Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Hagen, Haakon Kristian

Authenticated Key Exchange

An analysis of low-cost protocols

Master's thesis in Communication Technology and Digital Security
Supervisor: Boyd, Colin
Co-supervisor: Millerjord, Lise
March 2023

NTNU
Norwegian University of
Science and Technology

Hagen, Haakon Kristian

Authenticated Key Exchange

An analysis of low-cost protocols

Master's thesis in Communication Technology and Digital Security
Supervisor: Boyd, Colin
Co-supervisor: Millerjord, Lise
March 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Norwegian University of
Science and Technology

Authenticated Key Exchange: An analysis of low-cost protocols

Hagen, Haakon Kristian

Submission date: March 2023
Main supervisor: Boyd, Colin, NTNU
Co-supervisor: Millerjord, Lise, NTNU

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

Title: Authenticated Key Exchange: An analysis of low-cost protocols

Student: Hagen, Haakon Kristian

Problem description:

The recent SAKE (Symmetric-key Authenticated Key Exchange) protocol is designed to be a lightweight key-exchange protocol with strong security, specifically including forward secrecy. The project will explore the efficiency and security properties of the SAKE protocol, with comparisons to other proposed and established protocols. The project will estimate the power usage of SAKE and make comparisons with protocols intended for similar use. Other metrics that may also be used for comparison are: communications efficiency, computational efficiency and security properties.

Approved on: 2022-04-01

Main supervisor: Boyd, Colin, NTNU

Co-supervisor: Millerjord, Lise, NTNU

Abstract

In the Internet of Things there is a focus on efficient and low cost protocols. When it comes to key exchange protocols there is a need to balance security, complexity and cost. Because of this need, different protocols can often be specialized towards specific use cases. One such protocol is the SAKE protocol. This protocol is an attempt at creating an authenticated key exchange protocol that relies on symmetric encryption methods. The SAKE protocol is intended specifically for use in low-energy environments.

In this paper we make an analysis of the energy cost of the SAKE protocol. The cost of the EDHOC protocol is also analysed to provide a comparison. The result of the analysis and comparison is that SAKE is much more energy efficient than EDHOC. We also determine that SAKE has a good potential for future use in situations where pre-shared keys are available. EDHOC on the other hand seems well suited to use in situations where public keys are desirable for key management.

Sammendrag

I Internet of Things feltet er det et fokus på kosteffektive protokoller. Når det gjelder nøkkelutvekslingsprotokoller så er det et behov for å balansere sikkerhet, kompleksistet og kostnad. Forskjellige protokoller kan derfor bli spesialisert for spesifike bruksområder. SAKE-protokollen er et forsøk på å lage en autentisert nøkkelutvekslingsprotokoll som baserer seg på symmetriske krypteringsmetoder. Protokollen er designet med bruk i lav-energi situasjoner i tankene.

I denne oppgaven gjør vi en analyse av energikostnadene til SAKE-protokollen. Vi analyserer også kostnadene til EDHOC-protokollen for å gi en sammenligning. Analysen og sammenligningen kommer fram til et resultat som viser at SAKE er mye mer energieffektiv enn EDHOC og at SAKE har et godt potensiale for fremtidig bruk i situasjoner hvor forhåndsdelte nøkler er tilgjengelige. Vi kommer også fram til at EDHOC protokollen har et godt potensiale i bruksområder hvor offentlig delte nøkler er foretrukket.

Contents

List of Tables

List of Acronyms

AEAD Authenticated Encryption with Associated Data.

AES Advanced Encryption Standard.

AKE Authenticated Key Exchange.

CBOR Concise Binary Object Representation.

CoAP The Constrained Application Protocol.

DSA Digital Signature Algorithm.

ECDH Eliptic-Curve Diffie-Hellman.

ECDSA Elliptic Curve Digital Signature Algorithm.

EDHOC Ephemeral Diffie-Hellman Over COSE.

HKDF HMAC-based Extract-and-Expand Key Derivation Function.

HMAC keyed-Hash Message Authentication Code.

IETF Internet Engineering Task Force.

IoT Internet Of Things.

ISO International Organization for Standardization.

KCI Key Compromise Impersonation.

KDF Key Derivation Function.

MAC Message Authentication Code.

NIST National Institute of Standards and Technology.

PFS Perfect Forward Secrecy.

PRK Pseudo Random Key.

PSK Pre-Shared Key.

RSA Rivest–Shamir–Adleman.

SAKE Symmetric-Key Authenticated Key Exchange.

SAKE-AM SAKE *in Aggressive Mode*.

TLS Transport Layer Security.

WSN Wireless Sensor Network.

Chapter 1
Introduction

The intention of this thesis is to take a closer look at the efficiency of the SAKE protocol and see how it compares to a different protocol intended to be used for similar cases. The following chapter presents an introduction to the project and the motivation for exploring the topic. It also presents the research questions and gives an explanation for the structure of the thesis.

1.1 Motivation

The Internet Of Things (IoT) is a broad term used to describe physical objects embedded with sensors, processing capabilities and communications technologies [WF15]. Many of these objects exist in resource constrained environments[Li17a], requiring them to complete their tasks in a way that consumes as little energy as possible. To do so, it requires protocols that are highly efficient, requiring as few and simple operations as possible.

With the requirement of being lightweight, security protocols in constrained environments often need to balance between security features and efficiency [Li17b]. This is due to the fact that robust and secure protocols often come at a high cost of resources. The SAKE protocol is a suggested protocol that is intended for constrained environments and attempts to solve some of the issues of a protocol intended for this particular environment.

Specifically, SAKE is an Authenticated Key Exchange (AKE) protocol. That is, a protocol whose goal is to establish a secret key that is shared between two parties, while authenticating each party's identity to the other [DvOW92]. SAKE attempts to do so while providing strong security properties, but in a manner that makes it less resource intensive.

The field of IoT has a great use for protocols with the properties SAKE attempts to achieve. As the uses for embedded sensors becomes more apparent and widespread,

the need for protocols that are tailored for the emerging environments becomes increasingly important. It is therefore highly useful to determine whether SAKE does achieves its goal of being light weight whilst providing strong security properties.

1.2 Research question

To investigate the usefulness of SAKE the project poses the following research questions.

- Is the energy usage of SAKE efficient?
- How does SAKE compare to other protocols intended for similar use?

1.3 Structure

The study has an overall structure with 6 chapters, consisting of an introduction, theory, protocols, methodology, results and a discussion of the results and finally a conclusion. A brief summary of the sections and its purpose is described below.

The first chapter provides the background of the research and why it is meaningful to carry out. It will justify the importance of investigating how the SAKE performs and why we make a comparison with a different protocol. The second chapter gives a brief overview of important information and concepts that are necessary to understand this project. In chapter three the two protocols that the project focuses on are described in detail and a high level comparison is made. Chapter four explains the method used to find the results that are presented in chapter five. Finally, the last chapter will discuss the findings, summarise the research and answer the research question. Additionally, implications for practice and further research in the area is presented, as well potential limitations of the research.

Chapter 2
Background

The aim of this chapter is to provide an overview of the theory required to understand the scope of this project. It will lay out an introduction to the field at hand, to the types of protocol in question and several key concepts.

2.1 Cryptography and Security Features

The field of cryptography is the science of transforming data, or information, in such a way that it becomes incomprehensible, maintains its authenticity and prevents alteration or unauthorized use [Shi07; Sta20]. There are many important methods and techniques that make up this field. The following section will give a brief overview of the relevant parts.

2.1.1 Encryption

One of the key parts of the field of cryptography is encryption. Encryption is the act of transforming information in some way that hides the original meaning, thereby preventing the use of the original information. We often call the original piece of data "plain text" and the transformed data "cipher text" [Shi07]. The reverse process of encryption, transforming unintelligible cipher text into understandable clear text, is called decryption [Shi07].

In the field of cryptography we have two main ways of encrypting messages, symmetric and asymmetric encryption. These two cryptographic systems have different advantages and disadvantages which will be explained in detail below.

2.1.2 Symmetric Cryptography

Symmetric cryptography, which is also known as private key or secret key cryptography, is a cryptographic method where a single shared secret is used to encrypt data between parties. The same key is used by both parties to both encrypt plaintext and

decrypt ciphertext [Sta20]. Symmetric encryption is important to explain in this project as it is the method of encryption that SAKE relies on.

The Advanced Encryption Standard (AES) is an example of a well accepted and much used cipher that utilizes symmetric encryption. The AES algorithm is a block cipher developed by the National Institute of Standards and Technology (NIST) in the US [Sta20]. When discussing cryptography the term "cipher" is used to describe some algorithm that is used both for encryption and decryption. Saying AES is a block cipher means that it uses a type of algorithm that separates plain text into segments of fixed size and then uses the same key to encrypt each segment [Shi07]. AES is a well known algorithm and is in widespread use. To the point that it has been included as a standard by the International Organization for Standardization (ISO) [ISO10].

2.1.3 Asymmetric Cryptography

Asymmetric cryptography, also known as public key cryptography, is a cryptographic method based on each party having a unique key pair, where one key is public and one is private [Shi07]. The public keys are shared openly between parties while each party has their own unique secret key. The key pairs are mathematically related and their generation is based on one-way functions, functions where input is easy to compute but inversion is computationally hard. One-way functions are also known as non-invertible functions.

The public keys are used to encrypt plaintext. The resulting ciphertext can then be decrypted using the related private key. By having the public key anyone can encrypt a message, but only the holder of the private key can decrypt. This allows for easy distribution of keys.

One of the earliest published examples of a public key method is the Diffie-Hellman key exchange method [Sta20]. Pictured in Figure 2.1, the Diffie-Hellman method relies on the difficulty of computing discrete logarithms. In the figure the values g and p are the public parameters in the exchange, where p is a prime number and g is a primitive root of p. The parties each choose a secret value, a and b, which are kept secret throughout the exchange. Alice calculates the public value A using the expression:

$$A = g^a \ mod \ p$$

And Bob calculates the public value B with the expression:

$$B = g^b \ mod \ p$$

Once the values A and B have been exchanged, both parties are able to calculate the shared secret K. Both parties now share the secret value K without ever having

disclosed their secret values a or b. While all values except a and b are public, they are necessary to compute the shared secret and to find them using the public values is computationally hard.

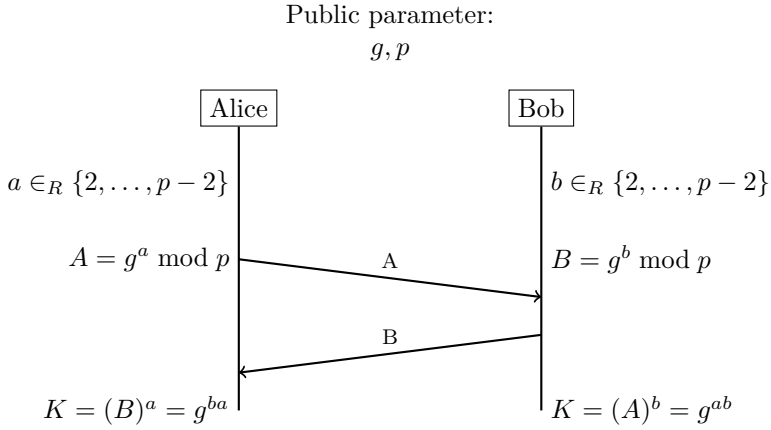


Figure 2.1: Diffie Hellman operation

Another example of a commonly used and well known system that uses asymmetric cryptography is the Rivest–Shamir–Adleman (RSA) cryptosystem. RSA is a block cipher that relies on the operations of modulo and exponentiation to create key pairs based on two large prime numbers. The security of the system relies on the difficulty of factoring the product of two large primes [Sta20]. RSA was one of the first published cryptosystems utilizing public key cryptography and has been one of the most implemented and accepted schemes of this type.

Asymmetric cryptography is important to this project as a counterpart to symmetric cryptography. It is widely used and accepted due to its strong security properties and scalability, and is the foundation of the security properties of the EDHOC protocol. To understand what the SAKE protocol is innovating from, one must understand the established schemes with their advantages and disadvantages.

2.1.4 Hash Functions

An important and often used tool in cryptography is the hash function. A hash function is a function that takes an input of variable length and maps this to a fixed length output [Shi07]. For the output of hash functions it is desirable to have unpredictability and uniformity. By unpredictability we mean that any change to an input string will give an unpredictable change in the output. And by uniformity we

mean that the output should be as evenly distributed as possible, over the possible output range.

When used in cryptography it is important for hash functions to have the properties listed below. These properties are what make hash functions a basic tool in cryptography and one of the building blocks of security protocols [Sta20].

- Ability to process input of any size.
- To always give an output of fixed length.
- The function is easy to compute for any input, and also quick to compute.
- The function is one-way. That is to say, if given the output it is computationally hard to find the input. This property is essential for providing forward secrecy, as old and deleted keys are hard to recover.
- For any input it is computationally hard to find another, different, input that would result in the same output.
- It is computationally hard to find any pair of unique inputs that would result in the same output.

Being used extensively in both SAKE and EDHOC, it is important to understand the capabilities and limitations of hash functions when analysing both protocols.

2.1.5 MACs

In the field of cryptography a Message Authentication Code (MAC) is a computed value that is used to authenticate a message [Sta20]. A MAC, or tag, is created using a function that takes a message and a secret key as input and gives a value of fixed length as output [Shi07]. The output of a MAC function will always be the same for a given message and key. This means that a receiver can verify a message by computing a MAC using a received message and a shared secret key, and then comparing the result with a MAC that came appended to the received message. If the MACs are the same then the receiver knows that the message came from someone who shares the secret key, thereby authenticating the message.

One way of creating MACs is by using hash functions. By basing a MAC function on a keyed hash function, a hash function that takes a key in addition to some other input, we get a MAC that inherits its strength from the hash function it is based on [Sta20]. The resulting function is called an keyed-Hash Message Authentication Code (HMAC) [GTDD08]. HMACs can use any iterated hash function and thereby vary its cryptographic strength by the strength of the used hash function [Shi07].

2.1.6 Key derivation functions

Another important type of function in the field of cryptography is the Key Derivation Function (KDF). A function that produces a key, or possibly several keys, deriving it from some form of base key and possibly other parameters [Kal00]. To derive the keys a KDF uses a pseudo-random function [lChe22]. A pseudo-random function being a function that through a deterministic computational process and the use of some form of input, gives an output of values that, according to specific statistical tests, appears random [Shi07]. It is common to use HMAC functions as a pseudo-random function [Zdz12].

As cryptography relies on using keys for encryption, decryption, keyed hash functions and more. It is of great relevance to understand the basic principles of KDFs and the use of their keys. The use of keys will be further explained in section 2.2.

Of particular interest to this project is the key derivation function HMAC-based Extract-and-Expand Key Derivation Function (HKDF). HKDF is a KDF based on HMAC functions and is composed of two main functions, HKDF-Extract and HKDF-Expand. The Extract function takes some key material and optionally a salt as input of a HMAC function to produce a Pseudo Random Key (PRK). The Expand function takes the PRK, an additional data field and a length specification as input. Using the PRK as a key and the additional data as a message, an HMAC function is called to create an output of specified length. These inputs are chained by using the output as the additional data the next time the HMAC function is called [KE10].

2.1.7 Digital Signatures

Another tool in the field of cryptography is the use of digital signatures. Digital signatures are schemes that create a value by using an algorithm and some data object as input for the algorithm. The resulting value is used to verify both integrity and origin of the data object [Shi07]. Digital signatures are similar to MACs, but offers some additional properties. The key property being the ability to uniquely identify as specific system as the signer.

RSA is an example of a cryptosystem that has the ability to add signatures to messages [Shi07]. It is a well adopted method of providing encryption with signatures, utilizing depending on the factoring of the product of two large prime numbers. As RSA is not a fast algorithm in comparison with symmetric encryption, it is often used to establish a shared secret which is then used for symmetric methods for further communication [Sta20].

An algorithm intended only for providing digital signatures is the Digital Signa-

ture Algorithm (DSA). A standard developed by NIST, DSA is based on modular exponentiation and the problem of discrete logarithms [Sta20]. The later versions of DSA includes a technique for digital signatures that relies on elliptic curve cryptography. This version is called the Elliptic Curve Digital Signature Algorithm (ECDSA). Utilizing elliptic curves ECDSA achieves the same level of security while using shorter key lengths than other algorithms would need [Sta20].

2.1.8 Post-Quantum Cryptography

A possible game-changer within the field of cryptography is the future of quantum computing. Quantum computing is the combination of computer science and quantum physics. This field is mainly theoretical as no functional system has been developed, but as explained below it has the possibility of greatly impacting cryptographic principles that are relied on today.

The theory of quantum computing is that it performs its calculations with the help of quantum physical principles. It relies on representing information as theoretical "qubits", also known as "quantum bits". A simplified explanation of qubits is that they are similar to classic bits in the way they represent information, but they have a behavior that follows the laws of quantum physics. This behavior gives qubits two unusual properties, entanglement and superposition.

The property of superposition comes from the behavior of a system following the laws of quantum mechanics. Such a system is only set to a specific state once it is measured. Until then the system remains in a superposition consisting of all the possible states. In quantum computing this property manifests itself in the ability of a qubit to exist in a superposition of both "0" and "1", only to collapse into one of the two when measured [Sta20; Hid21].

Entanglement is a particular case of superposition of two system where the measurement of one system is strongly correlated with the state of the second system. In quantum computing this gives qubits the ability to be linked in such a way that measuring one qubit will cause the second qubit to collapse into a state that can be known beforehand.

These two properties allow for a scaling of computational power far greater than conventional computing can, which leads to possible new threats to existing cryptographic methods.

One example of the threat quantum computing can bring is Shor's algorithm. As explained in section 2.1.7, RSA relies on the difficulty in factoring the product of two large prime numbers. Shor [Sho94] introduces a algorithm that can factorize the primes of any positive product in polynomial time. The algorithm is estimated to

need a few thousand qbits to break a 3072-bit RSA key. Also, the necessary amount of qbits to break a key scales linearly with the amount of digits in the key. This means that once functional quantum computers of some size are developed, systems that rely on RSA become highly vulnerable to attack. The algorithm also works for finding the discrete logarithm of elliptic curves, which means that the algorithm also threatens systems based on elliptic-curve cryptography [BL17; Sta20].

A second example of a quantum algorithm that can threaten established cryptographic methods is Grover's algorithm. Introduced in 1996 by Grover [Gro96], this algorithm can search an unordered list of size N in $\mathcal{O}(\sqrt{N})$ time. While not as impressive as the speed Shor's algorithm operates at, this is still an immense upgrade for the kind of algorithm that is used to perform brute force attacks on symmetric encryption and hash algorithms. Using Grover's algorithm, the security of an algorithm with key size n can be reduced to the level of a key size of $n/2$. As an example, this means that to maintain the security a 128-bit AES key provides currently, one would only have to double the key length from 128-bit to 256-bit. This, in combination with the discovery that searching algorithms at an exponential rate is impossible, presents the theory that current symmetric cryptography methods and hash functions can remain secure with minimal changes in a post quantum world [BL17; Sta20].

2.2 Key Exchange

2.2.1 AKE protocols

An important protocol among the many that make up modern digital communication are AKE protocols. These protocols are used to establish and exchange session keys between communicating parties, while also providing each party with authentication of the opposite party's identity [DvOW92]. The SAKE protocol is a suggested AKE protocol and as such it is important to understand the use of this type of protocol. When two parties want to communicate and keep their messages hidden from third parties, it is useful to encrypt the messages. To do so both parties must agree on some form of key for encryption. In addition it is useful to have some way of verifying that the messages one receives does indeed come from the expected party. This is what AKE protocols offer.

When discussing AKE protocols there are several important terms regarding types of keys that can be used. As many different types are mentioned and discussed throughout this project it is important to understand the difference.

Master key: also known as a key-derivation key, is a key that is used by some method to derive other keys. Master keys are long lasting and should be stored

securely as their disclosure can compromise derived keys [Sta20].

Long Term Key: Long term keys are meant for use over an extended period of time and maybe several sessions.

Session Key: A session key on the other hand is meant for use in only one session and not to be used again [Bar20b].

Ephemeral Key: One sub type of session keys is an ephemeral key. Ephemeral keys are for one time use only and never to be used again [Sta20; Bar20b].

2.2.2 Perfect Forward secrecy

In addition to easy key management, asymmetric cryptography has the advantage of easily providing perfect forward secrecy when used. Perfect forward secrecy, otherwise known as forward secrecy, is a security feature of key exchange protocols which guarantees that compromise of long-term secret keying material does not endanger the secrecy of keys that have been exchanged in previous runs [DvOW92]. That is to say, if a session key is compromised it will give access only to data encrypted using this key [HC98]. Meaning that each key must be derived from independent material. It also means that if a long term key is compromised this does not compromise previously used session keys. Perfect Forward Secrecy (PFS) is of interest to this project as the SAKE protocol is developed with the idea of providing forward secrecy using only symmetric cryptography. It is therefore important to understand what PFS does and how it is usually achieved.

A well known method of public key cryptography that can be used to provide forward secrecy is the Diffie-Hellman key exchange. However, on its own it does not provide authentication, nor forward secrecy. To provide authentication digital signatures are often added to the messages. To provide forward secrecy long-term keys are used for the digital signatures and ephemeral keys, that is to say temporary keys, are used for the Diffie-Hellman exchange.

A more updated version of Diffie-Hellman key exchange is the Eliptic-Curve Diffie-Hellman (ECDH) key agreement protocol. ECDH is a variant of the Diffie-Hellman scheme that is based on the mathematics of elliptic curves [Shi07]. This results in shorter key lengths and better computational efficiency than normal Diffie-Hellman. When applied with ephemeral keys, ECDH too can provide forward secrecy to an AKE protocol.

Another idea that can provide forward secrecy to a key exchange protocol is the concept of key evolution. Key evolution requires a protocol to evolve its master key or key derivation material between each derived session key. The term evolving in this

context means changing the original material in some way that is computationally hard to reverse, for instance by employing a hash function [ACF20].

One example of key evolution can be found in the Double Ratchet Algorithm, utilised by the app Signal [CCD+16]. In this algorithm the term "ratchet" is used to mean updating or evolving a key using some function. The algorithm intends to provide distinct message keys for all new messages by using key chains created using two distinctly separate ratchets. The first ratchet is based on the Diffie-Hellman key exchange and adds an ephemeral public key to each message. The second ratchet uses some KDF, for example a hash function, to update the symmetric key. Like SAKE this scheme also requires two parties to have a common secret before initialization.

A chain of root keys is instantiated using the shared secret. Each key being the input material for the next using a ratchet to derive a chain. For every root key the second ratchet is also applied to derive a chain of keys from which the message keys are derived. The second ratchet takes the root key as input and outputs two new values. One is a message key. The other is a chain key used as input to derive the next set of keys. Using deterministic functions, both parties can therefore produce the same series of keys. The algorithm operates asynchronously with each party using every other root key to encrypt messages before sending. That is to say a party A will use a key chain derived from root key k_1, until it receives at least one message from party B, where B has used keys from a chain derived from k_2. A will then send its next messages using root key k_3 as key chain material [BFG+22; PM16].

The scheme has a method for handling out of order messages. It includes in the header a message's number in its sending chain and the number of messages in the previous chain. The receiver therefore knows which keys it needs to store in wait for messages that have not arrived yet.

The result is a strong and secure algorithm. However this comes at the price of being computationally intensive and requiring space for a lot of stored keys.

We note that while Signal gets a new key for each message sent, both SAKE and EDHOC get session keys meant for use in a session consisting of several messages. This way of using session keys is the more common and is seen for instance in the Transport Layer Security (TLS) protocol [DR08]. A connection between our protocols is that SAKE uses a mechanism that is very similar to the second ratchet of Signal. That is, it updates the long-term keys each time it is run. EDHOC on the other hand uses a mechanism that is similar to the first ratchet in Signal, however on a per-session basis instead of a per-message basis.

Key evolution can introduce the challenge of synchronicity. Two parties must somehow stay in sync when evolving a master key, otherwise they will not be able to

find a shared secret key. This problem can be solved by keeping a counter to keep track of which evolution each party is at, or by relying on clock timers to evolve the master key at a set time or interval. These solutions however, are resource intensive and not necessarily perfect. We will see that SAKE solves this problem in a different manner in section 3.2.

2.3 Key Compromise Impersonation attacks

A particular challenge to AKE protocols is the exploit known as a Key Compromise Impersonation (KCI), which was first identified by *Blake-Wilson et al.* [BJM97]. A KCI exploit is possible when an adversary has gained access to the private key of an honest party. The exploit allows the adversarial party to impersonate the compromised party to other parties, understandably since the adversary has both keys of the honest party. More interesting is that the adversary can also impersonate any third party to the honest party. This opens up possibility for the adversary to amongst other things, perform Man-in-the-Middle exploits [HGFS15].

2.4 The Internet of Things

To understand the intentions behind the proposal of the SAKE protocol it is necessary to explain where the intended use for the protocol is and what challenges this environments can have.

In the field of information technology the IoT is a much used term that does not have a common definition. One such definition is that the IoT generally describes technologies and physical objects that come together to communicate data to other devices over the internet or other communication channels [WF15]. This is only one of many slightly differing definitions that attempts to describe a wide area within the field of communications technology. Other definitions specify that the IoT involves sensors embedded in devices that can communicate the data collected without human interaction [Li17a].

2.4.1 Wireless Sensor Networks

An important part of the Internet of Things is the concept of Wireless Sensor Network (WSN). WSNs are networks consisting of wireless connected nodes. Each node contains one or more physical sensors that record data from their environments, some computational power in the form of an embedded CPU and the hardware necessary to communicate data over the internet or other wireless solutions [KB17]. These nodes can often be low-cost, autonomous units that are expected to operate for longer periods of time without physical maintenance. The lack of physical maintenance may

come from reasons such as hostile environments or because widespread deployments amount to a lot of maintenance.

The devices used in WSNs can greatly benefit from protocols specifically designed for use in these types of networks. Lightweight protocols that have been designed for low resource use and economical power use can give lightweight devices a boost in longevity.

2.4.2 Related work

In the paper "A secure end-to-end IoT solution", *Mathur et al.* [MNE+17] proposes an IoT system for connecting sensors to other devices. The proposed scheme is a secure end-to-end system that is intended for monitoring medical patients using sensor nodes. The scheme is defined as being capable of connecting IoT-sensors to any PC, while keeping direct access to the sensors from the internet.

The paper is of interest as the proposed scheme relies on ECDH and HKDF for key management. The paper gives an analysis of both alongside some other algorithms like RSA and AES. The analysis gives results for time and energy spent generating keys. This is of interest as it presents comparable values from an implementation in the same system.

Chapter 3
Protocols

This chapter intends to give an understanding of the chosen protocols, how they work and to give a brief comparison of the two.

The EDHOC protocol was chosen because it is being developed as a possible industry standard for IoT and constrained environments. With increasing amounts of IoT devices being deployed it is desirable to have a protocol designed with the specific requirements in mind. EDHOC is the Internet Engineering Task Force (IETF)s response to this. The protocol uses established methods and algorithms to give a safe and secure service, such as using ECDH to achieve PFS.

Being based on asymmetric encryption and using established methods, EDHOC is a good protocol to use as a comparison to SAKE. SAKEs use of symmetric encryption and evolving keys is a very different solution and having a point of comparison that uses different but more established methods is useful for establishing how well SAKE achieves its goals.

3.1 EDHOC

The EDHOC protocol, as defined by the EDHOC draft paper [SMP21], is a protocol that is being developed by an IETF work group. This protocol is being developed to become a possible industry standard AKE protocol for constrained environments. It is being developed with other protocols and tools developed by IETF, such as The Constrained Application Protocol (CoAP) and Concise Binary Object Representation (CBOR), in mind. Of interest to this project is that while EDHOC is meant for the same use cases as SAKE, it utilizes asymmetric cryptography instead of the symmetric key cryptography used in SAKE. This will be further discussed in Sections 3.3.3 and 3.3.4 .

EDHOC uses ECDH to provide key agreement with forward secrecy, as explained in Section 2.2.2. The protocol is being developed with multiple negotiable ciphersuites.

A cipher suite in this case is a set of algorithms that an AKE protocol uses when exchanging keys [DR08]. In the case of EDHOC the cipher suites specify:

- an Authenticated Encryption with Associated Data (AEAD) algorithm
- a hash algorithm
- the length of the MAC in bytes
- a key exchange algorithm
- a signature algorithm and AEAD algorithm for applications
- a hash algorithm for applications

The protocol currently has 9 defined ciphersuites and 3 private suites that users can define themselves. Suites 0 to 3 are meant for constrained IoT use cases and are based on AES-CCM. Suites 4 and 5 are meant for less constrained environments and are based on ChaCha20. Suite 6 is intended for environments with no general constraints. Suites 24 and 25 are meant for cases that require high security, such as governments or financial use cases. The naming of the suites leave a large amount of undefined suites available for future needs. With multiple cipher suites being implemented it means that negotiation of which suite to use must be handled during the key exchange process.

Fig. 3.1 is a simplified representation of EDHOC made to show the operation of the protocol. The figure does not show the full operation of the protocol but tries to focus on the elements that are relevant to this project.

The version displayed uses cipher suite 0 and method 3 for authentication. This configuration is considered the most relevant for this project. This is because cipher suite 0 is intended for low power deployments in IoT. Method 3 for authentication is considered useful because this specifies the use of ephemeral-static Diffie-Hellman to achieve authentication. Method 0 has both parties use digital signatures for authentication, while in methods 1 and 2 a digital signature is used by one of the parties for authentication as the other party uses Diffie-Hellman.

Since method 3 is used for authentication both parties must generate an individual static key pair, R, G_R for the responder and I, G_I for the initiator.

Message 1: Before it can send the first message, the Initiator must generate an ECDH ephemeral key pair, X, G_X where $G_X = X \cdot G$ and G is a point on the elliptic curve. The public key G_X is sent in message one, along with identifiers for the chosen method and cipher suite.

Message 2: Upon receiving message 1 the responder Generates an ephemeral key pair, Y and G_Y, as well. The responder can then generate the shared ECDH secret G_{XY}. Using the shared secret a PRK, PRK_1 is generated. PRK_1 is used to derive the key-stream that will be used to encrypt part of message 2. G_{RX} alongside PRK_1 is used to generate PRK_2. The responder also generates a transcript hash of message 1. PRK_2 and the transcript hash are used to generate a MAC. This MAC is then encrypted using PRK_1 and sent in message 2 alongside G_Y and G_R. Upon receiving message 2 the initiator must generate PRK_1 as well in order to decrypt the MAC. It must then generate the MAC on its own in order to verify the MAC that it has received.

Message 3: Having authenticated the identity of the responder, the initiator uses a static ECDH key pair, I, G_I, to authenticate itself to the responder. The initiator generates PRK_3 and a transcript hash of message 2, which is then used to create a new MAC. The MAC is encrypted using the EDHOC AEAD algorithm and sent in message 3 along with the public key G_I. Upon receiving the message the responder decrypts the ciphertext and verifies the MAC.

Message 4: Message 4 is optional and only needs to be supported in deployments where no protected application message is sent from the responder to the initiator.

It is important to note that the protocol is still in development. This means that current specifications for EDHOC may not be final and that there is a possibility of changes and additions.

3.2 SAKE

The SAKE protocol, as defined in the SAKE paper [ACF20], is a proposed protocol intended to provide strong security properties while using symmetric-key cryptography [ACF20]. Figure 3.2 taken from [ACF20] shows the proposed operation of the protocol.

The protocol has been developed as an AKE protocol meant for low-resource environments. The protocol relies only on symmetric cryptography to avoid the heavier draw on resources that asymmetric methods rely on. In addition, the protocol is designed to provide stronger security than other symmetric key protocols by guaranteeing PFS.

It is important to note that SAKE is not a fully defined protocol but is loosely defined in a paper by *Avoine et al.* [ACF20]. This paper does not give specific communications details or alternatives to things like ciphersuites. Some assumptions must therefore be made when analyzing and making comparisons of the protocol.

Initiator		*Responder*
X, G_X		
	Message 1:$G_X \longrightarrow$	
		Y, G_Y
		PRK_1
		R, G_R
		PRK_2
		MAC_1
	Message 2:$\longleftarrow G_Y, G_R, enc(MAC)$	
if:$(vrf(MAC_1 \oplus PRK_2) \neq true$ **Then:** Abort		
I, G_I		
PRK_3		
MAC_2		
	Message 3: $G_I, enc(MAC) \longrightarrow$	
		if:$(vrf(MAC_2 \oplus PRK_3) \neq true$ **Then:** Abort
		EAD
		PRK_3
	Message 4: $\longleftarrow enc(EAD)$	

Figure 3.1: EDHOC protocol [SMP21] operation

The protocol provides PFS by employing an evolving master key, from which session keys are derived. The introduction of the evolving master key creates a challenge in that the initiator and receiver must somehow stay synchronised in the evolution of keys. Should one party evolve its master key out of sync with the other then the two will not find a common secret key.

The problem of synchronicity is solved by introducing a second master key used only for synchronization. The synchronization master key is updated at the same time as the authentication master key. The SAKE paper [ACF20] proves that by having the initiator store the previous, the current and the next synchronization keys, and by having the responder storing only the current synchronization key, the initiator will almost always have the sufficient information necessary to re-synchronize. This does however rely on the assumption that no concurrent sessions are running. As described by *Boyd et al.* in the paper "Symmetric Key Exchange with Full Forward Security and Robust Synchronization" [BDdK+21], if the protocol is run with two or more parties at the same time then synchronization can be permanently lost.

To check and maintain synchronization the proposed pseudo-code of SAKE uses several **if** and **else if** statements. These statements verify which step synchronization is at and dictate the necessary action to maintain synchronicity.

3.3 Comparison

To give a greater understanding of the protocols we make a comparison to highlight similarities and differences. Table 3.5 shows the results of this comparison.

3.3.1 Message count

An important part of any protocol is how many messages the protocol needs to send to complete its function. Metrics such as power usage and time cost will depend in part on the message count of the protocol. We therefore take a look at the necessary messages of each of our protocols.

First we look at the message count of SAKE. SAKE as defined by *Avoine et al.* [ACF20] has five mandatory messages. To achieve mutual authentication all five messages are required. Three sent by the initiator and two by the responder.

Another operating mode for SAKE named SAKE *in Aggressive Mode* (SAKE-AM), allows for one less message sent. This mode changes the formats of messages somewhat but keeps the calculations and formats largely the same as the regular operating mode.

A $(K, K'_{j+1}, K'_j, K'_{j-1})$	B (K, K')
$r_A \xleftarrow{\$} \{0,1\}^\lambda$	$\xrightarrow{A\|r_A}$
	$r_B \xleftarrow{\$} \{0,1\}^\lambda$ $\tau_B \leftarrow \mathsf{Mac}(K', B\|A\|r_B\|r_A)$ $m_B \leftarrow r_B\|\tau_B$
	$\xleftarrow{m_B}$
if $(\mathsf{Vrf}(K'_j, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow 0$ $K' \leftarrow K'_j$; kdf; upd_A; $\epsilon \leftarrow 0$ **else if** $(\mathsf{Vrf}(K'_{j-1}, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow 1$ $K' \leftarrow K'_{j-1}$; $\epsilon \leftarrow 1$ **else if** $(\mathsf{Vrf}(K'_{j+1}, B\|A\|r_B\|r_A, \tau_B) = \mathtt{true})$ $\delta_{AB} \leftarrow -1$ $K' \leftarrow K'_{j+1}$; upd_A; kdf; upd_A; $\epsilon \leftarrow 0$ **else** abort	
$\tau_A \leftarrow \mathsf{Mac}(K', \epsilon\|A\|B\|r_A\|r_B)$ $m_A \leftarrow \epsilon\|\tau_A$	
	$\xrightarrow{m_A}$
	if $(\mathsf{Vrf}(K', \epsilon\|A\|B\|r_A\|r_B, \tau_A) = \mathtt{false})$ abort **if** $(\epsilon = 1)$ upd_B kdf; upd_B $\tau'_B \leftarrow \mathsf{Mac}(K', r_B\|r_A)$
	$\xleftarrow{\tau'_B}$
if $(\epsilon = 0)$ $K' \leftarrow K'_j$ **if** $(\mathsf{Vrf}(K', r_B\|r_A, \tau'_B) = \mathtt{false})$ abort **else if** $(\epsilon = 1)$ $K' \leftarrow K'_{j+1}$ **if** $(\mathsf{Vrf}(K', r_B\|r_A, \tau'_B) = \mathtt{false})$ abort kdf; upd_A	
$\tau'_A \leftarrow \mathsf{Mac}(K', r_A\|r_B)$	
	$\xrightarrow{\tau'_A}$
	if $(\mathsf{Vrf}(K', r_A\|r_B, \tau'_A) = \mathtt{false})$ abort

Figure 3.2: SAKE Protocol [ACF20] operation

Boyd et al. [BDdK+21] make several suggestions of improved protocols that can provide the same or better results than SAKE, using the same methods while only sending 3 messages or less.

Now we look at EDHOC. The protocol has a message flow consisting of three messages that must be sent to achieve mutual authentication. Two sent by the initiator and one by the responder. The protocol also has an optional fourth message from the responder to the initiator. This message is only to be sent in cases when no protected application message is sent from the responder to the initiator. For instance if EDHOC is "only used for authentication and no application data is sent" [SMP21], then the responder must send message four. Another example is "When application data is only sent from the Initiator to the Responder" [SMP21] , then also message four must be sent.

Table 3.1: Message count

Protocol	Initiator	Responder
SAKE	3	2
SAKE-AM	2	2
EDHOC	2	1(2)

3.3.2 Message components

The efficiency of a protocol is also dependent on message sizes and what components are sent in each message. We therefore take a closer look at which components are necessary for our protocols and how they differ. In EDHOC the plaintext messages are comprised of concatenated strings. Cipher-text in the messages is derived by performing the XOR operation with a defined keystream.

EDHOC also encodes its messages as CBOR sequences. The CBOR data format is designed for small message sizes and very small code sizes [Bor20]. This results in EDHOC having quite short message sizes. From Figure 1 in the EDHOC specification paper [SMP21] we get example sizes in bytes of how large each EDHOC message is. These examples are shown in Table 3.2. This gives us a good idea of how large EDHOC messages are. The table shows examples with different authentication keys, static Diffie-Hellman keys or signature keys, and different header parameters, represented by "kid" or "x5t" .

	Static DH Keys		Signature Keys	
	kid	x5t	kid	x5t
message_1	37	37	37	37
message_2	45	58	102	115
message_3	19	33	216	242
Total	101	128	216	242

Table 3.2: EDHOC messages sizes in bytes [SMP21]

As SAKE is not a fully defined protocol, we make an approximation of the message sizes by looking at the components in each message. The size of these messages could be larger or smaller in a real life implementation. Additional data could be added to the messages to make them larger than our estimated sizes suggest. It is also possible that messages would be encoded in some format, like CBOR, that would decrease size of messages before sending.

Message 1 contains an identifier **A** concatenated with a pseudo random value r_A. r_A is used in the authentication process and as input for key derivation. We expect the keying material to be no larger than the output key. We therefore assume r_A to be at most 32 bytes. As **A** has no clear definition determining size is difficult. For simplicity's sake we assume it to be no larger than 32 bytes as well.

Message 2 is made up of a pseudo random value r_B, concatenated with a MAC. r_B has the same properties as r_A and as such is assumed to be 32 bytes. The MAC utilizes SHA256 and therefore has a hash value of 32 bytes [Han05]. Other MACs used in SAKE will also be 32 bytes. Message 3 consists of an integer, of a size of one byte, and a MAC. For a total size of 33 bytes. Both message 4 and 5 consist only of a MAC. Giving each a size of 32 bytes.

It is worth noting that while the message overhead of SAKE is slightly more than double that of EDHOC. It is still far smaller than that of the TLS protocol, which has a message overhead of 789 bytes [GM22]. This means that SAKE does have almost three times smaller messaging overhead than TLS, making it suited to its intended low cost purpose.

Message	Size
message 1	64
message 2	64
message 3	33
message 4	32
message 5	32
Average	45
Total	225

Table 3.3: SAKE message sizes in bytes

3.3.3 Key storage and key management

Key management is an important factor that often differs between symmetric- and asymmetric-encryption. In addition the storage of keys, and other components, makes demands of the storage space which affects the efficiency of the protocol. We therefore take a look at the key management strategies of both protocols and what components they need to store.

Once again we start by looking at EDHOC. With EDHOC each party needs to store an ECDH key pair for encryption and decryption of messages. Since ECDH is used for authentication as well, then a second key pair must be stored by each party. Note that the static keys only need to be stored while the protocol is still running and can be deleted once the session key is computed.

We know that EDHOC specifies the use of Curve25519 [SMP21]. We know that this curve specifies keys, both public and private, to be 32 bytes long. This results in each party needing to store 128 bytes worth of keys or 256 bytes in total.

Looking at SAKE we see that both parties must store at least one version of the keys from each key chain it uses. The parties can also store up to three keys to lower the need for recomputing keys. In addition both parties must also store a Pre-Shared Key (PSK) that is used to derive the key chains.

To be used as keying material we expect the pre shared keys to be at least 32 bytes long. In addition we also expect the keys derived from the key chains to of 32 bytes length. This results in each party needing to store at the least 96 bytes of keys and up to 224 bytes if more keys are stored to reduce the need for recomputing.

	SAKE	EDHOC
Stored components	4-8 keys	4 key pairs
Total stored component size	96 (224) bytes	128 bytes

Table 3.4: Keys stored by SAKE and EDHOC

It is also important to note the differences in how the protocols do key management. Because it uses asymmetric cryptography EDHOC has an easily scalable solution using public-key infrastructure, though this does come at a computational and storage cost. SAKE on the other hand relies on pre-shared keys which greatly limits the amount of connections a party can have. PSKs also bring the challenge of how the original keys have been shared and adds vulnerability to a system. With already existing keys the possibility of a third party gaining access to the keys is always there.

3.3.4 Security features

It is important to take a look at the security features of both protocols, so that we can determine what level of security they seek to achieve and whether they do so.

Providing forward secrecy is an important feature for both protocols. According to both *Avoine et al.* [ACF20] and *Boyd et al.* [BDdK+21] the SAKE protocol does provide forward secrecy, as explained in section 2.2.2. As EDHOC uses ephemeral Diffie-Hellman it is easy to see that it also provides forward secrecy, as this method is a well established way to provide PFS [BJPS18].

As mentioned in section 2.3, AKE protocols are vulnerable to KCI attacks and need protection against it. *Selander et al.* [SMP21] explain that while EDHOC is protected against KCI attacks when it is authenticated using signature keys, it has some vulnerability when static Diffie-Hellman is used. This means that a party who accepts authentication with digital signatures will be protected, but a party who accepts authentication using MACs with static Diffie-Hellman keys will be vulnerable to a KCI attack. In the case of SAKE, *Selander et al.* [ACF20] explain that their protocol would be entirely vulnerable to KCI attacks.

As both SAKE and EDHOC are AKE protocols, authentication is a key part of their purpose. Each protocol is capable of providing one-way authentication or mutual authentication, though both are primarily meant for mutual authentication. SAKE uses authentication keys and MACs to achieve mutual authentication. EDHOC has the option to either use static Diffie-Hellman or digital signatures provide mutual authentication, both well established methods for this goal.

According to *Gunther et al.* [GM22] EDHOC provides strong security for the AKE process. However they also point out that the protocol is somewhat brittle when the mac-then-sign method is employed as the authentication method. *Gunther et al.* go on to state that while EDHOC may have some security issues in its current state, the working group who are developing the protocol actively integrate recommendations and fixes from analyses.

As mentioned in section 2.1.8 methods of symmetric cryptography are resilient or even completely safe in a post-quantum environment. Some minor modifications such as doubling key length may be all that is needed to maintain the security level of SAKE when faced with the threat of quantum attacks. EDHOC on the other hand would be vulnerable and open to such attacks. This means that in a long term view, SAKE may be much more viable than EDHOC.

Table 3.5: Comparison of SAKE and EDHOC

	SAKE	EDHOC
Messages sent	5 mandatory messages	3 messages with optional 4th
Average Message size	45 bytes	34 bytes
Total size of messages	225 bytes	101 bytes
Stored component size	96 (224) bytes	128 bytes
Mutual Authentication	Yes	Yes
KCI resistance	No	Partial[a]
Post-Quantum resilience	Yes	No

[a]depends on authentication method

Chapter 4
Method

This chapter gives an explanation of the method used to find results that can bring clarity to the research questions, and the resulting findings.

4.1 Method

The research questions ask about SAKEs energy efficiency and how it compares to other protocols. To answer this, an estimate of the energy usage of SAKE is needed. This is achieved with a granular approach. We start by breaking the protocol down into its constituent cryptographic functions, the basic building blocks of protocols. Then we take already existing findings, from reports or papers, that establish how much energy these functions use. By adding up each cryptographic functions energy use, an estimate of energy usage is made through focusing on the simplest parts of the protocols.

Additionally a comparison is necessary as a point of reference to be able to gauge how efficient the estimate for SAKE appears to be. Therefore, an estimate of the EDHOC protocol is also made. The EDHOC protocol is meant for the same use cases as SAKE and uses asymmetric cryptography instead of SAKEs symmetric encryption. Thereby a comparison with EDHOC shows the difference between protocols that are developed for the same use cases, but use different methods for achieving their goals.

Further, to make a useful comparison of the two protocols, some assumptions were made. As previously stated, SAKE is not a fully defined protocol. That is to say specific algorithms and functions are not specified in the article that proposes the SAKE protocols [ACF20]. This analysis therefore chooses to use whatever equivalent primitives that are already defined for EDHOC, as it is developed for a similar environment and therefore assumed to use algorithms and functions with properties that are also wanted in SAKE.

4.2 Cryptographic Primitives

To make a comparison of the efficiency of the protocols, the cryptographic primitives of the protocols have been singled out. Cryptographic primitives are functions and algorithms that are well established within the field of cryptography and information security [Bar20a]. These primitives are the building blocks used to construct the algorithms and protocols of computer security.

In the paper that proposes the SAKE protocol the cryptographic primitives below are specified as being necessary for the protocol. The cryptographic primitives are all found in part 3.1, "Description of the protocol", in the SAKE paper [ACF20].These are all types functions that are required to construct the SAKE protocol.

- A non-invertible functions, see Section 2.1.3
- A secure MAC function, see Section 2.1.6
- A key derivation function, see Section 2.1.7
- A Pseudo Random function, see also Section 2.1.7

Likewise we need the cryptographic primitives that the EDHOC protocol uses. Chapters 2,3,4 and 5 are all used to determine the necessary cryptographic primitives listed below.

- A Key Derivation Function
- A MAC function
- A Hash algorithm
- An ECDH key pair derivation function
- A signature algorithm

4.3 HKDF

To find an estimate of energy use, algorithms and functions with tested implementations are needed. The choice was made to use HKDF in place of all the cryptographic primitives except for the ECDH key derivation function. This decision was based on several reasons.

The EDHOC protocol specifies several functions and algorithms. The ciphersuites 0-3 dedicated for constrained environments, specify the use of HKDF for pseudo

random key generation. HKDF is also specified in MAC, hash and signature operations in EDHOC. These are all forms of primitives that take an input, perform a mathematical operation on the input to transform it into a fixed length output [Shi07]. We also choose to focus on EDHOC using method 3 for authentication, that is to say static Diffie-Hellman keys, because this is the most efficient mode.

The operations of HKDF mostly rely on the computation of an HMAC, which in turn relies on a hash function. As such this is where the majority of HKDFs computational cost comes from. As there is little difference in these operations we assume them to have approximately the same computational cost, allowing us to use the cost of HKDF in place of the other operations.

Additionally, it was desirable to find cryptographic primitives that have all been implemented in the same system. If values of energy use was taken from implementations in different systems then the differences in implementations could have affected efficiency to a degree that would make estimations highly inaccurate or even useless. The implementation done by *Avijit et al.* [MNE+17] uses both HKDF and ECDH key derivation and the HKDF implementation uses SHA256, as specified in EDHOC. Therefore it suits the purpose of this project.

Having decided this, the values in table 4.1, taken from [MNE+17] are the only energy use values needed to make an estimate of both protocols' energy use. The two values are the recorded energy use of the specified algorithms when deriving keys.

Table 4.1: Energy usage values(mJ)

HKDF	0.049
ECDH	21.49

4.4 Summation of primitives

Finally, to make estimates of power use it is necessary to know how many times the cryptographic primitives are used in each protocol. We see in Fig 3.2 a representation of the SAKE protocol. By identifying the cryptographic primitives in this figure it is easy to count how many times the cryptographic primitives are called in a run of SAKE.

- r_A and r_B represent instances of Pseudo Random Function
- Mac(...) is an instance of a MAC function.
- Vrf(...) is a verification function in which the MAC function is performed once.

– kdf corresponds to one performance of the Key Derivation Function

– upd_A and upd_B are both functions where the respective authentication and derivation keys of a party are updated using the KDF. Meaning that upd_A and upd_B represent two performances of HKDF.

Knowing this, it is easy to make a simple estimate of the energy usage of a run of SAKE. It is important to note an element that allows for differences between separate runs of SAKE. Namely the **if** and **else if** statements. The possibility of different statements ending up true or false allow for different amount of calculations needed for separate runs, with even the possibility of runs being aborted. This opens up the possibility of best-case and worst-case scenarios where runs never abort, but there is a considerable difference in the amount of times cryptographic primitives are used.

To find the cryptographic primitives in EDHOC we use the explanations in chapter 5: Message Formatting and Processing in [SMP21]. This chapter explains how each message in EDHOC is created and processed, and by going through it is simple to count the amount of times the cryptographic primitives are used. The tally of cryptographic primitive instances in both SAKE and EDHOC is shown in table 4.2. This table does not include the instances of ECDH in EDHOC. This is because it is simpler to add up the amount of HKDF instances first. The use of ECDH key generation only happens once each for initiator and responder in EDHOC. The ECDH energy values can therefore easily be added after calculating the energy use of HKDF in EDHOC.

It is important to note that runs which end with the protocol being aborted have not been taken into account by this project. To do this it would have been necessary to have statistical data on how often the protocols abort instead of completing a handshake. Such data would need to come from some sort of implementation which is not available at the time of writing.

Table 4.2: Cryptographic primitives in protocol runs

Run	Initiator	Responder
SAKE Worst case	15	10
SAKE Best case	6	8
EDHOC	5	4

4.5 Communication cost

In addition to estimating the energy cost of computation it is necessary to gain some understanding of other ways the protocols consume power. When operating a node in a WSN draws different amounts of power when it is computing, transmitting, receiving, listening and waiting.

To estimate the consumption of transmitting and receiving we need to know the cost per bit for each of these operations and how many bits each protocol sends. From table 3.5 we get our estimate of message size. To find cost per bit we utilize the values in table 4.3, borrowed from *De Meulenaer et al.* [dMGSP08]. Multiplying cost per bit with the bit size of messages sent and received give us an easy estimation of how much these processes cost.

We also get an estimate of cost when computing, listening and sleeping from *De Meulenaer et al.*. However these estimates gives energy cost per cycle. It is difficult to use these estimates as we have no way of determining the amount of cycles or time spent by SAKE. An operational implementation would be required to find usable estimates.

We make the assumption that listening costs of EDHOC will be larger than those of SAKE. *De Meulenaer et al.* make the same assumption based on the fact that elliptic curve computations require considerably longer computations than computations for symmetric functions [dMGSP08].

It is important to note that the results we draw from these estimates are only rough estimates. The numbers we find are not applicable to real world situations, but they give an approximation that can guide choices regarding the usefulness and use-cases applicable to our protocols.

One reason why the estimates we find are only guiding answers is because we draw data from two different sources. From *De Meulenaer et al.* [dMGSP08] we get estimates for the energy use of communication and the energy cost of computation using elliptic curves. However it does not give values for HKDF. From *Mathur et al.* [MNE+17] we get estimates of the power consumption when computing keys using both ECDH and HKDF, but no indication as to how much communication costs. We still find the sources to be compatible for our comparison due to the similarity in costs. *De Meulenaer et al.* estimate the cost of elliptic curve multiplication to be 55 mJ or 17 mJ, depending on which hardware it is implemented in. *Mathur et al.* estimate the key generation of ECDH to cost 21.49 mJ. While the costs do vary they are not in any great order of difference and as such we deem them to be close enough to useful for our high level comparison. It is also notable that while the paper by *De Meulenaer et al.* was published in 2008 it focuses on hardware and protocols that

are still highly relevant and therefore useful for this thesis.

Energy cost	MICAz	TelosB
Compute for 1 T_{clk}	3.5 nJ	1.2 nJ
Transmit 1 bit	0.60 μJ	0.72 μJ
Receive 1 bit	0.67 μJ	0.81 μJ
Listen for 1 T_{clk}	9.2 nJ	15.0 nJ
Sleep for 1 T_{clk}	3 pJ	9 pJ

Table 4.3: Energy costs of MICAz and TelosB

Chapter 5
Results and Discussion

This chapter presents the results that have been found by using the previously mentioned methodology. The results are also discussed and expanded upon.

5.1 Results

The energy cost is divided into three sections. We estimate the cost of computation, first that of SAKE then that of EDHOC. The estimation of communication costs covers both protocols in one subsection because of how simple the estimations are to calculate.

5.1.1 Energy use of SAKE computation

Using the values from table 4.1 and the amount of uses of HKDF, shown in table 4.2, estimations of energy usage are found. By multiplying the energy usage of a single instance of HKDF key derivation by the amount of times HKDF is called throughout a run a simple estimate of energy usage is calculated. The table below shows the energy usage of best case and worst case runs and additionally the average of the two.

Table 5.1: Energy use SAKE (mJ)

Run	Initiator	Responder	Total
Worst case	0.735	0.490	1.225
Best case	0.294	0.392	0.686
Average	0.514	0.441	0.955

5.1.2 Energy use of EDHOC computation

In the same manner as done for SAKE, the values from table 4.1 and the determined instances of HKDF in EDHOC, from table 4.2, is used to find the estimated values

in the table below. In addition to HKDF an instance of ECDH key generation is done by both Initiator and Responder. As such the value for ECDH key generation is added to both parties energy usage.

Table 5.2: Energy use EDHOC (mJ)

	Initiator	Responder	Total
Cost	21.735	21.686	43.421

5.1.3 Energy use of communication

To make an estimate of how much energy each protocol requires to transmit its messages we multiply the total bit size of message by the cost per bit. We do the same with cost per bit for receiving to find the energy used in receiving the messages. The results are displayed in table 5.3.

	SAKE		EDHOC	
Hardware	MICAz	TelosB	MICAz	TelosB
Transmitting	1.08	1.296	0.484	0.582
Receiving	1.206	1.458	0.541	0.654
Total	2.286	2.754	1.025	1.236

Table 5.3: Cost of communication (in mJ)

5.1.4 Comparison of energy use

To give as complete a picture as possible we add together the energy use of computation and communication in table 5.4.

	SAKE		EDHOC	
Computation	1.225 (0.686)		43.421	
Hardware	MICAz	TelosB	MICAz	TelosB
Communication	2.286	2.754	1.025	1.236
Total	3.511 (2.972)	3.979 (3.4)	44.446	44.657

Table 5.4: Comparison of energy costs

5.2 Discussion

5.2.1 Energy cost

The results of the estimation of power use by each protocol shows a marked difference. From table 5.4 it is plain to see that even in its worst case scenario for power use SAKE has a distinctly lower cost than EDHOC. Looking at the computation cost, when using EDHOC the initiator expends over 29 times the energy than it does when using SAKE. For the responder the difference is even greater, over 44 times the power use. This shows that SAKE seems to be a substantially more energy efficient protocol than EDHOC is. Taking transmission cost into consideration the difference becomes slightly less pronounced, due to SAKEs higher message count. More messages invariably result in higher energy cost, however this increase is quite small compared to the computation cost. The result being that the cost of EDHOC is still an order of magnitude larger than that of SAKE.

But however great the difference in cost, we cannot take these results entirely at face value. There are several considerations that must be taken into account when viewing these result. Firstly, the implementation of the protocols can impact the performance of the protocol in question. With SAKE being only a proposed protocol with no implementations yet it is difficult to say how this may affect the efficiency. As for EDHOC, it is still in development and therefore has the possibility to change. With neither protocol being a finished product it must be kept in mind that choices could be made that can greatly affect the future efficiency of these protocols. It is also worth mentioning that the choice of hardware can greatly impact the energy costs. Both computation and communication costs can change depending on the choices of hardware.

5.2.2 Key management

The difference in key management is an important aspect of the protocols we look at. The differences between symmetric and asymmetric cryptography necessitate separate approaches to managing keys.

This is immediately clear when we see that SAKE relies on Pre-Shared Keys. The use of PSKs presents challenges that EDHOC does not need to take into account. When using PSKs the keys either need to come pre-loaded on the device or some other protocol is needed to communicate the keys before they need to be used. If the key are pre-loaded this poses a security risk and additional complexity if the keys for some reason need to be updated. If not then another key exchange protocol would have to be implemented in a device using SAKE adding complexity and taking up storage space.

EDHOC on the other hand requires a public key infrastructure to operate. As mentioned in section 2.1.3, the key management of asymmetric cryptography is much more easily scalable than symmetric and this is no exception. Once public key infrastructure is established it easily scales with many parties. SAKE on the other hand requires pre-loading keys in each individual device or an individual communication session with each device to establish the PSKs.

5.2.3 Security levels and features

Our findings in section 3.3.4 show that both protocols have quite solid security and similar features. *Fan et al.* [FCS+22] show that SAKE has strong security features. Likewise *Gunther et al.* supports EDHOCs security, although it does point out some possible brittleness.

SAKE does have an issue with parallel sessions. SAKE does not allow parallel sessions as it may cause sessions to abort. This is not an issue common to AKE protocols and not something EDHOC has trouble with. *Boyd et al.* [BDdK+21] suggest three protocols also based on symmetric methods, but with all protocols allowing for parallel sessions. This shows that this issue with SAKE can be solved without resorting to using asymmetric methods, but it necessitates modification of SAKE.

It is noteworthy to take a look at the differences in security features between the two protocols. EDHOC has many implemented security features such as PFS, identity protection and protection against some types of attacks [SMP21]. Being a protocol in development, EDHOC is updated with new security features after drafts have been analyzed. SAKE on the other hand has little focus on features other than authentication and PFS. This is natural since creating an AKE protocol using symmetric key cryptography and guaranteeing forward secrecy is the intention of the proposal. This does mean that any attempt to implement SAKE must take into consideration which other features, if any, are to be included.

As mentioned in section 2.1.8, symmetric encryption can remain secure in a post-quantum environment with minimal changes. Asymmetric methods however can become vulnerable in such an environment. This goes for our protocols as well. SAKE would only need minimal changes to key sizes, while EDHOC would become vulnerable to attack with no easy way to fix the issue. This means that in a long term perspective, SAKE preempts issues that may arise from developments in quantum computing.

Chapter 6
Conclusion

To conclude the thesis this chapter gives some remarks with regards to the research questions. The chapter also points out possible issues with the project and gives recommendations to future work on the SAKE protocol. Finally, some closing remarks to the project are given at the end.

6.1 Answering Research Questions

Taking the previous chapter into consideration, the research questions can be answered, albeit with some possible issues which will be discussed in the next section. We stated the following research questions to begin with.

- Is the energy usage of SAKE efficient?
- How does SAKE compare to other protocols intended for similar use?

Our results show that SAKE appears to be very efficient when it comes to power use. The results of the energy use estimations show us that the SAKE protocol is a lot less energy intensive than EDHOC. EDHOC, being on average far more energy intensive for both initiator and responder in the average case, is a great deal more taxing on the energy use of a system than SAKE. SAKE is slightly less efficient when it comes to message count, however this does seem to be outweighed by its low communication cost.

Also, when compared to EDHOC, the protocol does seem to hold up quite well. SAKE is a match in security features compared to EDHOC, and add resistance to quantum attacks. Key management does provide a challenge that asymmetric methods deal with more capably, but this is not insurmountable. And as mentioned SAKE has a much lower energy cost than EDHOC.

6.2 Possible issues

There are several possible issues that may have affected the results of this project, or may affect their future validity.

To start with, the results of the energy use estimations rely heavily on the results of *Mathur et al.* [MNE+17]. While the difference shown between HKDF and ECDH in energy use is considerable, the possibility of some mistake that could have effected the results does exist. Any mistake that could have impacted these measurements would effect the results of this project.

Additionally, this project does not take code size into account. It is hard to make an estimate of the code size and computational overhead of SAKE without an implementation to base it on. While there does exist implementations of EDHOC, a point of comparison is not useful without an estimate of SAKE to compare with.

Lastly, SAKE is only a proposed protocol and further development and implementation may change protocol drastically. The EDHOC protocol is also still just a draft. Further iterations of the protocol may make changes to features and specifications. Assumptions that have been made by this project may therefore prove to be in error and thereby invalidate the results of the project. However we do not expect any major changes to be made, making the results this project presents a solid guide to the cost of the protocols.

6.3 Future work

The results of the energy estimations show that the SAKE protocol has promise, but further work is required. An implementation of the protocol would be highly useful.

Any implementation would have to make choices regarding which algorithms and methods to use. These choices and subsequent attempts to use them in an implementation will give greater insights into the efficiency of the protocol. Simple simulation using an implementation may be used to get a better estimation of the energy usage of the protocol and give hints as to how the protocol will act in a physical environment. Even further, implementing the protocol in a physical device will allow for in depth testing in a realistic environment.

Testing an implementation of SAKE on the same hardware as a comparable protocol, such as EDHOC, would give a solid view of just how much more efficient the protocol is and would make for a good guide regarding choices of further use for the protocol. Adding to that it would be interesting to see the behavior of the protocol on newer hardware developed for the IoT field.

As further work it is therefore recommended to:

- Develop an implementation of the protocol
- Simulate the operation of the protocol
- Implement the protocol in a physical device for testing
- Make further comparisons with other protocols running on the same hardware

6.4 Final Remarks

It seems clear from our results that SAKE is much more efficient than EDHOC when it comes to energy use. The cost of computation is very low compared to EDHOC and while the cost of communication is somewhat larger it still outperforms EDHOC. Without a proper implementation it can not be said for certain exactly how much more efficient in energy use SAKE is in comparison with EDHOC, but our results give a clear indication that SAKE is at least an order of magnitude less energy consumptive than EDHOC. The SAKE protocol shows great promise and with further work can become a valuable tool in the field of IoT and communications technology.

References

[ACF20] G. Avoine, S. Canard, and L. Ferreira, «Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy», in *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*, S. Jarecki, Ed., ser. Lecture Notes in Computer Science, vol. 12006, Springer, 2020, pp. 199–224. [Online]. Available: https://doi.org/10.1007/978-3-030-40186-3%5C_10.

[Bar20a] E. Barker, «Guideline for using cryptographic standards in the federal government: Cryptographic mechanisms», National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP)800-175B, Rev. 4, Includes updates as of March 22, 2020, 2020.

[Bar20b] ——, «Recommendation for key management: Part 1 – general», National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP)800-57 Part 1, Rev. 5, Includes updates as of May, 2020, 2020.

[BDdK+21] C. Boyd, G. T. Davies, *et al.*, «Symmetric key exchange with full forward security and robust synchronization», in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, M. Tibouchi and H. Wang, Eds., ser. Lecture Notes in Computer Science, vol. 13093, Springer, 2021, pp. 681–710. [Online]. Available: https://doi.org/10.1007/978-3-030-92068-5%5C_23.

[BFG+22] A. Bienstock, J. Fairoze, *et al.*, «A more complete analysis of the signal double ratchet algorithm», *IACR Cryptol. ePrint Arch.*, p. 355, 2022. [Online]. Available: https://eprint.iacr.org/2022/355.

[BJM97] S. Blake-Wilson, D. Johnson, and A. Menezes, «Key agreement protocols and their security analysis», in *Cryptography and Coding, 6th IMA International Conference, Cirencester, UK, December 17-19, 1997, Proceedings*, M. Darnell, Ed., ser. Lecture Notes in Computer Science, vol. 1355, Springer, 1997, pp. 30–45. [Online]. Available: https://doi.org/10.1007/BFb0024447.

[BJPS18] A. Bruni, T. S. Jørgensen, *et al.*, «Formal verification of ephemeral diffie-hellman over COSE (EDHOC)», in *Security Standardisation Research - 4th International Conference, SSR 2018, Darmstadt, Germany, November 26-27,*

2018, Proceedings, C. Cremers and A. Lehmann, Eds., ser. Lecture Notes in Computer Science, vol. 11322, Springer, 2018, pp. 21–36. [Online]. Available: https://doi.org/10.1007/978-3-030-04762-7%5C_2.

[BL17] D. J. Bernstein and T. Lange, «Post-quantum cryptography», *Nat.*, vol. 549, no. 7671, pp. 188–194, 2017. [Online]. Available: https://doi.org/10.1038/nature23461.

[Bor20] C. Borman, 2020. [Online]. Available: https://cbor.io/.

[CCD+16] K. Cohn-Gordon, C. Cremers, *et al.*, «A formal security analysis of the signal messaging protocol», *IACR Cryptol. ePrint Arch.*, p. 1013, 2016. [Online]. Available: http://eprint.iacr.org/2016/1013.

[dMGSP08] G. de Meulenaer, F. Gosset, *et al.*, «On the energy cost of communication and cryptography in wireless sensor networks», in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2008, Avignon, France, 12-14 October 2008, Proceedings*, IEEE Computer Society, 2008, pp. 580–585. [Online]. Available: https://doi.org/10.1109/WiMob.2008.16.

[DR08] T. Dierks and E. Rescorla, «The transport layer security (TLS) protocol version 1.2», *RFC*, vol. 5246, pp. 1–104, 2008. [Online]. Available: https://doi.org/10.17487/RFC5246.

[DvOW92] W. Diffie, P. C. van Oorschot, and M. J. Wiener, «Authentication and authenticated key exchanges», *Des. Codes Cryptogr.*, vol. 2, no. 2, pp. 107–125, 1992. [Online]. Available: https://doi.org/10.1007/BF00124891.

[FCS+22] Q. Fan, J. Chen, *et al.*, «Sake*: A symmetric authenticated key exchange protocol with perfect forward secrecy for industrial internet of things», *IEEE Trans. Ind. Informatics*, vol. 18, no. 9, pp. 6424–6434, 2022. [Online]. Available: https://doi.org/10.1109/TII.2022.3145584.

[GM22] F. Günther and M. I. T. Mukendi, «Careful with mac-then-sign: A computational analysis of the EDHOC lightweight authenticated key exchange protocol», *IACR Cryptol. ePrint Arch.*, p. 1705, 2022. [Online]. Available: https://eprint.iacr.org/2022/1705.

[Gro96] L. K. Grover, «A fast quantum mechanical algorithm for database search», in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, G. L. Miller, Ed., ACM, 1996, pp. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866.

[GTDD08] C. M. Gutierrez, J. M. Turner, *et al.*, *Fips pub 198-1 federal information processing standards publication*, 2008.

[Han05] H. Handschuh, «SHA family (secure hash algorithm)», in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed., Springer, 2005. [Online]. Available: https://doi.org/10.1007/0-387-23483-7%5C_388.

[HC98] D. Harkins and D. Carrel, «The internet key exchange (IKE)», *RFC*, vol. 2409, pp. 1–41, 1998. [Online]. Available: https://doi.org/10.17487/RFC2409.

[HGFS15] C. Hlauschek, M. Gruber, *et al.*, «Prying open pandora's box: KCI attacks against TLS», in *9th USENIX Workshop on Offensive Technologies, WOOT '15, Washington, DC, USA, August 10-11, 2015*, A. Francillon and T. Ptacek, Eds., USENIX Association, 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/hlauschek.

[Hid21] J. D. Hidary, *Quantum Computing: An Applied Approach, Second Edition.* Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-83274-2.

[ISO10] ISO, *IT security techniques — encryption algorithms — part 3: Block ciphers*, https://www.iso.org/standard/54531.html, Dec. 2010.

[Kal00] B. Kaliski, «PKCS #5: Password-based cryptography specification version 2.0», *RFC*, vol. 2898, pp. 1–34, 2000. [Online]. Available: https://doi.org/10.17487/RFC2898.

[KB17] M. Kocakulak and I. Butun, «An overview of wireless sensor networks towards internet of things», in *IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC 2017, Las Vegas, NV, USA, January 9-11, 2017*, IEEE, 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/CCWC.2017.7868374.

[KE10] H. Krawczyk and P. Eronen, «Hmac-based extract-and-expand key derivation function (HKDF)», *RFC*, vol. 5869, pp. 1–14, 2010. [Online]. Available: https://doi.org/10.17487/RFC5869.

[lChe22] lily Chen, «Recommendation for key derivation using pseudorandom functions», National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP)800-108r1, Rev. 1, Includes updates as of August, 2022, 2022.

[Li17a] S. Li, «Chapter 1 - introduction: Securing the internet of things», in *Securing the Internet of Things*, S. Li and L. D. Xu, Eds., Boston: Syngress, 2017, pp. 1–25. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128044582000019.

[Li17b] ——, «Chapter 4 - iot node authentication», in *Securing the Internet of Things*, S. Li and L. D. Xu, Eds., Boston: Syngress, 2017, pp. 69–95. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128044582000044.

[MNE+17] A. Mathur, T. Newe, *et al.*, «A secure end-to-end iot solution», *Sensors and actuators.*, vol. 263, pp. 291–299, 2017.

[PM16] T. Perrin and M. Marlinspike, «The double ratchet algorithm», *GitHub wiki*, 2016. [Online]. Available: https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[Shi07] R. W. Shirey, «Internet security glossary, version 2», *RFC*, vol. 4949, pp. 1–365, 2007. [Online]. Available: https://doi.org/10.17487/RFC4949.

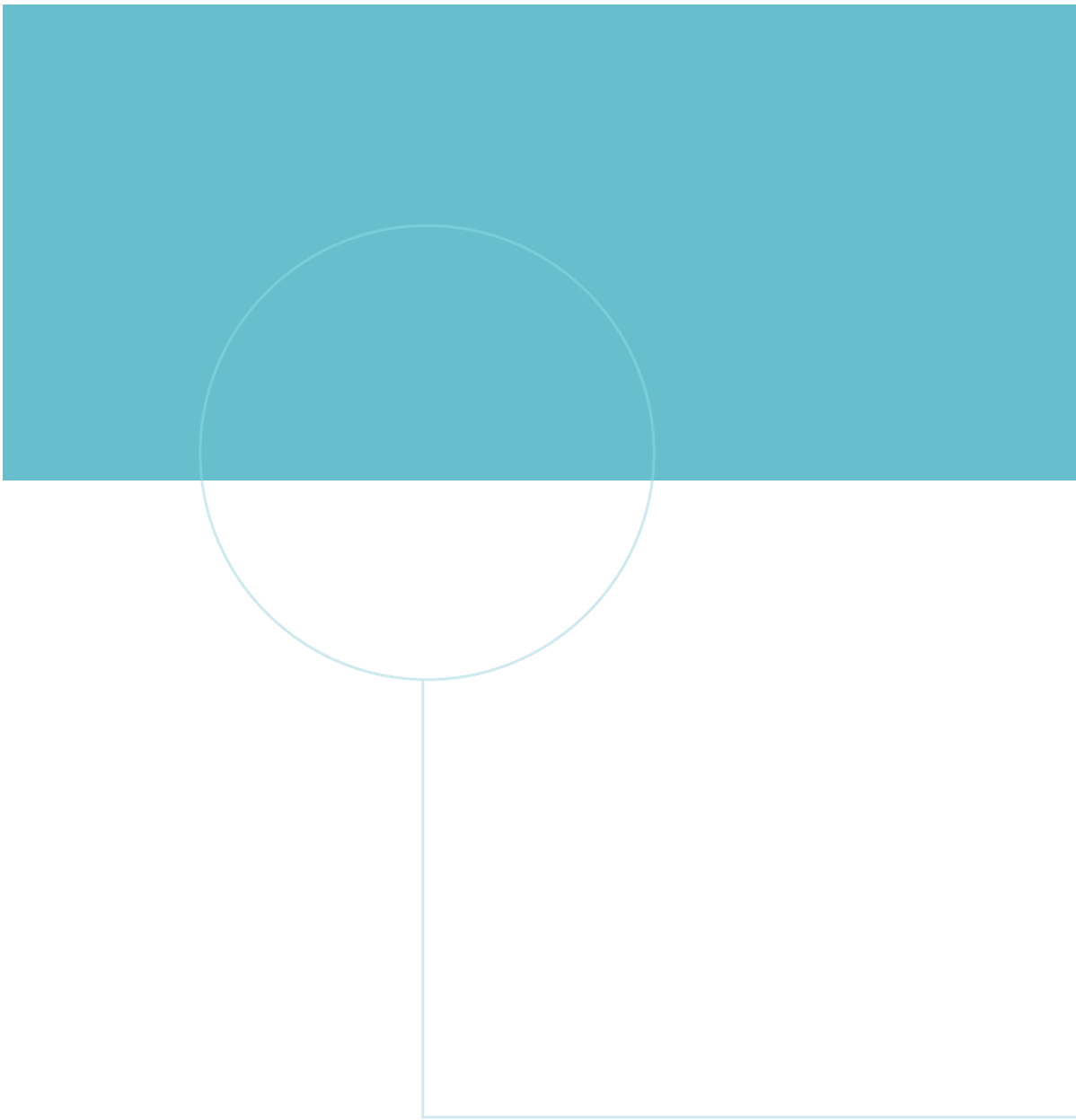
[Sho94] P. W. Shor, «Algorithms for quantum computation: Discrete logarithms and factoring», in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, IEEE Computer Society, 1994, pp. 124–134. [Online]. Available: https://doi.org/10.1109/SFCS.1994.365700.

[SMP21] G. Selander, J. Mattson, and F. Palombini, *Ephemeral diffie-hellman over cose (edhoc)*, https://datatracker.ietf.org/doc/draft-ietf-lake-edhoc/, Date: 2021-04-028, 2021.

[Sta20] W. Stallings, *Cryptography and network security - principles and practice (8. ed. Global ed.)* Prentice Hall, 2020.

[WF15] F. Wortmann and K. Flüchter, «Internet of things - technology and value added», *Bus. Inf. Syst. Eng.*, vol. 57, no. 3, pp. 221–224, 2015. [Online]. Available: https://doi.org/10.1007/s12599-015-0383-3.

[Zdz12] J. Zdziarski, *Hacking and Securing iOS Applications - Stealing Data, Hijacking Software, and How to Prevent It*. O'Reilly, 2012. [Online]. Available: http://www.oreilly.de/catalog/9781449318741/index.html.



NTNU

Norwegian University of
Science and Technology