01 65” or “00 00 00 01 65” in Hexadecimals, this can be written as “0 0 1 101” and “0 0 0 1 101” in 8-bit unsigned integers (unit8) (the units used in this project). The data headers are changed when going through the media muxer in the android device. When parsing an mp4 file, the sequence “0 0 X X 101 ” is the sequence for an I-frame, still in the primitive uint8 data format. The “X X” mid sequence are numbers being added by the muxer and indicate the size of the header, which will vary and therefore may not be used in the sequence when identifying an I-frame. The process of identifying P-frames is similar but with a different sequence to identify them(see Table 3.1.

Table 3.1: Header sequences

Header	Value (hexadecimal)	Value (uint8)
I-frame	00 00 01 65	0 0 0 1 101
P-frame	00 00 01 42	0 0 0 1 65

3.4.1.1 Find where a hash sequence start

As described in Section 3.4.1 the start of the video sequence being hashes is based upon specific I-frames in the video. However, since this needs sequence of data to be determined down to the exact starting bit, there is a need for great precision and accuracy in splitting the video into chunks at the proper place (and moreover the same place in both the mobile client and in the web client).

3.4.1.2 The Search Algorithm Comparing Hashes

The search method was originally designed as two for-loops comparing all hashes from the video with all the hashes received from the smart contract. This effectively gives an nxm matrix of values to compare and has a complexity of $\mathcal{O}(N \times M)$. In order to speed up the comparison two improvements have been implemented. The first improvement is a counter that increments for every matched hash. This counter is subsequently used as the starting value for the outer loop of the search in the next iteration and will cut the number of comparisons needed in approximately half. However, this method is only usable if the hashes are always in the correct order when being compared. It is also very important that both arrays being compared are approximately the same length or that the longer one is used in the outer loop. The second improvement made to the search algorithm to break the inner loop when a match is made. This potentially cuts the total number of comparisons in half. This effectively brings the complexity of the search algorithm to a complexity of $\mathcal{O}(N)$ when a matching video is being compared.

3.4.2 The smart contract

The smart contracts run on the blockchain within the EVM environment. The smart contract is a small script written in Solidity deployed on the blockchain which is called every time an interaction with the blockchain is made. These interactions are stored as transactions, the transactions are hashed and stored according to the consensus mechanism while the information carried in the transaction are stored locally on the server (laptop). The smart contract itself is deployed on the consensus server and therefore written into the transaction history which makes it impossible to change. The blockchain server and the consensus server runs on a laptop together with the web server application. However, the web application and the servers connected to Cumulus are completely separate and the only interaction between them are through the interface of the smart contract.

The Web application, EVM, the consensus server, and the storage server all run on the same laptop (see Table 1.1). However, this specific set up was done purely for convenience and any of these elements may run on any computer with an Internet connection, provided the communication bandwidth is sufficient and the connection is sufficiently reliable.

3.4.3 The separation of consensus and storage

Cumulus separates the consensus server from the storage server. This is very useful in some applications but not necessarily for this Master's thesis project. This separation makes it possible at a later stage change the storage since the the blockchain characteristics are located in the consensus server and not the storage server. The storage is separated from the blockchain which makes it very flexible to take, for the specific use case, the most efficient shape and form. Many of the blockchain's specific advantages for a system are, available from the consensus server, hence it is less interesting to actually save all information transmitted via the network into a blockchain. The integrity of the system will be guaranteed from the hash chain in the consensus server and the smart contracts may be implemented in the consensus server while the actual information could be saves elsewhere.

3.4.4 How the smart contracts work

The smart contract is built upon two functions: PUT and GET. The Andoid device uses the PUT function in order to add hashes to the blockchain and the verification client uses the GET function in order to retrieve the hashes for a specific range of dates.

3.4.4.1 Appending Information to the Blockchain

An HTTP POST function is used to append information to the blockchain. The response of the POST function for the put function is a Boolean variable indicating whether the hash was successfully appended or not. The hash is saved in a 'Struct'* together with the meta data of the video. The hashes of the struct are hashed together with previous transactions via the consensus algorithm, the resulting hashes are saved on the consensus server while the data from the transaction is saved on the storage server.

3.4.4.2 Retrieving Information from the Blockchain

The GET function looks through the entire blockchain to find dates matching the date specified in the web interface by the user. The hashes from the matched dates are added to a list and returned to the verification client.

3.4.5 The Connections

The communication between the different servers of the system are done over HTTP. This is mainly done due for convenience, robustness and to make the system as device agnostic as possible. Almost all devices are able to communicate over the HTTP protocol and for security reasons will future implementations use the HTTPS protocol (see Section 6.2). This proof of concept prototype was implemented on a local network and at a small scale, therefore it is important to strive towards compatibility with security standards, such as HTTPS, but less important, at this stage, to actually realize the level of security needed for full scale operations. For these reasons, this Master's thesis project has chosen to implement the HTTP standard for communication. However, the REST server is HTTPS-ready but uses self signing certificates in order to show functionality without the expense of getting certificates

*A Struct is a data type declaration in Solidity. A struct defines a list of physically grouped variables placed under one name.

signed by a third party CA. Future implementations of this model should investigate whether it is feasible to use smart contracts as a trusted third party and certification provider. This will be further discussed in Section 6.2.

All communications involving the smart contract are done through HTTP POST requests to the API server.

3.5 Performance limitations

The limitations of this Master's thesis project are divided into three different groups: limitations in the verification client, limitations in the Android device, and limitations in the server. All 3 groups of limitations have different constraints and some of the limitations limitations are interlinked.

3.5.1 Limitations concerning the Verification Client

One limitations in the verification client is the language in which the functionality is written. JavaScript is a front end scripting language and it is not optimized for highly demanding back-end computations which makes the processing less efficient and as a result the system is slower that it might have been if another programming language has been used. On top of this is the verification client is running on the same Macbook (Section 1.6) as the server running the consensus server (Section 2.2.16), the storage server (Section 2.2.16), and the REST server (Section 2.5.2) which may limit the system's performance when the system is under maximum load. The system is expected to be much faster if concentrated solely on the verification process in the web browser. However, combining all of this processing on one computer was easy to accomplish. This issue is further discussed in Section 6.2.

3.5.2 Limitations Concerning the Android Device

The MediaMuxer offers methods that can multiplex various kinds of meta data into the actual .mp4 file. However, multiplexing pure string information through the Android MediaMuxer class has not been introduced on a per frame level for Android version 5.0. The latest Android version (Android 8.0 O Developer Preview) includes functionality for frame by frame string information multiplexing. Thus, the proof of concept app has omitted the frame number string that was originally proposed. That being said, frames have to be verified through identification of the relevant source material on the verification side. The proposed signing of each hash cannot be done with Android 5.0, the actual solution of the problem for frame/GOP identification as a future work. This means that with the current solution, the hashed buffers could be scrambled internally and would be still be able to verify each hash found in the blockchain. (Barring the construction of the video based on frame sub-types described in Section 2.1.1.) Additionally, audio has been omitted in the video multiplexing. The prototype does not need to include audio for proving that the proposed solution can function. Therefore, the prototype is limited to the footage from the device's camera.

3.5.3 Limitations Concerning the Server

The server running the smart contract and the blockchain were not developed by the authors of this Master's thesis. The current server is limited to around 20 transactions

per second. However, this limitation is believed (by the authors) to be mainly due to hardware limitations and lack of optimization. The server is a lightweight REST server developed to serve the purpose of the POST request. The server is light weight and should therefore not be resource demanding, however it is not optimized for its specific purpose. The authors believe the main reason for the performance limit is hardware based since the prototype has been tested under different amount of stress. The higher test levels put on the system has resulted in a drastically lowered throughput of hashes on the REST server, which has been interpreted as a hardware limitation. However, this is not further investigated and considered outside of the scope of the scope of this thesis (see Section 6.2).

Chapter 4

Evaluation Framework

The results of this project will be evaluated in two steps: the first step will be a binary evaluation of whether this solution can provide verification of whether a video clip has been modified. The second step investigates the performance when verifying the integrity of the video clips. The foundation of the system is video integrity, but if the performance is poor few people will use it. Moreover, if the system is hard to use, then no user will use the system. Therefore, it is of great importance that a balance between performance, usability, and assessing video integrity is found in order for this solution to potentially be deployed and used. As a result the following criteria will be used to evaluate the system:

- Evaluate the added load on the Android system when performing the original hashing and signing.
- Evaluate the effect of the size of the content buffers to be hashed on the Android device in order to match the performance of the Android system and the blockchain.
- Measures the time needed in order to do verification via a web browser for an averaged sized video. In this work we will consider this time to be reasonable if it takes 60 seconds or less for video clips up to 600 seconds of video duration[†].
- Identify which hashing algorithm is the most viable. Are there any advantages to using different hashing algorithms?
- Evaluate the smart contract created. Measure the performance limitations it puts on the end system.

Section 4.1 describes an overview of the tests performed. Section 4.2 details how the evaluation of the built-in hashing function is done. Section 4.3 details the evaluation when appending various video lengths to the blockchain. Section 4.4 describes the evaluation of a blockchain that is growing in size. Section 4.5 looks into how to evalute the search algorithm for the blockchain.

[†]The 600 second recording translates to 225Mb of data given the parameters set in Table 4.1

4.1 Tests performed

The tests utilized the evaluation framework introduced above. These tests mainly concern aspects of the web client, and the Android device. These tests focused on constraints and limitations that will persist even if more powerful servers are used to deploy the model, i.e. the computing capacities of an Android device are compared using the prototype used for testing. The tests are conducted using a fixed set of parameters in order to eliminate artifacts and conduct the tests in a reproducible manner. For obvious reasons some tests will deviate from the expected parameter values given the nature of the test when testing a certain parameter. Additionally, the majority of the tests are done with exactly 10 measurement points. 10 points was chosen so that outliers and anomalies in the data sampling could be identified, while still being able to evaluate the system in a reasonable amount of time. Evaluating the prototype gives only some indication of the actual limitations of the system. A test consists of recording and transmitting hashes to the smart contract. After a recording is finished, the video file stored locally on the Android device is transferred to the verification side for verification. Given these circumstances, a lossless video transmission is assumed. The video is then uploaded via the verification web client and used for further testing.

Given the testing environment, the results for the server side testing must be seen as an indication of what the results might be when implemented on dedicated servers running the blockchain, rather than absolute performance values. As the goal was not to have high confidence, but rather simply an indication of the likely performance.

Table 4.1: Set of fixed parameters

Parameter:	**Value:**
Hash function	SHA-256
Recording time	60 seconds
Buffer size	1 frame
Resolution	720x1280
Bitrate	6Mbps
Framerate	30fps
Hash transmission rate	30hps
I-frame interval	5 seconds

Given the parameters in Table 4.1, 60 seconds of footage will contain up to 45MB of data. The bit rate is specified as an upper limit that the MediaCodec class will not exceed.

4.2 Test 1 - Hashing

This test is deigned to test the performance of the MessageDigest class in Android (described in Section 2.4.4). The MessageDigest class is a built-in class that digests byte arrays and returns a hash value corresponding to the input data. The design of the test is as following:

1. Receive byte array from MediaCodec class

2. Start timer before MessageDigest method call

3. Call MessageDigest class with byte array from the MediaCodec

4. Stop timer after MessageDigest returns corresponding hash value

5. Write elapsed time, and input byte array size to log file

The test is done using the standard parameter settings specified in Table 4.1 for each of the algorithms that the MessageDigest class can use (see Section 2.4.4). Roughly 1800 measurements were made for each hash function. A histogram of the size divided by time for each hash function was compiled. The results will be presented in Section 5.2.

4.3 Test 2 - Performance of Videos of Variable Length

This test is designed in order to characterize the system when verifying videos of different lengths. The test is designed mainly to see how the time for different processing steps varies with the length of the video and to evaluate the accuracy of the system by measuring the proportion of matched hashes. The time from a contract call to a response is measured from the time the verification client makes a request until the server responds based upon using the smart contract. The verification client makes a call to the server via the API. The client connects to the blockchain via communication with the smart contract. The smart contract searches for those hashes corresponding to the request and returns a verification response to the client.

Under the prerequisites that a video has been recorded, and hashes sent to a newly deployed contract on a new VM-adress. The test is designed as follows:

1. Upload a video (locally stored) through the web interface

2. Press “Lookup”

3. Wait for response from smart contract

4. Save response time and video length in a log file.

5. Deploy new contract on new VM-address and start over.

The verification time is the time it takes for the verification client to input a video, digest it into chunks, hash these chunks, and compare each hash with the hashes previously received via the smart contract. The time for a blockchain call and verification constitutes the total time it take for the system to verify the video.

The fraction of hashes that match indicate the accuracy of the process. The test environment is assumed to be a system with zero communication losses, when transmitting and handling both the video and hashes. This means that match results below 100% indicates shortcomings in the prototype. This test is to verify a correlation between the accuracy of the system and the length of the videos used for testing.

All tests in Test 2 are conducted with a newly deployed contracts and without previously saved data. The results will be presented in Section 5.3.

4.4 Test 3 - Analysis of Changes in the size of the Blockchain

This test investigates whether the performance of the system is affected by the amount of data stored by a deployed contract. The test is based upon comparing the 'verification time' and the blockchain call time'. Each video is recorded and the hashes of the video is added to the smart contracts. The number of video hashes stored by the smart contract will increase with each iteration of the test since the same contract is used for every video. It is always the latest video taken that will be subject to the verification process.

Under the prerequisites that a video has been recorded, and hashes sent to a single contract with one VM-adress. The testing process is as follows:

1. Upload a video (locally stored) through the web interface
2. Specify the current date for smart contract look up.
3. Press "Lookup"
4. Wait for response from smart contract
5. Save verification time and blockchain call time to log file
6. Record new video to the same contract and the same VM-address with either the same, or new* date on the next recording.
7. Start over

This test utilizes two methods. The first method simulates the case when all videos are taken on the same day, hence all hashes are saved in the blockchain with the same date's time stamp. This means that when the call from the verification client asks for the hashes of a specific date, all hashes accumulated from that day will be sent in the response to the verification client. In the second method all videos will be sent to the blockchain with unique dates. This means that the verification client's request will only match one video (the last one recorded) and this video's hashes will be compared to all of the accumulated hashes obtained by the other method.

The test starts by deploying a smart contract. The first test method responds to the request by sending all hashes (from all videos in the test) back to the verification client. The second method only sends the hashes from the latest video back to the verification client. The difference between these two methods will show how different parts of the system are affected by the growing blockchain. The methods will be evaluated based upon 10 iterations per method. During the test the blockchain will grow based upon the 10 videos' corresponding hashes. The results will be presented in Section 5.4.

4.5 Test 4 - Analysis of the Search Method

This test is designed to evaluate the speed of the search algorithm used in the verification client for matching the hashes received from the smart contract with the

*Used in the second method when all videos have different dates

hashes generated from the video. As mentioned in Section 3.4.1.2 the speed of the search algorithm is very dependent upon the amount of matching hashes. A larger number of matches will yield a faster search. By comparing the number of video hashes from the Android device with the correct video and an incorrect video of the same size will examine how much faster the verification client is when the video is a perfect match compared to a 0% match. The relevant measurement is the time for verification.

The prerequisites for this test is that a new contract, and new VM-adress is deployed for each new test point. The test is designed as follows:

1. Upload either a video with corresponding blockchain hashes, or a video without corresponding hashes* video (locally stored) through the web interface
2. Press "Lookup"
3. save search time to log file
4. Deploy new contract with new VM-address and record new video.
5. Start over

The results will be presented in Section 5.5.

*Video without corresponding hashes is used for when no hashes will match the content of the blockchain.

Chapter 5

Results and Analysis

In this chapter, the results of the tests described in Chapter 4 are presented. Additionally, analysis is done for each test and comments are made regarding their relevance and impact on the prototype.

Section 5.1 succinctly states all the results of tests on the prototype. Section 5.2 details the results regarding the different hash functions. Section 5.3 showcases the performance for videos of varied lengths. Section 5.4 presents the results regarding the performance with changes to the length of the blockchain. Section 5.5 presents the results for the execution of the search method by the web verification client. Section 5.6 presents a reliability analysis of the tests performed and the general setup.

5.1 Performance Results

The main performance results of the tests of the prototype are:

- The accuracy of the system is 98.1 % on average (this result does not vary depending on recording length) throughout all tests.
- The prototype is able to handle approximately 20 transaction per second.
- The response time for the HTTP POST request to the smart contract mainly depends on the amount of data being requested.
- The search algorithm is extremely fast for a perfect data set and the comparison of hashes is primarily a function of the time it takes to process the video.

5.2 Results and Analysis - Hashing

Figure 5.1 shows histograms corresponding to the execution time of the different hashing methods included in the MessageDigest class is plotted as a histogram of the message size divided by time. The tests were conducted using the parameter values from Table 4.1, varying only the hash function parameter. All histograms were generated using approximately 1800 measurements. The fastest method on average is the SHA-1 algorithm, while the slowest being the SHA-384 and the SHA-512

algorithms. Table 5.1 presents the equation and the corresponding goodness fit (R^2-value) for each of the tested algorithms. The linear regression does not represent the data well given the low values for the R^2-value.

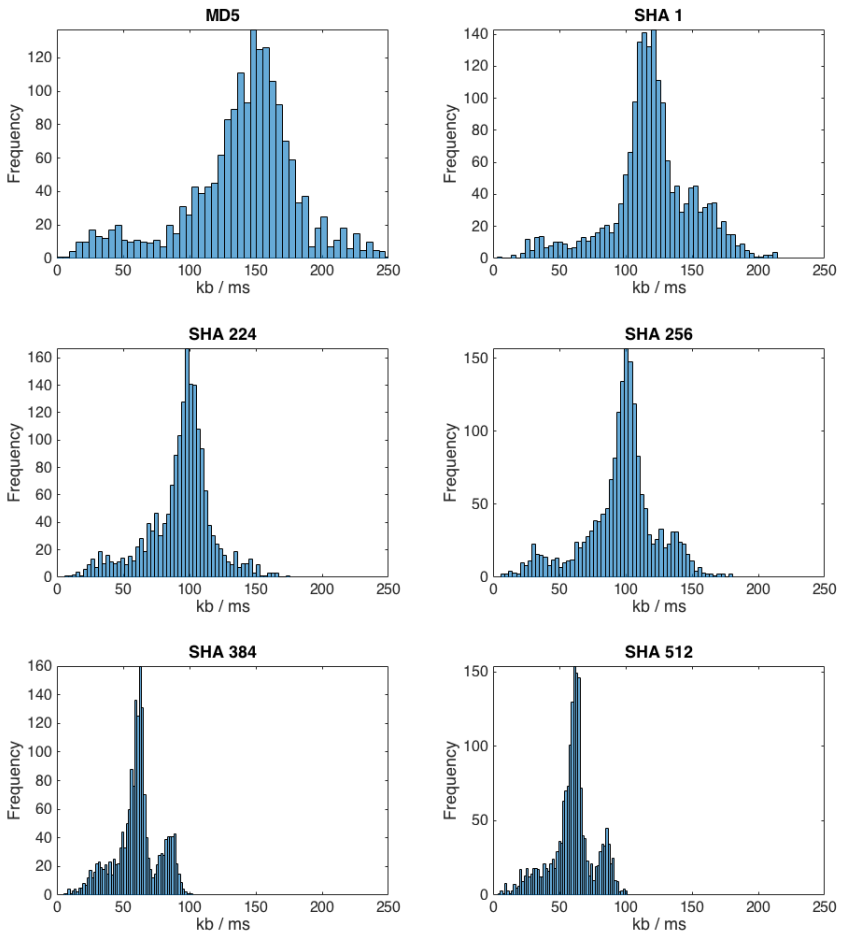

Figure 5.1: Histograms from the different hash methods using the MessageDigest Java class

Table 5.1: Equations and goodness fit corresponding to the different alorithms

Hashing method:	**Equation**	R^2**-value**
MD5	$y = 0.0346x + 17171$	0.210
SHA-1	$y = 0.0039x + 24237$	0.019
SHA-224	$y = 0.0309x + 16068$	0.272
SHA-256	$y = 0.0195x + 19477$	0.142
SHA-384	$y = 0.0196x + 16133$	0.308
SHA-512	$y = 0.0195x + 15972$	0.278

Given that both I-frames and P-frames will pass through the MessageDigest class means that using the mean of the data is not a good indicator of the average buffer size. The I-frame will always contain more information and thus skew the mean value for both time and size. In the case of the histograms shown in Figure 5.1, bigger buffer sizes will take longer time to digest. But the size difference is accounted for with the size divided by time factor which otherwise would have skewed the data in the histogram. Furthermore, The SHA-2 family of algorithms seems to perform more consistently than the deprecated MD5, and SHA-1 algorithms. In both SHA-384 and SHA-512 we can also see some bimodal tendencies. With a median size of around 23kb and median time around 0.30ms. This time is less than the time between two frames even when the camera is operating at 30fps (i.e. a per frame time of 33ms).

The results of the equations and R^2-values in Figure 5.1 for each algorithm is quite poor, a linear regression does not give a good indication of the relationship between byte array length and time it takes to digest the data into cryptographic hashes. The reason for this might be that there are a lot of concurrently running tasks within the Android Device that use the same resources. The computational resources are quite limited and the Android app might prioritize other concurrently running threads.

This test took 60 seconds for each algorithm tested. The test was conducted so that when the Android device was running, it was also storing the time, and size for each buffer that passed through the MessageDigest class.

5.3 Result and Analysis - Performance when processing Videos of Various Durations

Figure 5.2 illustrates the response time for a blockchain call from the web verification client via the smart contract. It also presents the linear regression, and goodness fit (R^2-value). As can be seen from the regression line in Figure 5.2 this is a linear process. Five measurements have been taken for each video length. The video length test points are: 15 seconds, 30 seconds, 60 seconds, 180 seconds, and 300 seconds. The test points span short video recordings from 15 seconds up to 300 seconds and incremental data points in order to see the performance on shorter recordings. Each measurement is the amount of time from when the verification client sends a call until the verification client receives a response.

As previously mentioned, calculating the regression between the test measurements

yields a linear function with the duration of the video. The linearity of the curve indicates that the process depends of the amount of information being processed and transmitted by the smart contract and the REST server. This is not a problem for shorter length videos, as in this test for videos up to up to 300 seconds in duration, but could potentially be a problem for much longer videos as a user is not interested in waiting a long period of time to verify the integrity of video. If the user needs to wait for a long time for verification, the user-friendliness would be quite lacklustre.

According to the evaluation framework in Section 4 the time for verification was desired to be under 60 seconds for a 300 second long video. In the measurements above in Figure 5.2 the 300 second mark yields a call time of 100 seconds. This is however easily managed by decreasing the number of hashes per second from the 30 hashes per second the system uses in this test. In Figure 5.3 the time for verification is plotted against the the time duration of the video. The linear regression, and goodness fit (R^2-value is also presented. As in Figure 5.2 the average is taken from 5 measurements for each different video duration. The length of the video is, as before: 15 seconds, 30 seconds, 60 seconds, 180 seconds, and 300 seconds. Each measurement corresponds to the time it takes for the verification client to digest the video, hash the smaller chunks and compare the hashes made from the video with the hashes fetched via the smart contract.

As can be seen in Figure 5.3 this is a linear process that depends on the duration of the video. As when discussing time for the blockchain call this is not a problem when handling smaller sized videos but could potentially become an issue when handling bigger videos since the time for verification is a linear function which will increase with the videos size. The curve indicates that the time it would take to verify a 300 second long video would be around 100 seconds. This means that the total time to verify a 300 second long video would be given as per Equation 5.1, where t_{bc} is the time to make a blockchain call, t_{vt} is the verification time, and finally t_t is the total time.

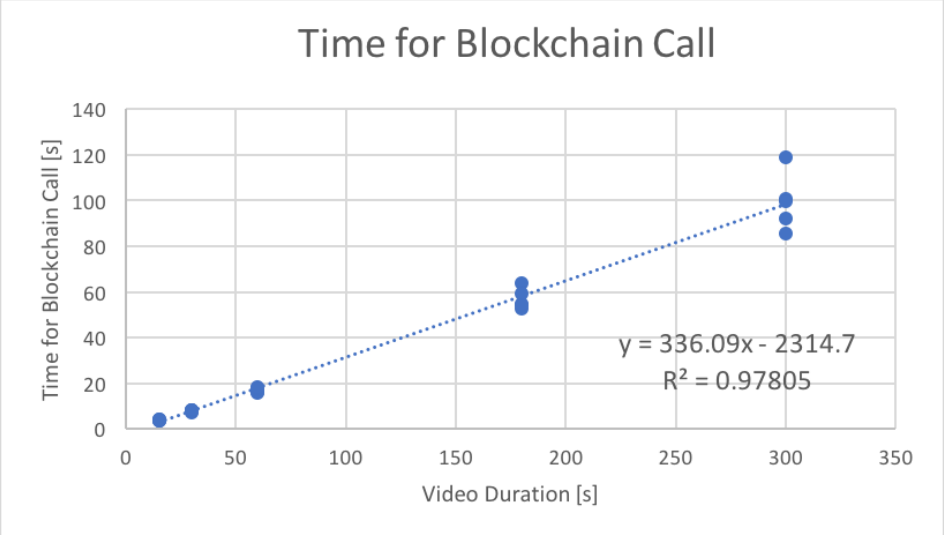

Figure 5.2: The time it takes for a blockchain call for videos of variable length

$$t_{bc} + t_{vt} = t_t \tag{5.1}$$

Since the evaluation framework in Chapter 4 clearly states that the upper limit of verifying a video should be less than 60 seconds for a 300 second long video this result is not acceptable. However, this is easily achieved by decreasing the frequency of creating hashes. In this test, 30 hashes are created per second (one per frame), this is both demanding on the hardware & battery and more frequently than would be necessary in most cases. This will be further discussed in Section 6.2.

Figure 5.4 indicates the accuracy of the entire system over different length videos, the same lengths as in previous tests. The linear regression equation and goodness fit (R^2-value) is also presented in the figure

In this plot every value from the test is plotted. The same data points are used from the previous figure, but with the corresponding hash match percentages plotted instead. As can be seen the spread between data points is greater for short duration videos, while the spread decreases with increasing duration of videos. The over all accuracy of verification of the videos are very high - with an overall average of around 98% which is considered good in this project. The cause of the error was not investigated, but will be discussed further Section 6.2.

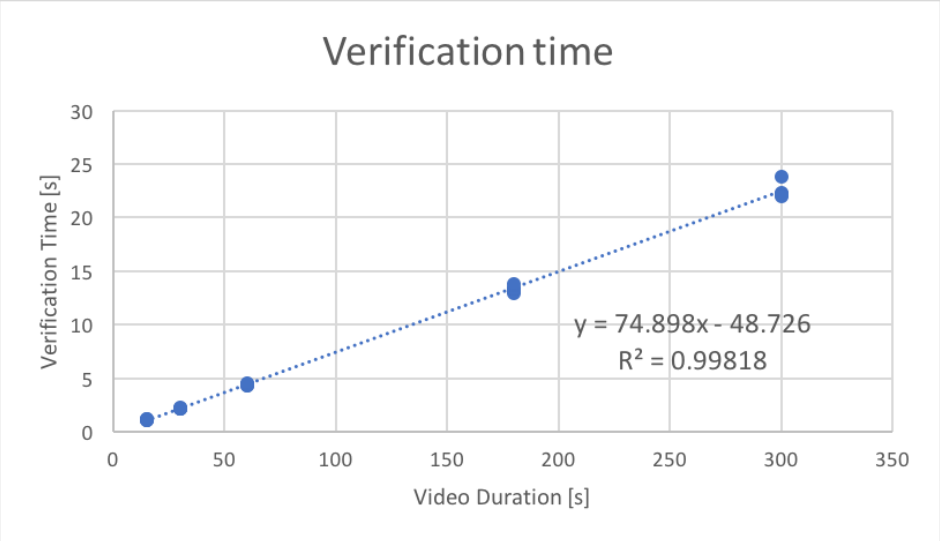

Figure 5.3: Time for the matching process of video and the hashes from the smart contract smart contract

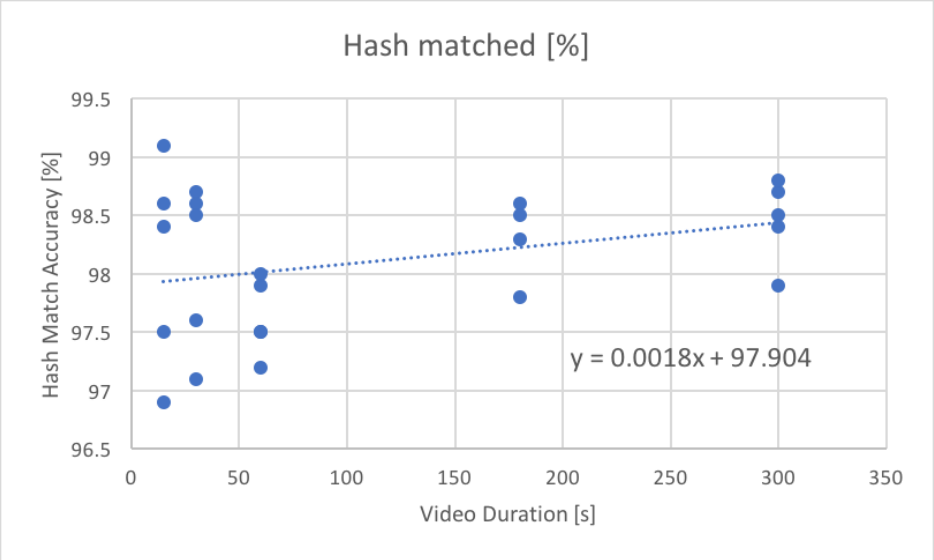

Figure 5.4: Percentage of hashes matching with video file length

One complete test iteration can be broken down to five time consuming events, each event with a different time consumption, see Table 5.2. For each iteration of the test is a new contract deployed which means that there will be no accumulated data from previous transaction for the smart contract to consider. The first event is to record the video from the mobile device. The hashes are during the recording simultaneously being transmitted to the smart contract. The second event is to upload the recorded video to a shared Google Drive folder. This is done in order to transfer the video to the verification client from the mobile device. The video is subsequently downloaded by the verification client. The time consumption of the upload and the download are estimated since they are not interesting for the solution, and thus were not measured. Simultaneously with the up- and download of the video (step 2 & 3) the blockchain call to the REST server is made (step 4). This means that the longest time of either the combination of step 2 and 3, or step 4 is considered when calculated the total time. The next event is step 5, verification of the video. The total time for one iteration of the test is the sum of the events necessary to preform the test. Five measurements for each pre-set time interval are recorded which makes a total of 25 measurements for each event. Since videos of longer time duration are bigger in size every step will have a bigger time consumption and therefore make every aspect of the test more time consuming.

Table 5.2: The Events and Time Consumption of one Test Iteration

Steps	**Event**	**Time consumption [s]**	**Comment**
1	Record video	15, 30, 60, 180, or 300	Section 5.3
2	Uploading the video	$20 \leq t \leq 60$	Estimated
3	Downloading the video	$30 \leq t \leq 70$	Estimated
4	Blockchain call	$4 \leq t \leq 120$	Figure 5.2
5	Verifying video	$1 \leq t \leq 24$	Figure 5.3
Total	One test iteration	$70 \leq t \leq 574$	

5.4 Performance as a function of changes to the Blockchain

The linearity of the process is independent of whether the the date searched for matches all of the video hashes or just the latest video's hashes. However, there is a clear difference in the slope of the lines. The line illustrating the time for a blockchain call for different dates has a lower slope.

The fact that both lines pass through close to origin indicates that there is little or no overhead of the system and that the time consumption of the system is primarily dependent of the size of the input data rather than the function call itself. The different lines illustrated two processes that are very similar to each other. The blue line illustrates the time taken when asking for all hashed data stored by the smart contract while orange line asks only for those hashes matching the specific video being verified. The search process being conducted by the smart contract is therefore different since less data is identified and sent back by the call function. This difference is clearly visible in Figure 5.5 from the difference in the slopes of the lines. The linear regression, and goodness fit (R^2-value) is also presented in the figure.

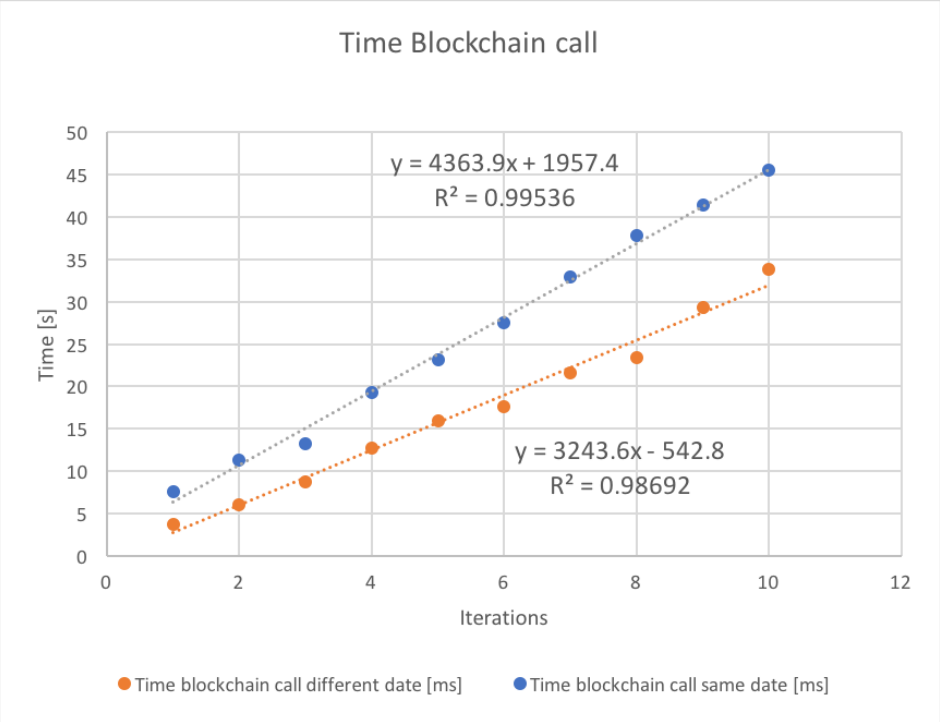

Figure 5.5: Blockchain call time as a function of changes to the Blockchain

Figure 5.6 illustrates the time for verification for the two cases tested. the figure also presents the linear regression equation, and the goodness fit (R^2-value). The process measured in this test is the time it takes for the web client to divide the video stream into frame sizes, hashing each frame, and compare each frame's hash with the hashes

received from the smart contract. Both lines illustrating the different test cases cross the y-axis at more than 1000 ms. The line illustrating the verification time for a video on the date of the last video has a very small slope while the line illustrating the verification time for verification of a video against all videos on different dates has a greater slope.

The fact that both lines cross the y-axis at a large value indicates the overhead of processing being done for every video independently of the data stored by the smart contract. This processing overhead is explained by the deconstruction of video into frames and the hashing of those frames. This processing was expected to consume a lot of resources. The difference in the slope between the two curves is explained by the search algorithm tested and discussed in Section 5.5. The search algorithm is design to be as effective as possible when comparing matching data sets. The smart contract always returns arrays the length of the data set stored by the smart contract. Thus, the search algorithm needs to compare partly empty arrays with the arrays based upon hashes of the video, which makes the search function less effective. This is further discussed in Section 6.2.

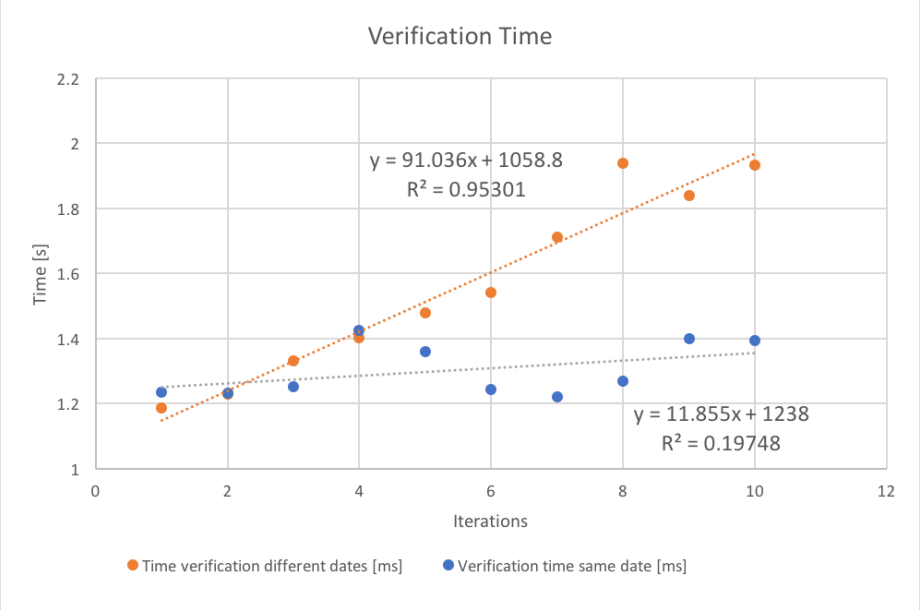

Figure 5.6: Verification time for two cases

The test are conducted very similar to previous test (see Section 5.3). The video is recorded, uploaded from the Android device, and downloaded on the verification client. The verification client sends a request for dates to the smart contract and receives a response. The video is broken down, hashed and compared (verified) with the hashes requested from the blockchain. The difference from this test compared to the test conducted in section 5.3 is that there is not a new smart contract deployed between the videos and the video hashes are accumulated within the same contract making search times longer for each iteration. The test is done in two ways: every video has the same date (see Table 5.3) and that the date of each video is unique (see Table 5.4). The difference in the tests result in different time consumption for one iteration of the test.

Table 5.3: The Events and Time Consumption of one Test Iteration, same dates

Steps	Event	Time consumption [s]	Comment
1	Record video	15, 30, 60, 180, or 300	Section 5.3
2	Uploading the video	$20 \leq t \leq 60$	Estimated
3	Downloading the video	$30 \leq t \leq 70$	Estimated
4	Blockchain call	$8 \leq t \leq 45$	Figure 5.5
4	Verification	$1.2 \leq t \leq 1.4$	Figure 5.6
Total	One test iteration	$73 \leq t \leq 475$	

Table 5.4: The Events and Time Consumption of one Test Iteration, different dates

Steps	Event	Time consumption [s]	Comment
1	Record video	15, 30, 60, 180, or 300	Section 5.3
2	Uploading the video	$20 \leq t \leq 60$	Estimated
3	Downloading the video	$30 \leq t \leq 70$	Estimated
4	Blockchain call	$4 \leq t \leq 34$	Figure 5.5
4	Verification	$1.2 \leq t \leq 1.9$	Figure 5.6
Total	One test iteration	$69 \leq t \leq 464$	

5.5 Result and Analysis - Search method

The Figure in 5.7 shows the slopes from the test described in Section 4.5. the figure also includes the regression, and goodness fit (R^2-value) for both the unmatched, and the matched video. The test is conducted using both a video with matching hashes, and a video without matching hashes. This test is conducted using videos with duration 15s, 60s, and 180s. The unmatched videos use a second order polynomial regression, whereas the matching videos use a linear regression. The test is conducted using three data points for video duration and each data point uses two iterations. In total 6 data points for unmatched videos, and 6 data points for matching videos.

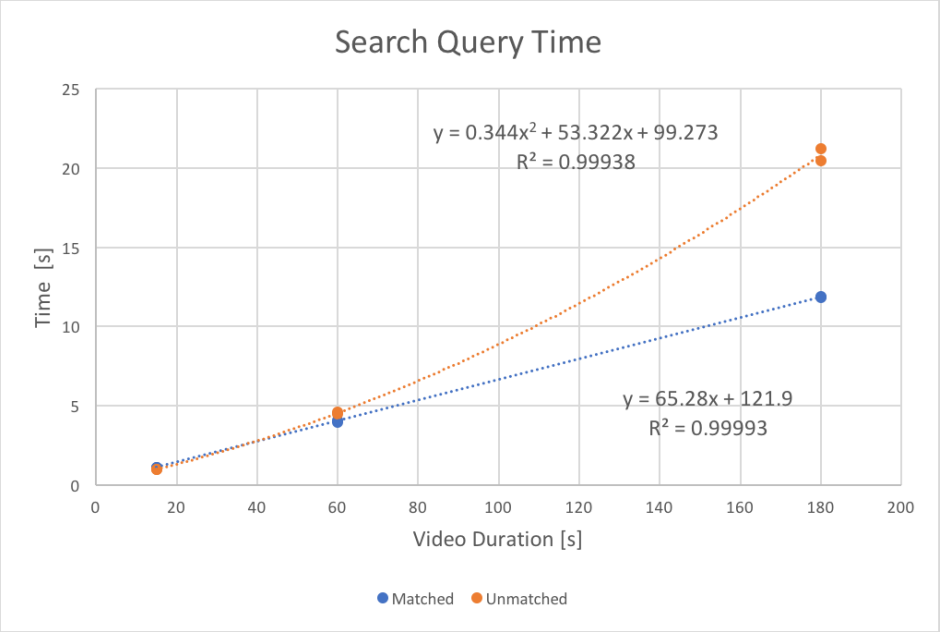

Figure 5.7: Verification time for two cases

As can be seen, the complexity for the unmatched video is shown to be of complexity $\mathcal{O}(N \times M)$, and the corresponding matched version has a complexity of $\mathcal{O}(N)$. Note that this search algorithm could be further improved by using a hash table which is discussed in Section 6.2.

5.6 Reliability Analysis

The proposed system was tested in an open office environment on computers and phones running software out of the control of the Master's thesis project group. The internal processes of both the Macbook and the Android device are hard to control. However, the tests were conducted with the minimum of programs, applications, and background processes running on both systems. The running background processes can be found in Appendix A. The wireless network connecting the devices was a network accessible for anyone working in close proximity and therefore was not isolated. With that said, the tests were done in the exact same way, and the local network used is a private network only accessible to few individuals, with the consensus of the authors being that the test are reliable.

Due to the time consuming nature of the tests being conducted the number of data points are limited, statistical analysis of the prototype's behaviour is therefore impossible.

Chapter 6

Conclusions and Future Work

This chapter reflects on the results presented in Chapter 5. Reflections both in what the results actually mean, and also what can be improved upon in future work.

Section 6.1 concludes the thesis and describes the outcome. Section 6.2 states what should be done to build upon this project. Additionally, Section 6.3 presents reflections regarding the thesis project itself. Finally, Section 6.4 discusses concerns regarding both the intricacies of the proposed solution and what value this solution actually could offer to others.

6.1 Conclusions

A proof of concept prototype was implemented and evaluated. The content creator client part of the prototype parses sources data from a smartphone camera and creates hashes of the video sequences. The hashes are consequently in real-time stored by a smart contract in a blockchain ledger for safekeeping and verification. A web client together with a blockchain will enable an end user to be able to prove that data uploaded through the prototype has or has not been tampered with. The Web verification client interacts with a smart contract. The smart contract is set up to handle requests from both the Android device client, and the web verification client. The smart contract is the only entity interacting with the blockchain itself.

The prototype at the current stage would not be a viable solution that could be trusted to keep the integrity of a video. Given that the signing process is missing, there would be no way in determining the order of the hashes corresponding to which frame(s). The signing process proposed in Section 6.2 would enable the signing of each hash, and also enable the verification side to be able to more easily identify where the hashes corresponds to in the video itself.

The prototype demonstrates much of the functionality that was desired, hence this Master's thesis have achieved part of its major goal by designing and implementing a working proof of concept prototype of the proposed system.

Section 3.5.2 describes the limitations of the Android platform. The prototype excludes the ability to inject and sign the frame(s)/GOP(s) used for hashing. As mentioned, this

is one of the fundamental limitations in the API used for developing the prototype. Furthermore, hashing GOPs rather than frames was implemented but not used in the evaluation framework of the testing.

As discussed in Section 2.1 there are mainly two ways in which the video may be divided and later hashed: frame by frame or in data chunks. The natural way from the mobile client point of view is to hash the frames individually since the data buffers in the Android phone handles the buffers on a frame by frame basis. However, building from this it is fairly straight forward to either group a number of frames together before being hashed or e.g hash a concatenated buffer with data from in between two I-frames. This particular method was implemented but not used in the testing due to the GOPs being less demanding for the system; and the evaluation framework was setup in order to stress test the prototype. The method of hashing on frame basis is further strengthened by the accuracy of the frame identification on the verification client side. As mentioned earlier the over all matching rate is around 98.1%, see Section 5.1, we have only analyzed the bit stream in order to identify the starting point for I-frames, and P-frames. Also, we are convinced this number could reach 100% without major difficulty, the reason being that there might be additional information sent that do not contain frame information. Additionally, the prototype generates hashes from H.264 buffer data in the Android device, whereas the hashes are recreated in the web client using mp4 format (i.e. the H.264 video is not demultiplexed from the .mp4 container before being hashed). The differences in header information in the H.264 to header information in .mp4 files have been identified, but not the other differences that the container might introduce when solely multiplexing video footage.

The Android device does not handle the video data as a long fixed length byte array which means that a method creating the array needed would have to be implemented. This would cost both memory and processing power. On the side of the verification client there would be a problem to know where to divide and hash the data into chunks. Frames are natural places to cut a video and they are also easy to identify. In order to create the chunks a method is needed to identify the pieces to be created. This method would also have to be implemented. However, a benefit of this method is the flexibility of the chunk sizes. It would be easy to set a specific chunk size and optimize the size to suit the rest of the system. Such optimization has not been investigated by this Master's thesis.

The RESTful server is able to handle HTTP POST requests over the wireless local area network coming from an Android device. Deploying the servers and the system in a real scenario would require more extensive work for handling requests that are sent over the internet. Furthermore, the prototype assumes that the system has not been developed with a lossless package stream in mind, certain existing live-streaming protocols can only work under lossy condition, thus losing information due to noisy transmission channels. Hence introducing false negatives to the blockchain verification. In order to facilitate a lossy channel for streaming, the hash function would need to be changed in order to mitigate the problem of recalculating the same hashes from lossy media. One of the main purposes in hash-functions is to avoid different data generating the same hashes i.e. avoiding collisions. A more robust solution would need to address the problem by either sending video data losslessly, or by introducing a method for identifying similar videos that have been corrupted by perturbation in the video transmission channel.

6.2 Future Work

All discrete parts of the prototype could use upgrades, and improvements in order to make the solution faster and more robust. Given that the developed prototype is a "minimum working solution", developing a future product based on this initial prototype would require lots of work.

The implementation within this project could be modified to add additional functionality to the system. First and foremost, the hashes need to be signed by introducing meta data in the transmitted video stream that tells the system which frames belongs to which hashes. If the hashes remain the same, but the frames have been scrambled (i.e., stored in a different order than i the original video file), the integrity of the video will still be intact. Injecting the meta data in the container (mp4-file) for signing hashes should be possible in the latest developer preview of Android 8.0, however it is yet to be implemented. Furthermore, additional meta data from the device may be hashed together with the video content to increase the level of trust in the video. This meta data could include geolocation, camera and device settings, account associated signatures (e.g. asymmetrical cryptography), firmware of the phone, audio from the recording, etc. All of this additional meta data hampers the possibility to tampering with the media content or the device.

The smart contract currently searches through the entire blockchain for hashes to match. Unfortunately, this is a tedious process when one has to use a specific date. Moreover, the request returns all of the hashes for the specified date. A more robust solution would be more automated and would not require input by the end-user and the time stamp of the transaction should be generated from the smart contract. The search function itself and the return function of the smart contract should each be optimized. The optimization of these would most likely reduce the time needed by the blockchain call function during the verification by the client and by extension speed up the entire verification process.

The algorithm for matching hashes from a target video with the blockchain has not been optimized for usage in a working system. A search algorithm using hash tables could be implemented in the web client in order to speed up the searching process. With a worst case complexity of $\mathcal{O}(N)$ the hash table will not make the search process linear in both the case with a non-matching video and with a video that finds matches.

The RESTful server is currently able to handle around 20 transaction per second which is likely to be a bottle neck if many users were to use the service at the same time. To scale the solution other server solutions be looked at, especially hardware-wise since the hardware is the biggest limiting factor at the current implementation.

The communication between the device and the RESTful-server, and between the web verification client and the RESTful-server is currently done using the HTTP protocol. In order to ensure security for transactions should the connection be done using the HTTPS protocol and SSL, this to ensure the communication is encrypted and that communication can be trusted. This is very important for future big scale applications when the communication is done over internet.

The blockchain itself only serves the web client with hashes. Some optimizations

to be looked at could be to include data structures to query the blockchain etc. The limitations of the blockchain are rather the hardware that it runs on top of.

The Android device employs and sends HTTP Posts, enabling the RESTful-server to also be run on the same or another computer. Another approach could instead be to run a Cumulus client on the Android device, thus omitting the need for a central RESTful-server which currently needs to handle all incoming HTTP(S)-requests from all nodes (devices) that wish to interact with the smart contract. However it is of great importance to secure the communication between the devices.

The accuracy of the matching of hashes needs to be increased. The prototype has an average matching rate of around 98% which is sufficient for this Master's thesis project but not for a product. The reason for the discrepancy in accuracy needs to be understood and dealt with. An accuracy of 100% should be an achievable end goal.

The verification client is at the moment able to handle whole videos. The next step would be to add the functionality to handle live video streams such as those that can be found on, for example Youtube, or Facebook. In this setting the user never really downloads the entire video but rather only access parts of it at a given time.

The current prototype utilizes hashes as a digital fingerprint of the data and it is these hashes which are compared to each other in order to identify video segments. This is a very binary way of identifying the video and the slightest change in the videos on any segment would completely change these hashes. Small changes such as: a slight color difference, the addition of a watermark, or a format change would completely alter the input data to the hash function and make the video completely unidentifiable by the current implementation. This is true even though it would be very easy to identify with the naked eye. Other methods should therefore be investigated as an alternative to hashing or possibly in conjunction with the hashing techniques.

In regards to the actual CODEC used, this solution only looked at H.264/.mp4 format/container. A real system should be compatible with more than just one standard format and container, and be more agnostic with respect to the CODEC.

6.3 Reflections Upon the Achievements of the Prototype

The parts lacking implementation in the prototype are the actual signing process of the hashes to be transmitted to the blockchain. This has been omitted due to the limitations in the Android 5.0 MediaMuxer class. Android 8.0 O Developer preview includes functionality such that the signing process can be included when muxing audio/video into the mp4 container, with additional meta data multiplexed in the form of string values. The proposed solution cannot, as previously stated in Section 3.5.2 distinguish the order of the buffers corresponding to each frame. Unfortunately, this limits the usage of the prototype solution. However, due to time constraints and not having access to an Android device with the latest version of the software, this solution was unfeasible at the time of this thesis project. Multiplexing further meta data is also something that should be evaluated and investigated. Applying meta data in the form of signatures connected with identities, location meta data, etc. is something that we

filed as an Intellectual Property Rights disclosure within Ericsson. The Intellectual Property Rights disclosure has been filed under the Patent Cooperation Treaty, and filed to the European Patent Office.

Other parts of the project that were not achieved are the security aspects of the communication between the different nodes of the prototype.

6.4 Required Reflections

The purpose of the prototype is to give the users of media platforms a service to use in order to verify the integrity of the video media that they are viewing. The information learned by implementing and evaluating this prototype could if realized in a product provide a base on which a user could decide whether a video is trustworthy or not. For good or evil, the prototype is essentially a system which verifies the integrity of certain aspect of the media content, specifically time and evidence of content tampering, and should be used in conjunction with other systems and/or aspects to establish trust in a given video. Additionally, the usage of this service also requires the use of common sense. A user should not place all of their trust in a single system, not even in this case. A system may suffer from system failure, manipulations, flaws not obvious to the user, or just bad intent. The prototype created in this Master's thesis is subject to all of these risks and may give a false sense of security to the user. It should therefore always be clearly stated **what** the prototype is actually verifying, thus giving the user a fair chance to make their own assessment without adding any bias.

The prototype may in the future be able to include identities coupled to specific media content. This is very useful in terms of verifying the integrity of the media but it may also pose a risk to personal integrity which is something which should not be taken lightly. The same problem may occur when using positioning together with time stamps. All of these additional features that may pose a threat or compromise to the personal integrity of the end-user should always be at the behest of the user.

The prototype of this Master's thesis could if developed into a product provide the consumers of online media content with a tool to verify the integrity of online content. The media content consumer does no longer have to rely upon the social media platform provider to verify the integrity of the media content found on the platform, the usage of such a product will thereby give power back to the content consumer. This power shift is an important social aspect of this Master's thesis. The process of verifying the content is triggered by the content consumer but otherwise should be highly automated. Aside from maintenance work and system updates, the verification process requires little or no human interaction. The automated nature of the verification process is a strong economic incentive for the provider of the online media content platform since this product could perform a service much cheaper and more efficient than trying to verify videos by using a human operator. However, there are some costs associated with such a service: namely storage, computational costs, and network traffic. These are not only monetary costs, but such a product also has an environmental footprint in terms of the energy consumption, manufacturing of the components, and recycling. These are the environmental aspects identified with this Master's thesis project.

Appendix A

Background Processes Running During Testing

The processes are listed in descending order of RAM usage for the Android Device, and in descending order of processor usage for the Macbook.

A.1 Android Device

- Settings: 1 process and 0 services
- CM Logger: 1 process and 2 services
- com.qualcomm.qcrilmsgtunnel: 1 process and 1 service
- Google Play services: 1 process and 12 services
- Google: 1 process and 1 service
- AudioFX: 1 process and 1 service
- SwiftKey: Keyboard: 1 process and 1 service
- Black Hole: 1 process and 1 service
- Theme Chooser: 1 process and 1 service

A.2 Macbook

- kernel_task
- WindowServer
- hidd
- sysmond
- Google Chrome Helper
- Google Chrome Helper

- SophosSXLD
- SystemUIServer
- SophosScanD
- launchd
- opendirectoryd
- Microsoft Excel
- syslogd
- SophosAntiVirus
- notifyd
- SafariCloudHistoryPushAgent
- SophosWebIntelligence
- gamecontrollerd
- UserEventAgent
- Google Chrome Helper
- SophosUIServer
- SophosServiceManager
- Google Chrome
- Google Chrome
- loginwindow
- UserEventAgent
- mDNSResponder
- usbd
- SpotifyWebHelper
- powerd
- cfprefsd
- Google Chrome Helper
- Google Chrome Helper
- ntpd
- watchdogd
- mtmfs

- cloudd
- awdd
- mapspushd
- symptomsd
- findmydeviced
- sua-1702150-0-InfiniteMediaAcceleration_SUA
- mdworker
- com.apple.spotlight.IndexAgent
- SophosAutoUpdate
- ctkd
- USBAgent
- usernoted
- CalendarAgent
- distnoted
- aslmanager
- WiFiProxy
- autofsd
- Spotlight
- Finder
- periodic-wrapper
- AssetCacheLocatorService
- syspolicyd
- apsd
- mdworker
- systemsoundserverd
- Citrix Service Record Application
- CallHistorySyncHelper
- coresymbolicationd
- diskarbitrationd
- Microsoft Update Assistant

- useractivityd
- AirPlayUIAgent
- secinitd
- lsd
- printtool
- nsurlsessiond
- airportd
- IMDPersistenceAgent
- icdd
- identityservicesd
- storeaccountd
- smd
- kdc
- configd
- BezelUIServer
- sharingd
- distnoted
- mdflagwriter
- mdworker
- com.apple.AddressBook.ContactsAccountsService
- com.apple.ctkpcscd
- amfid
- SpotlightNetHelper
- installd
- cfprefsd
- revisiond
- com.apple.CloudPhotosConfiguration
- storedownloadd
- racoon
- pkd

- LaterAgent
- akd
- blued
- fseventsd
- com.apple.audio.SandboxHelper
- FolderActionsDispatcher
- com.apple.InputMethodKit.TextReplacementService
- iconservicesagent
- IMRemoteURLConnectionAgent
- nsurlsessiond
- SocialPushAgent
- CommCenter
- system_installd
- mtmd
- coreservicesd
- DiskUnmountWatcher
- nsurlstoraged
- pboard
- sharedfilelistd
- mds_stores
- appleeventsd
- com.apple.AddressBook.InternetAccountsBridge
- suhelperd
- com.apple.PerformanceAnalysis.animationperfd
- IDSKeychainSyncingProxy
- sharedfilelistd
- securityd
- mdflagwriter
- askpermissiond
- wdhelper

- SubmitDiagInfo
- accountsd
- ScopedBookmarkAgent
- corestoraged
- VTDecoderXPCService
- mds
- TMCacheDelete
- AppleSpell
- iTunes Helper
- AppleIDAuthAgent
- WirelessRadioManagerd
- nbagent
- com.apple.speech.speechsynthesisd
- storeinstalld
- networkd_privileged
- kextd
- netbiosd
- TISwitcher
- nsurlstoraged
- wirelessproxd
- photolibraryd
- com.apple.CommerceKit.TransactionService
- tccd
- diagnostics_agent
- CVMServer
- distnoted
- iconservicesd
- bird
- IMRemoteURLConnectionAgent
- gamed

- com.apple.AmbientDisplayAgent
- backupd-helper
- Google Chrome Helper
- KernelEventAgent
- com.apple.CommerceKit.TransactionService
- Notiscenter
- crashpad_handler
- DataDetectorsDynamicData
- systemstatsd
- softwareupdated
- CloudKeychainProxy
- usbmuxd
- mdworker
- Citrix Receiver Authentication
- imklaunchagent
- thermald
- coreduetd
- spindump
- com.apple.hiservices-xpcservice
- AppleCameraAssistant
- com.apple.Safari.History
- cloudpaird
- secd
- EscrowSecurityAlert
- com.apple.audio.DriverHelper
- sandboxd
- Google Chrome Helper
- warmd
- rtcreportingd
- deleted

- iconservicesagent
- securityd_service
- com.apple.CodeSigningHelper
- AlertNotificationService
- Bilder Agent
- CrashReporterSupportHelper
- CallHistoryPluginHelper
- Keychain Circle Notification
- tccd
- Google Chrome Helper
- storelegacy
- networkd
- CalNCService
- storeaccountd
- Wi-Fi
- Textredigerare
- com.apple.ifdreader
- secinitd
- SophosCleanD
- Google Chrome Helper
- diagnosticd
- nbstated
- logind
- recentsd
- filecoordinationd
- PlexHelper
- com.apple.AccountPolicyHelper
- lsd
- authd
- reversetemplated

- com.apple.geod
- spindump_agent
- tccd
- CoreServicesUIAgent
- launchservicesd
- mdworker
- mdworker
- com.apple.CommerceKit.TransactionService
- soagent
- fmfd
- com.apple.cmio.registerassistantservice
- com.apple.dock.extra
- imagent
- fontd
- applessdstatistics
- nehelper
- coreaudiod
- Google Chrome Helper
- AirPlayXPCHelper
- suggestd
- SophosConfigD
- diskmanagementd
- storeinappd
- storeassetd
- pbs
- Logitech Control Center Daemon
- awacsd
- locationd
- swcd
- CMFSyncAgent

- taskgated
- akd
- Dock
- mdworker
- com.apple.activitymonitor.helper
- ocspd
- com.apple.iCloudHelper
- com.apple.appkit.xpc.openAndSavePanelService
- ViewBridgeAuxiliary
- Dropbox Finder Integration

Bibliography

[1] P. Debraj Ghosh. (5 April 2016) The Byzantine General's Scenario. Downloaded: 2017-02-20. [Online]. Available: https://cdn-images-1.medium.com/max/1600/0*-xCD-El4LZ48dji1.png

[2] J. Smith, "Here's why consumers are increasingly turning to streaming media devices to view content," June 16 2016, BI Intelligence, Business Insider Nordic, Accessed: 2017-05-10. [Online]. Available: https://www.theguardian.com/global-development-professionals-network/2017/jan/17/blockchain-digital-technology-development-money

[3] S. Goel, A. Anderson, J. Hofman, and D. J. Watts, "The structural virality of online diffusion," *Management Science*, vol. 62, no. 1, pp. 180–196, 2016. doi: 10.1287/mnsc.2015.2158. [Online]. Available: http://dx.doi.org/10.1287/mnsc.2015.2158

[4] M. Morshed, "Voice over IP and Lawful Intercept : Good cop/Bad cop," Master's thesis, KTH, School of Information and Communication Technology (ICT), 2010, urn:nbn:se:kth:diva-24260, oai:DiVA.org:kth-24260, diva2:346180.

[5] G. J. Sullivan, P. N. Topiwala, and A. Luthra, "The H.264/AVC advanced video coding standard: overview and introduction to the fidelity range extensions," *Proc. SPIE*, vol. 5558, pp. 454–474, 2004. doi: 10.1117/12.564457. [Online]. Available: http://dx.doi.org/10.1117/12.564457

[6] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec 2012. doi: 10.1109/TCSVT.2012.2221191

[7] P. Wilkins, Y. Xu, L. Quillio, J. Bankoski, J. Salonen, and J. Koleszar, "VP8 Data Format and Decoding Guide," RFC 6386, nov 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6386.txt

[8] Adrian Grange, Peter de Rivaz, and Jonathan Hunt, "VP9 Bitstream & Decoding Process Specification," Version 0.6, March 2016, https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.6-20160331-draft.pdf.

[9] R. Finlayson, "A More Loss-Tolerant RTP Payload Format for MP3 Audio," Internet Request for Comments, vol. RFC 5219 (Proposed Standard), Feb. 2008 [Online]. Available: http://www.fc-editor.org/rfc/rfc5219.txt, 2001.

[10] "Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC)," International Organization for Standardization, Geneva, CH, Standard, Dec. 2006, ISO/IEC 13818-7:2006/Cor 2:2010.

[11] JM. Valin, K. Vos, and T. Terriberry, "Definition of the Opus Audio Codec," RFC 6716, September 2012, 10.17487/RFC6716. [Online]. Available: http://www.rfc-editor.org/info/rfc6716

[12] E. Fleischman, "WAVE and AVI Codec Registries," RFC 2361, June 1998, 10.17487/RFC2361. [Online]. Available: http://www.rfc-editor.org/info/rfc2361

[13] Apple Computer, Inc, "Quicktime File Format," March 2001. [Online]. Available: https://developer.apple.com/standards/qtff-2001.pdf

[14] A. Noé, "Matroska file format (under construction!)," Matroska (non-profit org), pp. 1–48, Jan 2009. [Online]. Available: https://www.matroska.org/files/matroska.pdf

[15] Apple Inc., "MPEG-2 Reference Information," Published: http://documentation.apple.com/en/compressor/usermanual/index.html#chapter=18%26section=5%26tasks=true, accessed: 2017-05-09.

[16] K. Purvis, "Blockchain: what is it and what does it mean for development?" January 17 2017, accessed: 2017-02-10. [Online]. Available: https://www.theguardian.com/global-development-professionals-network/2017/jan/17/blockchain-digital-technology-development-money

[17] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, 1991. doi: 10.1007/BF00196791. [Online]. Available: http://dx.doi.org/10.1007/BF00196791

[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, 2008," *URL: http://www. bitcoin. org/bitcoin. pdf*, 2012.

[19] "Hyperledger," https://www.hyperledger.org/about/members, a Linux Foundation Collaborative Project. Accessed: 2017-02-10.

[20] "About r3," http://www.r3cev.com/about, accessed: 2017-02-10.

[21] Secretariat of the Internet Governance Forum (IGF), "Dynamic coalition on blockchain technologies," UN-DESA, http://www.intgovforum.org/cms/130-dynamic-coalitions/2249-dynamic-coalition-on-blockchain-technologies, accessed: 2017-02-10.

[22] National Institute of Standards and Technology, "NIST brief comments on recent cryptanalytic attacks on secure hashing functions and the continued security provided by SHA-1," http://csrc.nist.gov/groups/ST/toolkit/documents/shs/hash_standards_comments.pdf, accessed: 2017-02-21.

[23] Google Security PR, "Announcing the first SHA1 collision," https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html?m=1, 2017.

[24] H. C. Williams, Ed., *Advances in cryptology–CRYPTO '85: proceedings*, ser. Lecture notes in computer science. Berlin ; New York: Springer-Verlag, 1986, no. 218. ISBN 978-0-387-16463-2

[25] N. P. Smart, *Cryptography: an introduction*. McGraw-Hill New York, 2003, vol. 5. ISBN 9780077099879

[26] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4/3, pp. 382–401, July 1982. [Online]. Available: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/

[27] M. Vukolić, *The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication*. Cham: Springer International Publishing, 2016, pp. 112–125. ISBN 978-3-319-39028-4. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-39028-4_9

[28] J. Stark, "Making sense of blockchain smart contracts," CoinDesk, Published on June 4, 2016, http://www.coindesk.com/making-sense-smart-contracts/, accessed: 2017-02-10.

[29] "Smart contracts: The blockchain technology that will replace lawyers," http://blockgeeks.com/guides/smart-contracts/, accessed: 2017-02-10.

[30] Blockchain Luxembourg S.A., "Market capatilization," https://blockchain.info/sv/charts/market-cap?timespan=60days, accessed: 2017-02-28.

[31] R. Cohen, "Global bitcoin computing power now 256 times faster than top 500 supercomputers, combined!" http://www.forbes.com/sites/reuvencohen/2013/11/28/global-bitcoin-computing-power-now-256-times-faster-than-top-500-supercomputers-combined/#7652859928b, Forbes, 28 November 2013, accessed: 2017-02-10.

[32] Blockchain Luxembourg S.A., "Transaction rate," https://blockchain.info/sv/charts/transactions-per-second, accessed: 2017-02-21.

[33] A. Zohar, "Bitcoin: Under the hood," *Commun. ACM*, vol. 58, no. 9, pp. 104–113, Aug. 2015. doi: 10.1145/2701411. [Online]. Available: http://doi.acm.org/10.1145/2701411

[34] F. L. David Voell *et al.*, "Hyperledger whitepaper," Published: https://wiki.hyperledger.org/groups/whitepaper/whitepaper-wg, August 3, 2016, http://www.the-blockchain.com/docs/Hyperledger\%20Whitepaper.pdf, accessed: 2017-02-10.

[35] B. Vitalik *et al.*, "Ethereum whitepaper," Published: https://github.com/ethereum/wiki/wiki/White-Paper, September 1, 2014, accessed: 2017-02-20.

[36] P. Bajpai, "Bitcoin vs ethereum: Driven by different purposes," accessed: 2017-02-10. [Online]. Available: http://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp

[37] "Ethereum," [Online]. Available: https://www.ethereum.org/, accessed: 2017-02-10.

[38] Gavin Wood, "Ethereum: A secure decentralised generalised transaction ledger," Published: https://ethereum.github.io/yellowpaper/paper.pdf, Mars 12, 2017, http://www.the-blockchain.com/docs/Hyperledger\%20Whitepaper.pdf, accessed: 2017-04-19.

[39] Ethan Buchman and Jae Kwon, "Docker," https://github.com/tendermint/tendermint/blob/master/DOCKER/README.md, accessed: 2017-02-10.

[40] Tendermint, "Introduction to tendermint," [Online]. Available: https://tendermint.com/intro, accessed: 2017-02-10.

[41] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Master's thesis, University of Guelph, http://hdl.handle.net/10214/9769, 2016.

[42] Z. Ramsay, "On Eris and Tendermint: Application and Consensus," Monax Blog, https://monax.io/2016/03/02/eris-and-tendermint/, accessed: 2017-02-10.

[43] Ethan Buchman and Jae Kwon, "Github Wiki: Introduction," https://github.com/tendermint/tendermint/wiki/Introduction, accessed: 2017-02-28.

[44] Tendermint, "Tendermint vs. other software," https://tendermint.com/intro/tendermint-vs, accessed: 2017-02-10.

[45] Monax, "Platform," https://monax.io/platform/, accessed: 2017-02-10.

[46] M. Bellare and P. Rogaway, *The Exact Security of Digital Signatures-How to Sign with RSA and Rabin*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 399–416. ISBN 978-3-540-68339-1. [Online]. Available: http://dx.doi.org/10.1007/3-540-68339-9_34

[47] Android Open Source Project, "Camera2 API," Published: https://developer.android.com/reference/android/hardware/camera2/package-summary.html, accessed: 2017-05-08.

[48] ——, "MediaCodec," Published: https://developer.android.com/reference/android/media/MediaCodec.html, accessed: 2017-05-08.

[49] ——, "Mediamuxer," Published: https://developer.android.com/reference/android/media/MediaMuxer.html, accessed: 2017-05-08.

[50] ——, "Messagedigest," Published: https://developer.android.com/reference/java/security/MessageDigest.html, accessed: 2017-07-06.

[51] ——, "Application Fundamentals," Published: https://developer.android.com/guide/components/fundamentals.html, accessed: 2017-04-21.

[52] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[53] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University Of California, Irvine, 2000, http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf.

[54] Ludovico Fischer, "A Beginner's Guide to HTTP and REST," Published: https://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340, Jan 2009, accessed: 2017-06-03.

[55] S. Josefsson, "The Base16, Base32, and Base64 Data Encodingsl," *Request for Comments*, Oct. 2006. [Online]. Available: https://tools.ietf.org/html/rfc4648

[56] E. Rescorla and A. Schiffman, "The Secure HyperText Transfer Protocol," *Request for Comments*, vol. 2260, Aug. 1999. [Online]. Available: https://tools.ietf.org/html/rfc2660

[57] B. Gipp, J. Kosti, and C. Breitinger, "Securing video integrity using decentralized trusted timestamping on the bitcoin blockchain," *MCIS 2016 Proceedings. 51.*, 2016. doi: http://aisel.aisnet.org/mcis2016/51

[58] B. Gipp, N. Meuschke, and A. Gernandt, "Decentralized trusted timestamping using the crypto currency bitcoin," *CoRR*, vol. abs/1502.04015, 2015. [Online]. Available: http://arxiv.org/abs/1502.04015

[59] J. Clark and A. Essex, *CommitCoin: Carbon Dating Commitments with Bitcoin*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 390–398. ISBN 978-3-642-32946-3. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32946-3_28

[60] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)," *Internet Request for Comments*, vol. RFC 3161 (Proposed Standard), Aug. 2001. [Online]. Available: https://rfc-editor.org/rfc/rfc3161.txt

[61] R. Poisel and S. Tjoa, "Forensics investigations of multimedia data: A review of the state-of-the-art," in *2011 Sixth International Conference on IT Security Incident Management and IT Forensics*, May 2011. doi: 10.1109/IMF.2011.14 pp. 48–61.

[62] I. Echizen, S. Singh, T. Yamada, K. Tanimoto, S. Tezuka, and B. Huet, "Integrity verification system for video content by using digital watermarking," in *2006 International Conference on Service Systems and Service Management*, vol. 2, Oct 2006. doi: 10.1109/ICSSSM.2006.320788. ISSN 2161-1890 pp. 1619–1624.

[63] "Assureon archive storage data sheet," [Online]. Available: https://www.nexsan.com/wp-content/uploads/datasheets/Assureon-DS-v2.pdf, Nexsan, 900 E. Hamilton Ave., Suite 230, Campbell, CA 95008, Revision 22 November 2016, 2 pages, accessed: 2017-02-14.

[64] Göran Almgren, Mats Stengård, and Hans Almgren, "Enigio AB," [Online]. Available: https://enigio.com/, accessed: 2017-02-24.

[65] Dimitri de Jonghe and Trent McConaghy, "SPOOL," [Online]. Available: https://github.com/ascribe/spool, accessed: 2017-02-24.

[66] S. Higgins, "Blockchain startup raises $2 million for intellectual property solution," [Online]. Available: http://www.coindesk.com/blockchain-startup-2-million-intellectual-property/, accessed: 2017-02-24.

TRITA-ICT-EX-2017:122