NTNU – Trondheim
Norwegian University of
Science and Technology

Secure user-friendly on-premises storage for sensor data

Théau Giraud

Submission date:	June 2022
Supervisor:	Colin Boyd, NTNU
Co-supervisor:	Håvard Skara Mellbye, Disruptive Technologies

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

Title: **Secure user-friendly on-premises storage for sensor data**

Student: **Théau Giraud**

Problem description: Small sensors often communicate to a cloud server for data

storage. User data privacy is very important to some clients using such sensors, which is why an on-premises option can be preferred. This option must consume very little power on the sensor, due to their size, and address the low-latency requirements of edge support.

The goal of this thesis will be to explore options for design of on-premises storage for sensor data, mainly focusing on the right cryptographic tools. Activities may include a comparison of different cryptographic protocols to evaluate the best candidate for this solution, as well as a proof-of-concept implementation in a suitable environment.

Date approved:	2022-02-14
Responsible professor:	Colin Boyd, NTNU
Supervisor(s):	Håvard Skara Mellbye, Disruptive Technologies

Abstract

Sensors in the Internet of Things are becoming more and more prevalent, particularly lightweight sensors relying on low-power microcontrollers. Many companies develop such sensors with cloud storage in mind. But the need from clients to have an on-premises storage alternative has recently arisen, whether from a desire to be independent from sensor manufacturers' public clouds or for security and back-up purposes. There is a clear need for new storage, authentication and encryption methods. But when dealing with lightweight sensors, certain communication and cryptographic protocols can not be implemented due to their heavy resource requirements. Finding the right trade-off between security, performance constraints and user-friendliness is the true heart of this problem.

In this thesis we design state-of-the-art on-premises storage solutions for sensor data as well as alternatives to pre-existing sensor-to-cloud authentication and encryption in a lightweight sensor environment. This is backed by a threat model of different devices involved in a sensor's communications to cloud or on-premises storage along with a literature study of existing solutions. Our main contributions consist of suggestions for data storage, key storage, key generation at manufacturing and authentication independent from the sensor manufacturer's cloud while keeping the client's sensor data private. We also discuss different promising lightweight encryption primitives which may be applied to communications in such a use case.

Our results show that it is possible to add an on-premises storage solution to a pre-existing sensor-to-cloud protocol while keeping sensor data only available to the client. Storage is not a big concern, but different options bring up different security concerns. Authentication alternatives will always require a certain level of trust between the sensor manufacturer and client, as the manufacturer cannot be completely absolved from key generation and exchange. We have also found promising lightweight cryptographic primitives which can contribute to these solutions and make them more secure.

Norwegian Abstract

Sensorer i Internet of Things blir mer og mer utbredt, særlig lettvektssenorer som er avhengige av lav-energi mikrokontrollere. Mange selskaper utvikler slike sensorer med skylagring i tankene. Men nødvendigheten for at klienter har et lokalt lagringsalternativ har nylig oppstått, enten om det er med et ønske om å være selvstendig fra sensorprodusentens offentlige sky eller for sikkerhets- og backupformål. Det er et klart nehov for nye lagrings-, autentiserings- og krypteringsmetoder. Når man har med lettvektssensorer å gjøre er det visse kommunikasjons- og krypteringsprotokoller som ikke kan bli implementert på grunn av deres ressursbehov. Å finne riktig avveining mellom sikkerhet, ytelsesbegrensninger og brukervennlighet er kjernen i problemet.

I denne masteroppgaven designer vi en state of the art lokal lagringsløsning for sensordata i tillegg til alternativer til allerede eksisterende sensor-til-sky-autentisering og -kryptering i et lettvektssensormiljø. Dette er støttet av en trusselmodell for ulike enheter som er involvert i sensorers kommunikasjon med skyen eller lokal lagring sammen med en litteraturstudie av eksisterende løsninger. Hovedbidraget vårt består av forslag til datalagring, nøkkellagring, nøkkelgenerering hos produsent og autentisering uavhengig av sensorprodusentens sky samtidig som klientens sensordata forblir privat. Vi diskuterer også ulike lovende lettvektige krypteringsprimitiver som kan bli anvendt for kommunikasjon i et slik brukstilfelle.

Resultatene våre viser at det er mulig å legge til en lokal lagringsløsning til en allerede eksisterende sensor-til-skyprotokoll samtidig som sensordata kun er tilgengelig for klienten. Lagring er ikke en stor bekymring, men ulike muligheter bringer opp ulike sikkerhetsbekymringer. Autentiseringsalternativer vil alltid kreve et visst nivå av tillit mellom sensorprodusenter og klienter siden produsenten ikke kan fullstendig frikjenne seg fra nøkkelgenerering og -utveksling. Vi har også funnet lovende lettvektige krypteringsprimitiver som kan bidra til disse løsningene og gjøre dem mer sikre.

Preface

This research was undertaken at the Department of Information Security and Communication Technology at the Norwegian University of Science and Technology (NTNU) as the final Master's Thesis for Théau Giraud's Master of Science in Communication Technology - Information Security.

The subject of the thesis was a joint idea between NTNU and Disruptive Technologies. The work was supervised by Colin Boyd, who was the responsible professor at NTNU and Håvard Skara Mellbye, the co-supervisor working for Disruptive Technologies.

Acknowledgments

I would like to personally thank Disruptive Technologies for giving me the opportunity to get to know their protocol stack, interact with their engineers and letting me work on location with them. I would also like to thank Sigve Tjora who was my original co-supervisor on this thesis. I want to thank Colin Boyd for his very helpful insights about which directions to take when writing this paper. I particularly want to thank Håvard Skara Mellbye for his highly helpful insights which helped me re-think the thesis when I was stuck. And a big thank you to Nora Lien Røneid for helping me with the abstract in Norwegian.

I also want to give a big thanks to all my professors, both at NTNU and Institut National des Sciences Appliquées de Toulouse (INSA Toulouse), where I started my studies and who allowed me to come to Norway for the joint-Master's degree I am currently finishing.

I want to thank all the friends I have made at Trondheim this semester, with a special mention to all my teammates at NTNUI Rugby for helping me forget about the stress of writing a thesis by tackling me at training at the end of the day. I especially want to thank my girlfriend, Siri, for her support.

I want to thank my parents and sisters for reminding me where I come from, for making me laugh, and especially for sending me food from France. I wouldn't have made it without you.

Lastly, I want to thank Nahom Aseged, my desk mate and amazing friend without whom this whole Norwegian chapter of my studies would have been much bleaker. Thank you for discussing the thesis with me, for constantly laughing at my bad jokes and for all the good times.

Contents

List of Figures

List of Tables

List of Acronyms

AEAD Authenticated Encryption with Associated Data.

AES Advanced Encryption Standard.

APE Authenticated Permutation-based Encryption.

API Application Programming Interface.

CVE Common Vulnerabilities and Exposures [CVE].

DAS Direct Attached Storage.

DDoS Distributed Denial of Service.

DES Data Encryption Standard.

DH Diffie-Hellman.

DoS Denial of Service.

DT Disruptive Technologies [Dis].

ECC Elliptic Curve Cryptography.

ECDH Elliptic Curve Diffie-Hellman.

FIPS Federal Information Processing Standards [NIS].

GSM Global System for Mobile communications.

HSM Hardware Security Module.

ID Identifier.

IDS Intrusion Detection System.

INSA Toulouse Institut National des Sciences Appliquées de Toulouse.

IoT Internet of Things.

IP Internet Protocol.

KDF Key Derivation Function.

KEK Key Encryption Key.

LAN Local Area Network.

LFSR Linear-Feedback Shift Register.

MAC Message Authentication Code.

MiTM Man-in-The-Middle.

NAS Network Attached Storage.

NIST National Institute of Standards and Technology [Kis13].

NRO Non-Repudiation of Origin.

NRR Non-Repudiation of Receipt.

NTNU Norwegian University of Science and Technology.

OSI Open Systems Interconnection.

P2P Peer-to-Peer.

PRNG Pseudo-Random Number Generator.

RAM Random Access Memory.

RSA Rivest-Shamir-Adleman.

SAN Storage Area Network.

SRAM Static Random Access Memory.

STRIDE Spoofing - Tampering - Repudiation - Information Disclosure - Denial of Service - Elevation of Privilege.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

Chapter 1 Introduction

The Internet of Things (IoT) has become an increasingly important part of our day-to-day lives. Today, we can count over 12 million connected devices in the world [Sin21]. This includes connected lights, cars, doors, health devices and many others. Many of these devices operate using sensors (possibly built-in sensors) and operate by adapting according to the data they collect from their surroundings.

With many systems and devices today so heavily reliant upon sensor data, a natural question to ask is: “Can we trust this sensor data?”. For example, if an attacker can tamper with a heat sensor’s data, they could stop the connected sprinkler system from going off in case of a fire. This is why implementing secure communications between IoT devices is so important.

1.1 Motivation

Figure 1.1 shows the typical network architecture for sensor data usage. Sensor data will most often be stored in a cloud. We will be focusing on the following use case; a client orders sensors from a company manufacturing sensors which communicate with the sensor manufacturer’s cloud for storage.

In our use case, only the cloud possesses the private key needed for decryption of sensor data. The client must trust the sensor manufacturer to securely handle and transmit that data. This is an ever-recurring problem in security; you can either implement things yourself or trust in the people who made the product you are using [Tho84].

But some clients would rather keep their sensor data private from the sensor manufacturer, as the manufacturer is the one who sets up authentication from the sensor to the cloud and is able to encrypt and decrypt that data. Their sensors may be used in critical services such as the health or military sectors. Clients such as these would rather store their data using an on-premises solution.

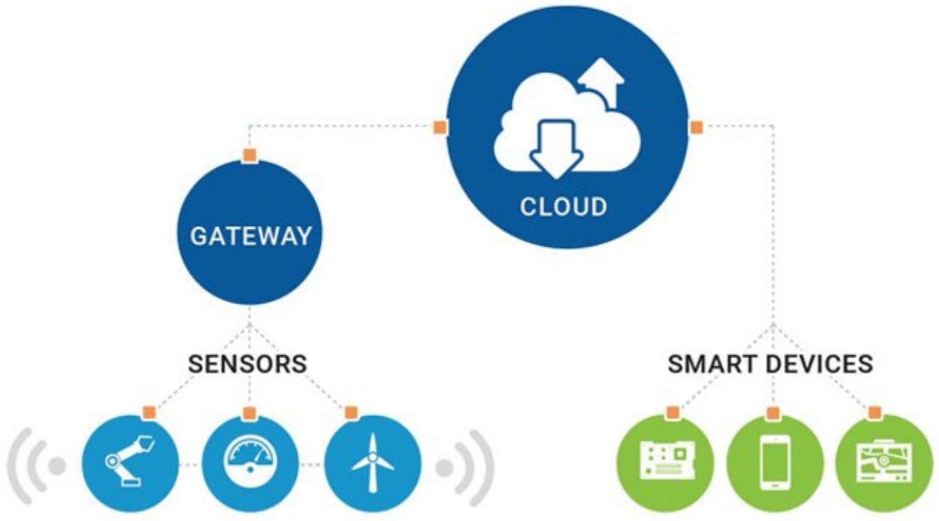

Figure 1.1: Typical sensor-to-cloud configuration

Other motivations for on-premises sensor data storage may be that clients want to keep a back-up of data stored on the cloud if servers go down or if internet access is disabled. They may also have their own private network set up according to their own company's protocol and do not want to have to communicate with the sensor manufacturer's cloud. The expected outcomes of using this alternative storage are to increase security by having data storage be controlled without the need for outwards connection, as well as less demanding communication operations as sensors will communicate directly with on-site devices.

Security of on-premises storage and transmission of sensor data to that storage will be the main focus of this thesis. We will compare the different approaches a company can take when manufacturing sensors for a client who wants their data kept private from the manufacturer. More specifically, we will be looking at the requirements needed to make such a solution work in terms of network architecture, storage options, key distribution, key storage and encryption of data in terms of security, feasibility and user-friendliness.

1.2 Context

The basic network architecture we will be working with can be seen in Figure 1.2. The user's sensor data is kept private using the storage option of their choice, which will be further detailed in Chapter 3. The public data going to the sensor

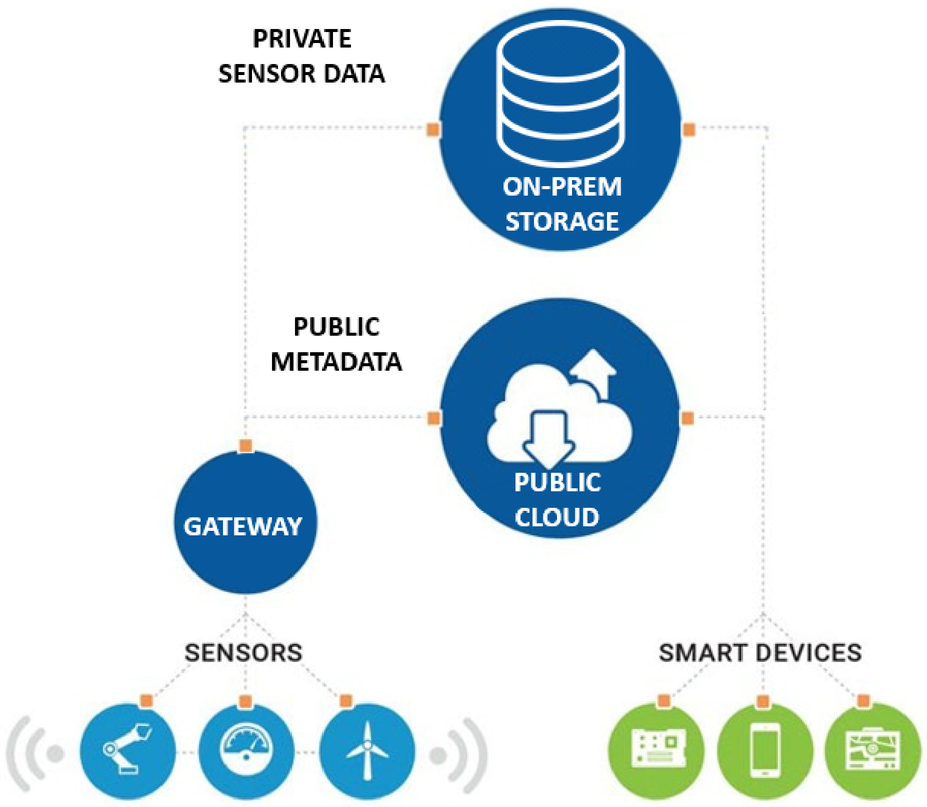

Figure 1.2: Combined cloud & on-premises storage configuration

manufacturer is the sensor's Identifier (ID), some protocol-specific data needed by the manufacturer to ensure proper interaction with the cloud or for sensor maintenance.

Private data in this case means only the client has access to it while **public data** means the client and the sensor manufacturer can access it. The private data is the encrypted data that the sensor periodically collects, such as the room temperature for a heat sensor. The public data should not allow the cloud provider to decrypt private sensor data.

In Figure 1.2, the **gateway** is a device whose main function is to forward packets from the sensor to its end point. Depending on the packet's destination address, it will forward it either to the cloud or the on-premises storage. The **smart devices** in this use case are any devices that rely on sensor data to function. Communications to smart devices is out-of-scope for this thesis, as we want to focus on working with lightweight sensors and the authentication problem.

1.2.1 Disruptive Technologies context

The idea for this thesis came from Disruptive Technologies [Dis] (DT) who manufacture ultra lightweight sensors communicating with their own cloud. These include temperature, pressure, proximity and motion sensors. Some of their clients have expressed interest in being able to store their sensor data on-premises, resulting in this research topic.

Their sensors currently run on a CC1350 chip [Ins]. It is a 32-bit ARM microcontroller with 128 kB of Flash memory and 28 kB of Random Access Memory (RAM). It is chosen for the wide temperature ranges at which it can operate, as DT aims to make very sturdy sensors. These specifications are what we will be basing our analysis on for further discussion.

We will thus be focusing on extremely small sensors, which have very low computing power and limit the scale of security that can be achieved. They are not capable of implementing certain resource-demanding protocols, such as Transport Layer Security (TLS) and take time performing demanding computations. They are also incapable of sending data packets over long distances which explains the need for a gateway to relay information.

1.3 Objectives

The main objectives that arise in this use case are:

– What on-premises storage option or on-premises network architecture are safer for data security and privacy,

– How can the sensor and the on-premises storage mutually authenticate when the shared secret used for sensor-to-cloud communications is shared during the manufacturing phase,

– How to securely encrypt sensor-to-on-premises-storage communications given the low level capabilities of the sensor, and doing so in a user-friendly fashion.

The research questions we have defined from our objectives are as follows:

– **What are the security requirements for an on-premises alternative to cloud data storage?**

– **What possible on-premises storage options or on-premises network architectures are more suited to safe user-friendly sensor data storage?**

– **What possible key exchange solutions exist to authenticate sensors to on-premises storage, as an alternative to an existing sensor-to-cloud protocol?**

– **What possible user-friendly low-level encryption protocols exist to protect sensor data?**

1.4 Research methodology

The aim of this work is to produce a state-of-the-art solution for anyone who would wish to implement safe on-premises storage for sensors that originally communicate with a cloud. To answer these research questions, we will take the following approach;

in Chapter 2, we will define the required notions for this thesis, in terms of security and cryptography, as well as relevant work. In Chapter 3, we will discuss different on-premises storage alternatives and their trade-offs. In Chapter 4, we will perform a threat analysis of the proposed network architecture seen in Figure 1.2 in order to discuss on-premises storage alternatives. From this threat analysis, Chapter 5 will discuss the mutual authentication problem, encryption possibilities and key storage regarding the security of sensor data. Chapter 6 will make a comparison of different lightweight encryption protocols to accompany the methods discussed in Chapter 5.

Chapter 2
Background & Related Work

In this chapter, we will introduce the notions and notations required for the rest of this thesis. Security principles, cryptography terms and network-centric terms are needed to better understand our work. This chapter will be referenced every time an underlying notion is required to understand a proposed implementation.

2.1 Information security background

Cybersecurity and information security have evolved extremely fast in the last 50 years. Since the arrival of the internet, the amount of cyber threats has increased rapidly [AGGL22] and a need to define these threats has grown.

2.1.1 Principles of information security

The first time security principles were mentioned is in an National Institute of Standards and Technology [Kis13] (NIST) publication from 1977 [RM77]. In computer networks and cryptography, the key principles that must be followed in order to implement a secure system are defined as such by the NIST glossary [PB19]:

- **Authenticity** is about being able to identify users and entities when establishing communications. In order to have safe communications, you need a way to tell if users are who they say they are. This is often done using a pair of credentials, such as a username and password, or in the case of cryptography by using a pre-shared key or pair of keys.
- **Integrity** of information is being able to trust that that information has not been tampered with. If the received information is the original information that was sent, then the communication channel kept the integrity of the message. This is often checked by the use of a Message Authentication Code (MAC), which we will elaborate on in Section 2.2.4.

- **Non-repudiation** Non-Repudiation of Receipt (NRR) prevents a user from denying having received a message and lying about it. Non-Repudiation of Origin (NRO) prevents a user from lying about the source of its information. A legitimate user could actually have not received something, and this is a problem that must be addressed (and should be covered by authenticity and integrity, which are pre-requisites for non-repudiation), but a lying user saying they did not get the original message must not have the opportunity to be sent the message another time.
- **Confidentiality** is the degree of secrecy of the information. Only the sender and receiver of a message should have access to that message unless they choose to give access to other users. This is not to be confused with integrity, as a message can be sniffed (meaning an attacker successfully got the contents of the message) but not tampered with.
- **Availability** of resources is about having the information you are using be available to authorized parties at all times. A system should have sufficient availability if it is to be useful.
- **Authorization** is about the level of access a user can have once authentication is achieved. The information that will be displayed to a user should be dependent on their degree of authorization. This is done through the use of administrative policies for example.

2.1.2 Classification of attacks

There are many different types of cyber attacks that exist and have grown in popularity as computer network usage has grown. Attacks may be known under different names, but there is a recognised catalog of vulnerabilities called Common Vulnerabilities and Exposures [CVE] (CVE) that can be found online. We will only cite the most common form of vulnerabilities that are relevant to IoT devices.

Spoofing

Spoofing is the act of passing as a valid user or entity in order to gain access to a system or information which the attacker should not have access to. For example, a successful Internet Protocol (IP) spoofing attack where an attacker alters the source IP address in their network packet in order to pass as trusted IP address for a certain system.

Denial of Service

A Denial of Service (DoS) attack is the exact opposite of availability, as it aims to make services and information unavailable to legitimate users. A very common

Distributed Denial of Service (DDoS) attack is IP spoofing from many different points, as multiple incoming IP addresses can not be blocked all at the same time.

Backdoor

A backdoor in a system is a secret infiltration method that allows an attacker to gain access to a system or network unseen. Backdoors may exist as a part of the system's design for admin use but could be exploited by attackers. Manufacturers can also create their own, for various reasons, but it is a very risky implementation. Backdoors are very dangerous because they can allow complete undetected access to all resources in a network or system.

Eavesdropping

Eavesdropping, or sniffing, is the act of listening in on a communication between two parties in a network. Examples include electromagnetic sniffing attacks, where an attacker can observe electromagnetic transmissions generated by a device to gain information on the processes being run or Man-in-The-Middle (MiTM) attacks, where an attacker can capture packets sent between two parties and read the data before sending it along to avoid detection.

Elevation of Privilege

Elevation of privilege is any situation where an attacker gains access to data or operations they initially were not allowed to access. The most coveted elevation of privilege is to become the root user in a system, thus gaining full access to a system.

2.2 Cryptography background

Cryptography in communications is the science of encryption (altering data so it is unreadable by an outside source) and decryption (decyphering, or recovering that data from its encrypted state). It enables two parties to communicate privately without having an outside party or attacker decrypt their messages. Many different methods of encrypting communications exist, but they all follow these same three steps [KL14]:

- **Auth** (Authentication): process of validating a communicating party's identity through the use of a pre-shared secret
- **Gen** (Generation): probabilistic algorithm that generates one or multiple keys used by *Enc* and *Dec*
- **Enc** (Encryption): probabilistic or deterministic algorithm that receives a key K and a message m as input and outputs a cyphertext c

– **Dec** (Decryption): deterministic algorithm that receives a cyphertext c and a key K as input and outputs a message m

2.2.1 Cyphers

A cypher is an algorithm implementing cryptographic principles for encryption and decryption. It is deployed on the two end-points of a communication. The two broad categories of cypher families are **symmetric cyphers** and **asymmetric cyphers**.

Symmetric Cyphers

Symmetric cyphers are so named because both entities use the same key for encryption. It relies on both communicating parties already having that key available to them, meaning it needs to be delivered through safe channels. Two historical examples of symmetric cyphers are the Data Encryption Standard (DES) and Advanced Encryption Standard (AES) which were standardised by the Federal Information Processing Standards [NIS] (FIPS), now an established source for encryption standards.

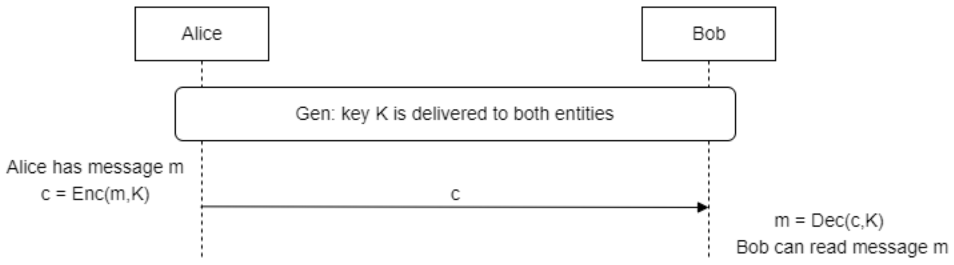

Figure 2.1: Symmetric cypher encryption

Asymmetric Cyphers

For asymmetric cyphers, each communicating entity has its own pair of keys, one public and one private. The public key is known to any and all parties on the network, while the private key is only known to its owner. To send a message you encrypt it using the receiver's public key which you have access to and they will decrypt it using their private key, as shown in Figure 2.2.

To illustrate this concept, we will give the example of Elliptic Curve Cryptography (ECC). ECC is one of the most common applications of asymmetric cryptography and allows for shorter keys than the classic Rivest-Shamir-Adleman (RSA) encryption. It relies on the fact that solving the elliptic curve discrete logarithmic problem

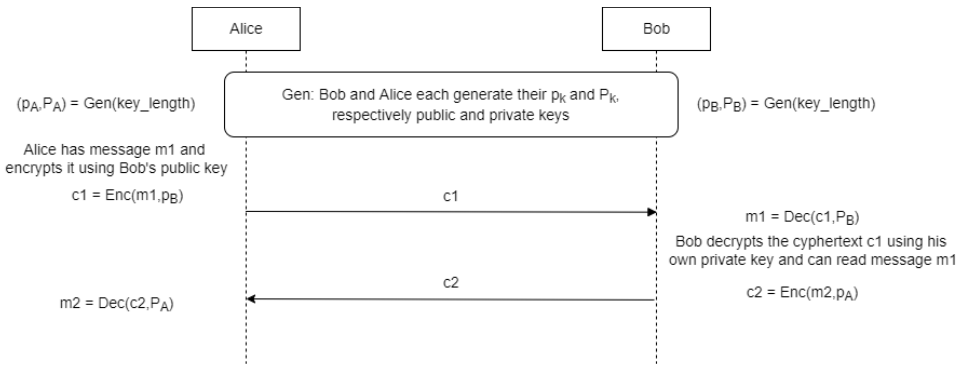
Alice
Bob
Gen: Bob and Alice each generate their p_k and P_k, respectively public and private keys
(p_A,P_A) = Gen(key_length)
(p_B,P_B) = Gen(key_length)
Alice has message m1 and encrypts it using Bob's public key
c1 = Enc(m1,p_B)
c1
m1 = Dec(c1,P_B)
Bob decrypts the cyphertext c1 using his own private key and can read message m1
c2
c2 = Enc(m2,p_A)
m2 = Dec(c2,P_A)

Figure 2.2: Asymmetric cypher encryption

is infeasible. It is stated as such; ”In the multiplicative group *Zp**, the discrete logarithm problem is: given elements r and q of the group, and a prime p, find a number k such that r = qk mod p” [TGP21]. In the case of ECC, k is the private key an attacker is trying to compute, given r the cyphertext and q the message. It takes an extremely high amount of calculations to recover k the private key, which makes ECC secure.

2.2.2 Hash functions

A hash function is a one-way function used to map data of arbitrary size to fixed-size values. Hashes are often used in cryptographic cyphers to compress cyphertexts during the encryption phase. These functions are deterministic and un-keyed which makes them very useful for performing integrity checks on the encrypted data. Hash functions are collision-resistant, meaning that given an output, it is computationally infeasible to find two different inputs giving that same output. This means that you can check the hash value received with the expected hash value, confirming whether or not data has been tampered with.

2.2.3 Key management

Types of keys

There are many different names cryptographic keys can have in the steps used to encrypt and decrypt information. Here are the most important key types we will use in this thesis:

– **Master keys**: Keys that are used to derive other keys used for authentication, encryption and other cryptographic methods. They are mostly used to respect forward secrecy.

- **Long term keys**: Keys that are used for encryption of communications as long as that communication is active. Key Encryption Keys (KEKs) are a good example of long term key, as these are not renewed very often.
- **Session keys**: Keys that are only used for a single session of communications, meaning if communications are cut-off, a new session key is generated (either from a master key or through other means).
- **Authentication keys**: Keys used only for the authentication phase, to determine the authenticity of communicating parties.
- **Encryption keys**: Keys used only for encryption of data, like the keys obtained from a Diffie-Hellman (DH) handshake.

Key generation

Keys should be generated in a fashion such that attackers cannot reverse-engineer the generation process. A cryptographic protocol's implementation should be available to the public if it is to be considered secure, so key generation requires a form of random input at some level, otherwise no matter how complicated calculations are, an attacker can calculate the same keys in the same way that the target device originally did.

Key generation can be:

- **Fully random**, meaning that a key is randomly generated. This is the ideal implementation, but true randomness with computers has yet to be achieved, and randomizing calculations for a long key (particularly asymmetric key pairs) requires extensive calculations.
- **Pseudo-random**, through the use of a PRNG. A PRNG imitates the workings of a truly random generator but considerably reduces the number of output possibilities.
- **Pseudo-random**, through the use of a Key Derivation Function (KDF). A KDF is a one-way function that takes as input a master key and combines randomness to produce new keys used for other cryptographic purposes. A KDF validate forward-secrecy.

A **PRNG** takes an input of fixed size λ and gives an output of size n*λ ($n = 2$ in Figure 2.3). This means that although the output domain size is n times bigger than the input domain size, the actual number of outputs is much lesser than that of a

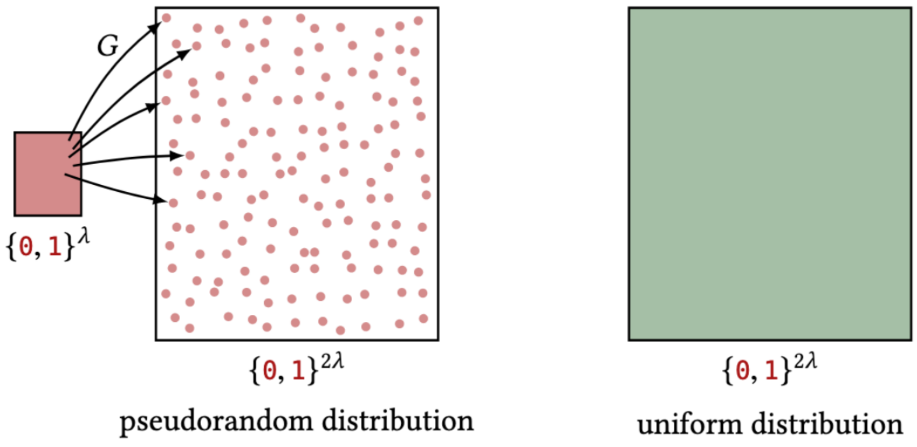

Figure 2.3: PRNG output distribution, taken from Rosulek's *joyofcryptography* [Ros]

true random generator. A PRNG "maps" outputs to imitate a random distribution. To be considered a PRNG, the output must not be discernible from a random output. Figure 2.3 illustrates this.

Forward secrecy is a feature that a cryptographic system must have. It means that if an attacker can recover master keys, it will not be able to calculate session keys derived from that master key, and thus will not be able to decrypt data sent using those session keys. This is why a KDF is very important, as it introduces randomness into the making of session keys and validates forward secrecy for a cryptographic system.

Key exchange

A key exchange protocol is a mix between symmetric and asymmetric cryptography. It consists of two parties using public/private key pairs to agree on a common secret key used for further encryption. This common secret is established by both combining their private key and the other entity's public key to arrive to the same result. Since an entity's public and private keys are mathematically linked, this method works.

The most popular key exchange protocols are the DH and Elliptic Curve Diffie-Hellman (ECDH) key exchange protocols. The inner workings of DH are shown in Figure 2.4.

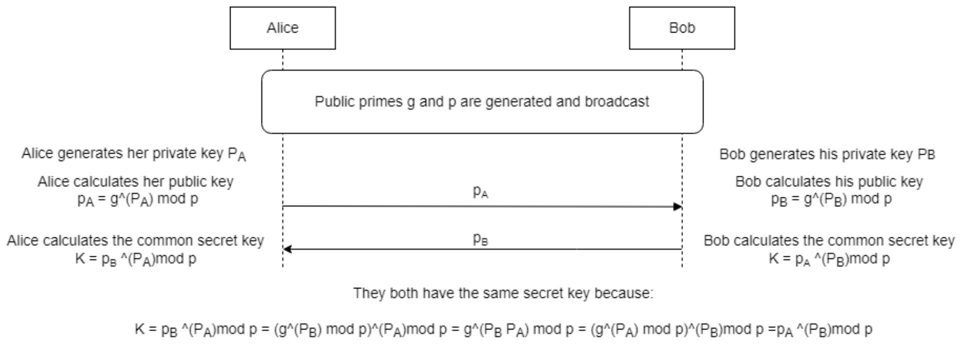
Alice
Bob
Public primes g and p are generated and broadcast
Alice generates her private key P_A
Alice calculates her public key p_A = g^(P_A) mod p
p_A
Bob generates his private key P_B
Bob calculates his public key p_B = g^(P_B) mod p
Alice calculates the common secret key K = p_B ^(P_A)mod p
p_B
Bob calculates the common secret key K = p_A ^(P_B)mod p
They both have the same secret key because:
K = p_B ^(P_A)mod p = (g^(P_B) mod p)^(P_A)mod p = g^(P_B P_A) mod p = (g^(P_A) mod p)^(P_B)mod p =p_A ^(P_B)mod p

Figure 2.4: Diffie Hellman key exchange

Both entities end up calculating the same secret key. An attacker cannot calculate this secret key, as being able to reverse engineer these calculations even with access to g, p, p_A and p_B takes an extremely long amount of time if g and p are large enough primes. ECDH relies on the same principle, but using ECC calculations instead to calculate the common secret key.

2.2.4 Authenticity & data integrity

Message Authentication Code

A MAC (or tag) is used to check the authenticity and integrity of data. MACs are computed by taking as input the message or cyphertext of the message depending on the implementation and possibly the encryption key. This value is then compared to a calculated MAC on the receiver's side to validate that the message has not been tampered with and does come from the right sender.

Authenticated Encryption with Associated Data (AEAD) can be done several ways:

- **Encrypt-then-MAC**: The plaintext is encrypted, then a MAC is generated using the cyphertext. They are both sent together. Then, upon reception the receiver calculates the MAC associated to the cyphertext and compares its value to the one received.

- **Encrypt-and-MAC**: The MAC is calculated using the plaintext, then the plaintext is encrypted without the MAC. The cyphertext and MAC are sent together. Upon reception the cyphertext is decrypted, a MAC is calculated using the decrypted message and compared to the received MAC.

– **MAC-then-Encrypt**: The MAC is calculated using the plaintext, then the plaintext is encrypted along with the MAC. They are both sent in encrypted form. Upon reception, the package is decrypted, a new MAC is calculated using the decrypted message and compared to the received MAC.

Nonce

A Nonce is a value used to distinguish between two messages encrypted using the same key in order to avoid replay attacks. Nonces must be random. For example, an attacker can recover the encryption key quite easily if Nonces are not updated [DRA16]. It is an extra security add-on to distinguish a message's integrity.

2.3 Network basics

2.3.1 OSI model

The Open Systems Interconnection (OSI) model, as shown in Figure 2.5, is the basis of computer networks. Packets are encapsulated in layers of protocols, each protocol taking care of referencing and orientating the packet for one of the layers. For example, the widely used IP protocol operates at the network layer and is used to reference a packet's destination in an IP network. The Transmission Control Protocol (TCP) operates at the transport layer and relies on the underlying protocol at the network layer (almost always IP, as the TCP/IP protocol is the most widespread) to itself encapsulate a packet and establish connections with distant hosts.

2.3.2 The TLS protocol

Today, the TLS protocol is the preferred security protocol at the transport layer. Mutual TLS ensures authenticity of hosts by having them check the validity of each others' TLS certificates. These digital certificates are proof that an entity is the owner of the public key being used for asymmetric communications. This certificate has been issued by a trusted third-party, and that party's authenticity is validated by the above party that issued its certificate, and so on, until we reach the first issuer of certificates which is a referenced trusted party.

Thanks to this signature verification, communications between end-points can be effected without a previous shared secret. TLS also supports many cryptographic protocols, meaning a device must implement code to treat each TLS extension. This is too heavy a task for the sensor's microcontrollers we are dealing with, which is why the easy-to-use TLS protocol cannot be implemented in our use case.

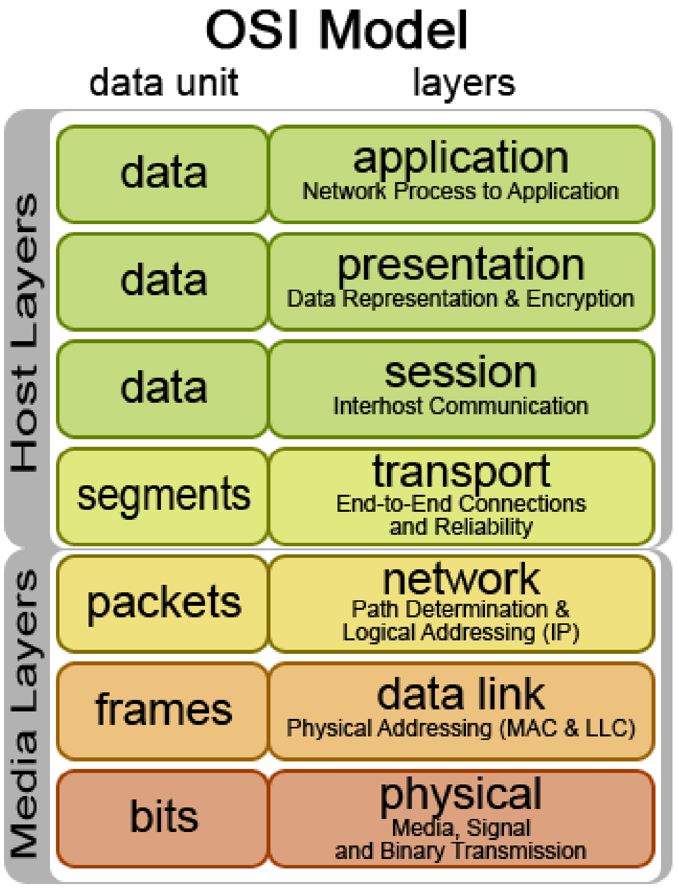
OSI Model
data unit
layers
Host Layers
Media Layers
data
application
Network Process to Application
data
presentation
Data Representation & Encryption
data
session
Interhost Communication
segments
transport
End-to-End Connections and Reliability
packets
network
Path Determination & Logical Addressing (IP)
frames
data link
Physical Addressing (MAC & LLC)
bits
physical
Media, Signal and Binary Transmission

Figure 2.5: The OSI model

2.3.3 Client-Server model

At the lowest level of data storage, servers are always involved. **Servers** store resources (in our case sensor data) and **clients** are the ones requesting those resources. There can be more than one client per server, and servers and clients can be on the same system, although in our use case, the server will be the on-premises storage and the client will be the gateway. The server sharing resources is called a service. The client-server model is a distributed application structure, meaning it is a way of looking at client-server communications as a partitioning of tasks between the servers and clients. What is important here is to remember this terminology of server and client.

2.3.4 Network security

Securing a network from connections of users with malicious intent is done mainly by filtering who can and cannot connect to your devices. **Firewalls** are the main means of filtering potentially dangerous connections. A firewall is a network monitoring system which lets you ”whitelist” and ”blacklist” IP or other addresses of malicious connections, types of connections depending on on the network protocols used, amount of successive connections, etc.

Another, more effective system is an **Intrusion Detection System (IDS)**. IDSs are specific devices or software dedicated to monitoring network traffic. They are more powerful than firewalls, in that they can monitor a large amount of devices and the policies that can be implemented using IDSs are much more sophisticated and can target specific types of attack patterns.

Chapter 3

On-premises Storage Options

Choosing the right storage for an on-premises solution is one of the cornerstones of this thesis. There are three types of on-premises storage [HMLY05]; **Direct Attached Storage (DAS)**, **Network Attached Storage (NAS)** and **Storage Area Network (SAN)**, shown in Figure 3.1, with **private cloud** being a hybrid approach. Following the client-server model mentioned in Section 2.3.3, in our use case the client is the gateway as it is the last connected device in the data path of sensor data (as shown in Figure 4.1). If another device like a switch or another gateway is used as the entry point to the storage option, then that device is the client.

3.1 Direct Attached Storage

The simplest of the three options is DAS, which is directly connected to a client via a **Peer-to-Peer (P2P) connection**. DAS offers low latency communications due to this proximity to the client and the lack of a connecting device (such as a switch). But this implementation also limits scalability and imposes a physical constraint as the DAS can not be established too far from the client.

In terms of security, we see in Section 4.4 that restraining the access to servers is primordial in keeping data secure. With DAS, this P2P connection to the server means that if a client is compromised then so is the server. Server set-up should only allow normal behaviour, meaning reading and writing of that client's sent data. The use of an IDS surveying the storage option should also be considered.

3.2 Network Attached Storage

NAS is accessed through a classic **Ethernet Local Area Network (LAN)**. The advantage NAS offers over DAS is that it can be deployed further from the client. It also allows for the interconnection of several NAS and competing access

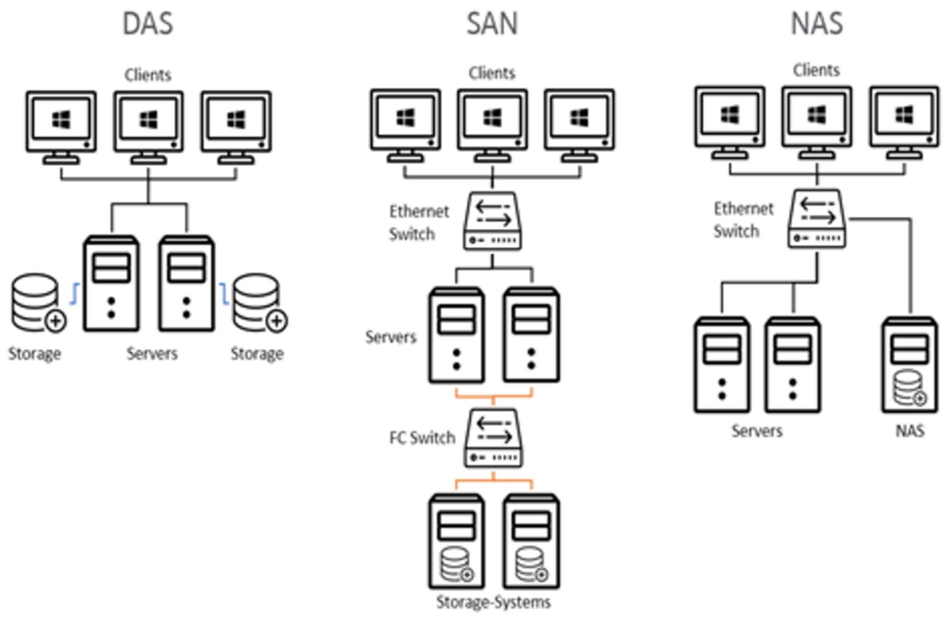

Figure 3.1: DAS, NAS and SAN storage options, adapted from Krenn's "Storage basics: DAS, NAS and SAN at a glance" [Kre]

from multiple clients (in our case there could be multiple gateways to deal with). A problem that one could encounter when using NAS is the high load being put on the LAN by adding the use of TCP/IP protocols which are not optimised for heavy storage traffic.

Once again, the access points to a NAS must be closely monitored. If the number of clients communicating with the storage increases, entry points also increase. An extra network firewall can also be added on the entry switch, further filtering traffic and decreasing malicious client interactions.

3.3 Storage Area Network

A SAN consists of storage devices having their **dedicated storage network**, typically Fibre Channel infrastructure. Client-side access is done through Ethernet but further communications within the SAN are done through Fibre Channel. The SAN allows for the connection of multiple storage options and thanks to Fibre Channel usage, transfer speeds can be compared to those of a DAS, but can cover greater distances which resolves the issue of scalability for DAS. Downfalls of SAN are the upkeep of the network as it requires more configuration and administrative tasks, as well as having to deal with an additional infrastructure on top of Ethernet.

Choosing a NAS or SAN over a DAS increases the amount of connections between nodes in the on-premises network, which increases the amount of possible entry points for attackers. Network security is primordial for keeping data safe. This means having a proper firewall set up at each entry point to filter malicious traffic. An even safer implementation would be to use an IDS covering the entry points to the network.

3.4 Private cloud

The use of a private cloud for data storage is also an option for the user. Private cloud consists of using **virtualized** computing resources using the company's physical on-premises hardware or a cloud service provider. In our use case, it is unlikely the client will use a cloud provider for data storage, as they already did not want to use the sensor manufacturer's cloud. This leaves using their own provisioned hardware to establish a cloud network and this option should only be considered if the client wishes to deploy a very high number of sensors. A hybrid approach could also be considered, as using a private cloud solution is more adaptable than having to remodel your storage layout every time you want to add more options.

Threats to physical storage are still relevant in this case, but a distributed cloud adds another layer of access to storage which makes the end server harder to reach, meaning more difficulty for the attacker. But it also brings about more network oriented threats, as access points are multiplied in a cloud environment. A private cloud is often expensive to set up, requiring more space and hardware than conventional storage options as well as having to virtualize the whole storage environment.

3.5 Data storage format

One would think that storing data in encrypted fashion is beneficial, but the client should focus more on clearly implementing authorization privileges. The first option for storing encrypted data is receiving encrypted data and immediately storing it, meaning that the client would need to store the decryption key that was used at the time data was received if they ever needed to decrypt that data. This is extremely unsafe, as keys should be properly deleted once they are no longer in use.

The second option consists of decrypting the data upon reception then using a separate encryption key only used for storage. This is feasible, but having to decrypt data every time you want to access it is not viable compared to trusting that access to that data is truly secure. **Implementing proper user access and a good firewall is more important than encrypting storage data**.

3.6 Discussion

We have seen our four on-premises storage option. Table 3.1 compares the trade-offs that have to be made for each solution, in terms of security, implementation costs, limitations and user-friendliness.

Storage type	**Security**	**Implementation pros**	**Implementation cons**
Direct Attached Storage	The direct connection from the client to the server means that compromising the client also means compromising the server. Access should be restricted to reading and writing of data.	This option offers low-latency communications due to the proximity of the client and server.	This option limits scalability and requires the client and server to be physically close to each other.
Network Attached Storage	More access points means more attack vectors. But the added switch on the entry point can be used to set up and extra network monitoring service.	This option is not limited by physical distance and allows the interconnection of several NAS as well as multiple client access.	This option does not offer higher latency communications than DAS and SAN
Storage Area Network	More access points means more attack vectors. But added switches to interconnect storage options can be used to set up and extra network monitoring service.	This option offers low-latency communications due to Fibre Channel usage, is not limited by physical distance and allows the interconnection of several networks storage options.	This option requires more upkeep.

Storage type	**Security**	**Implementation pros**	**Implementation cons**
Private cloud	Virtualization of the whole storage infrastructure makes access harder. But if the cloud is accessed through the an internet browser, this leads to a whole slew of new threats.	Virtualisation of the network makes management more user-friendly.	This option is more costly than the others because of virtualization.

Table 3.1: Comparison of on-premises storage options

No matter the storage choice, the user will have to deal with the physical storage unit's management interface. This can cause security flaws in the system as the interface creates one or more extra access points, which adds to the need for proper network security. These interfaces must also be password protected which leads to threats linked to password recovery as cited in Table 4.4.

Chapter 4

Network Architecture Threat Model

One of the main challenges in this thesis is identifying the possible threats to our network architecture. In this chapter, we will discuss the different threats IoT devices face in our context and what choices can be made to mitigate these threats.

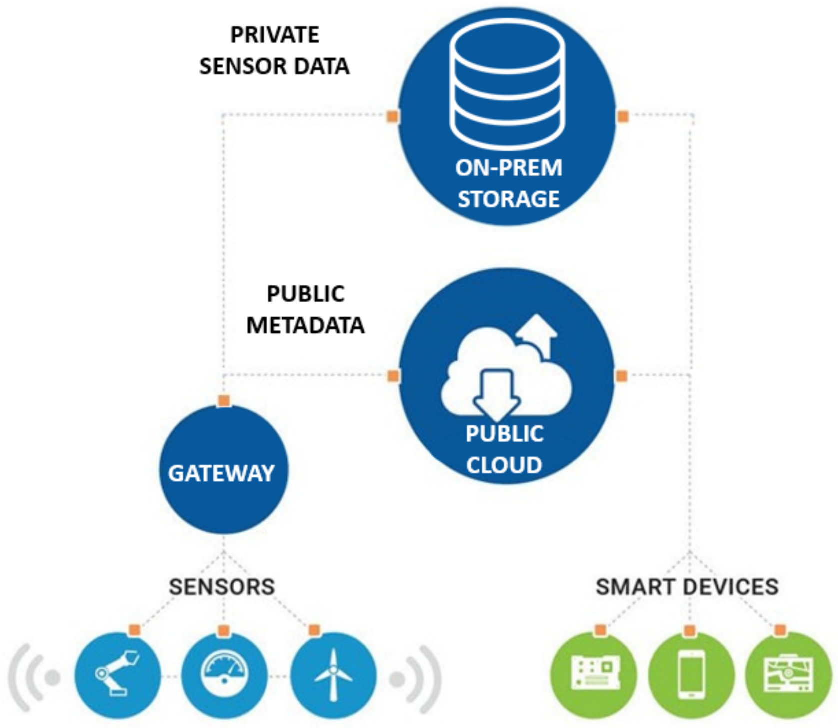

Figure 4.1: Combined cloud & on-premises storage configuration

We will base our analysis on Figure 4.1, the same figure as the one discussed in Section 1.2. It encompasses all possible data flows for sensor data in an on-premises solution. The amount of data going to the public cloud or the on-premises storage is up to the client. The separation of data will be further discussed in Section 5.5.

Each device will be viewed as a **singular node**, to consider what is needed at every level following the **Spoofing - Tampering - Repudiation - Information Disclosure - Denial of Service - Elevation of Privilege (STRIDE)** threat modeling method, as described in Chapter 3 of Threat Modeling by Shostack [Sho14]. Each threat is equivalent to a security property a system should have. Respectively to each threat in the STRIDE model these are; authentication, integrity, non-repudiation, confidentiality, availability and authorization (as described in Section 2.1.1.

We will consider each element of Figure 4.1 and its potential threats, using the **STRIDE-per-element** method, as described in page 78 of Threat Modeling [Sho14]. This method of STRIDE threat modeling consists in cataloguing threats for each device that can be found in the chosen network architecture. Threats will be ranked *Low*, *Medium* or *High* according to the potential system exposure or sensitive data access they can give an attacker. We will not, however, be looking at threats to the end devices as they are varied and depend more on the type of device.

4.1 Sensor Threat Model

We will begin our analysis with the sensors. They communicate solely with a gateway which acts as a transmitter as it has a stronger range and passes on the information to the next node (here being the public cloud or on-premises storage). There could be more than one gateway for the sensor to connect with and send the information to, so during the connection phase (which is occasionally renewed), a sensor will look for the gateway offering the best latency (among other factors) and pair with it. Sensors also have very little computing power, which limits them in some security parameters.

STRIDE Category	**Threat**	**Solution**	**Threat Level**
Spoofing	A spoofed machine could pretend to be the gateway during the pairing phase and communicate with the sensor.	Having a proper authentication phase upon connection absolves this threat.	Low

STRIDE Category	**Threat**	**Solution**	**Threat Level**
Tampering	An attacker with access to the sensor could potentially alter sensor data before it is sent out, making it give wrong readings.	Physical and backdoor access to the sensor must be limited - as we are focusing on lightweight sensors this is mitigated.	Low
Repudiation	A spoofed gateway can claim not to have received data from the sensor, with enough repetition this can cause a DoS.	Set a limit on re-emission of data (once a connection is secure).	Low
Information Disclosure	If an attacker can find the encryption keys used for communication in a sensor's storage they have total access to all further communications.	This depends on the safety of key storage chosen for the sensor, discussed in Section 5.6.	High
	An attacker can passively monitor power or CPU consumption on a sensor to gain information on what calculations are being done for cryptographic purposes, or for data being recorded and sent.	Implement randomised power processing to avoid power analysis attacks. This is harder on a lightweight device.	Medium
Denial of Service	An attacker can send a surge of requests to the sensor and monopolise its network resources by having it treat these endless useless requests.	Blacklist any address that tries this - although a lightweight sensor may not be fast enough to prevent such an attack.	Medium
	Physical damage to the sensor may render it unavailable to collect or send data	It is assumed that the sensor is not physically accessible to an attacker, although wear and tear may happen.	Low

STRIDE Category	**Threat**	**Solution**	**Threat Level**
Elevation of Privilege	Gaining full access to the sensor gives you access to unencrypted data. Even worse, an attacker could possibly falsify data with such access.	This again depends on gaining access to the sensor. A lightweight sensor makes accessing data harder, as instructions are more automatic.	Medium

Table 4.1: STRIDE threat model of sensors

It seems that with sensors, the issue is the level of access an attacker can attain on the device itself [SPA+21]. Network attacks such as ARP-spoofing depend on the device's packet filtering capabilities. Since we are focusing on more lightweight sensors, such as those used by DT, these types of network attacks can be more effective given that a sensor with lower computing power cannot match a stronger sensor's firewall capabilities. We will consider attacks that require full access to the sensor as out-of-scope, as a lot of tampering is needed to gain full access on such a miniature device.

4.2 Gateway Threat Model

Gateways act as transmitters for encrypted data between sensors and either the public cloud or the storage. Gateways also have internet access, compared to sensors. There may also be an extra layer of encryption added between the gateway and the destination point which strengthens encryption on that end if the protocol is safe enough.

STRIDE Category	**Threat**	**Solution**	**Threat Level**
Spoofing	A spoofed sensor could potentially connect with the gateway.	Having a proper authentication phase upon connection absolves this threat.	Low
	A spoofed sensor could pass as an existing sensor activating its re-connection phase.	The authentication phase also absolves this threat. Re-connection should also trigger authentication.	Low

STRIDE Category	**Threat**	**Solution**	**Threat Level**
	Having the right sensor data, one could bypass authentication with the gateway.	Implement a routine manual check-up of connected sensors to avoid this.	Medium
Tampering	With access to the gateway, an attacker can tamper with firewall setup.	Access to the gateway must be closely monitored and limited to only a few users.	Medium
	An attacker could alter sensor data passing through the gateway, rendering it unreadable or false.	Access to the gateway must be closely monitored and limited to only a few users.	Medium
Repudiation	A spoofed sensor or storage entity can claim not to have received data and trigger a DoS by having the gateway constantly resend packets.	Set a limit on re-emission of data (once a connection is secure).	Low
Information Disclosure	None. The gateway only transmits encrypted data.	None needed.	None
Denial of Service	Gateways are vulnerable to IP/ARP spoofing and any other network congestion attack just like sensors.	A strong firewall setup should prevent this type of attack. If not, an IDS will be more secure.	Low
Elevation of Privilege	An attacker that manages to gain total access to the gateway could potentially gain access to the rest of the system.	Having a strong firewall is a must, as well as adding an extra authentication phase to access certain capabilities of the sensor, like potential encryption keys between it and the cloud.	High

Table 4.2: STRIDE threat model of gateways

In the case of gateways, the threat of having access to sensor data is removed, unless an attacker can break the encryption protocol between the sensor and the storage. This is because a gateway only relays information, it is encrypted before being sent to the gateway. Gateways also have more processing power and better network protection capabilities than sensors, thus mitigating some threats they have in common.

4.3 Cloud Threat Model

Cloud networks are large networks distributed over multiple data centers. We will focus on public clouds here, to treat the pre-existing cloud solution offered by the sensor manufacturer in our use case, as shown in Figure 4.1. Note that if a client wanted to set up a private cloud as their on-premises option, the threats would be the same as on a public cloud, albeit a little scaled down due to the local aspect of the cloud, making credential attribution and data access the client's responsibility.

STRIDE Category	**Threat**	**Solution**	**Threat Level**
Spoofing	Unauthorised changes in admin privileges for certain users can lead to a security breach.	Closely monitoring and logging admin activities.	Medium
	Compromised login credentials (potentially obtained through brute force attacks) could be used to gain access.	Check login credentials for potential misuse; the same user logged in twice, a discarded user being used, etc.	Medium
Tampering	Any unauthorised alteration/deletion of logging data or alteration of logging policies can have consequences on later log needs for bugs.	Do not leave logging data in the clear.	Low
	An attacker can alter data or metadata being sent to the device.	Proper encryption absolves this problem.	Low
Repudiation	An attacker can attain access to an insecure cloud by claiming not to have user access.	This is easily avoided by applying the above principles.	Low
Information Disclosure	Unauthorised cloud outbound connections to malicious IP addresses.	Set up a strong firewall with white-listed addresses and monitor new connections closely.	Medium

STRIDE Category	**Threat**	**Solution**	**Threat Level**
	Unauthorized access to data or logging data.	Closely monitoring and logging admin privileges. Add an authentication step for access to unencrypted sensor data.	Medium
Denial of Service	Multiple simultaneous connection attempts can cause a DoS.	Once again strong firewall setup and/or IDS is needed against this but should suffice.	Low
Elevation of Privilege	Unauthorised change of privilege for users can be dangerous.	Once again, closely monitoring and logging admin activities avoids this.	Low

Table 4.3: STRIDE threat model of public clouds

As we can see from Table 4.3, threats to the cloud can come from outside of the cloud network, as with the sensors and gateways, but also from inside. Users with malicious intent or given too much privilege can be a danger [UVSL18].

Threats to the sensor manufacturer's public cloud are not our concern here. We will focus on threats that apply to the on-premises storage, but included the analysis as that storage may be a private cloud. If so, user privilege and admin privilege must be closely monitored and the cloud's outbound connections must be limited to only certain connections needed for a production cloud environment, like access to software package repositories.

4.4 Server Threat Model

In this section we will look at potential threats to a server, which are the physical backbone of any on-premises storage choice (see Chapter 3). Servers are hardware devices (although they can be digitised on a computer) that offer services to clients, including data storage. They are often quite powerful and managed remotely.

STRIDE Category	**Threat**	**Solution**	**Threat Level**
Spoofing	An attacker can infiltrate the storage with stolen credentials or by brute forcing them.	Closely monitoring and logging admin activities.	Medium
Tampering	An attacker can alter data or metadata being sent to the device.	Proper encryption absolves this problem.	Low
Repudiation	An attacker could potentially falsify an end device's request for sensor data.	Proper authentication and firewall setup rectifies this.	Low
Information Disclosure	If an attacker can gain access to deallocated memory or deleted storage blocks, there may be some accessible data if the deletion was not done properly.	Protect all memory and storage blocks and delete after deallocation.	Low
	Storage traffic can be sniffed through power analysis.	It is assumed that the server is physically inaccessible to attackers.	Low
	If an attacker can snoop on the buffer caches used by the storage device, they have access to the data.	Make sensor data the only accessible thing and set up good network protection.	Medium
Denial of Service	If an attacker can tamper with the proper functioning of the OS or kernel (through subversion attacks) they can cause damage to the data.	Limit access to the server and log divergent operations.	Medium
	Physical damage to the storage will render it useless.	It is assumed that the server is physically inaccessible to attackers.	Low
	An attacker with network access to storage can cause a DoS through network congestion.	Strong firewall setup and/or IDS is needed against this but should suffice.	Low

STRIDE Category	**Threat**	**Solution**	**Threat Level**
	Disabling the storage's power input renders it useless.	It is assumed that the server is physically inaccessible to attackers.	Low
Elevation of privilege	If an attacker can gain access to access credentials for the storage device, they will have access to all data.	Make password protection or encryption too hard to compute and do not store passwords on the network.	High

Table 4.4: STRIDE threat model of storage servers

As long as a server is physically inaccessible to attackers, the most dangerous threats (such as access to data, being able to tamper with data, tampering with the server's inner operations) can be avoided. Admin privileges must also be closely monitored and a strong firewall must be set up around the access points to storage to avoid fraudulent access to servers, no matter the storage configuration chosen.

4.5 Discussion

As we have seen from our threat models, each node of the sensor data path has its own particular set of threats that must be protected against. But there are some threats common to all these devices that seem to come up often; **unauthorised data access** and **unauthorised connection to the devices**. These issues can be dealt with by implementing proper authentication and encryption between nodes. Key exchange and key storage are very important aspects of securing communications from sensor to storage and will be further expanded upon in Chapter 5.

Chapter 5
Key Management

As was previously stated in Chapter 4, the main threats that the connected devices of our use case face are unauthorised access to devices and data confidentiality. These threats can be respectively circumvented through the implementation of proper authentication (giving authenticity to the transferred data) upon connection and encryption (giving confidentiality to the data) across communication channels.

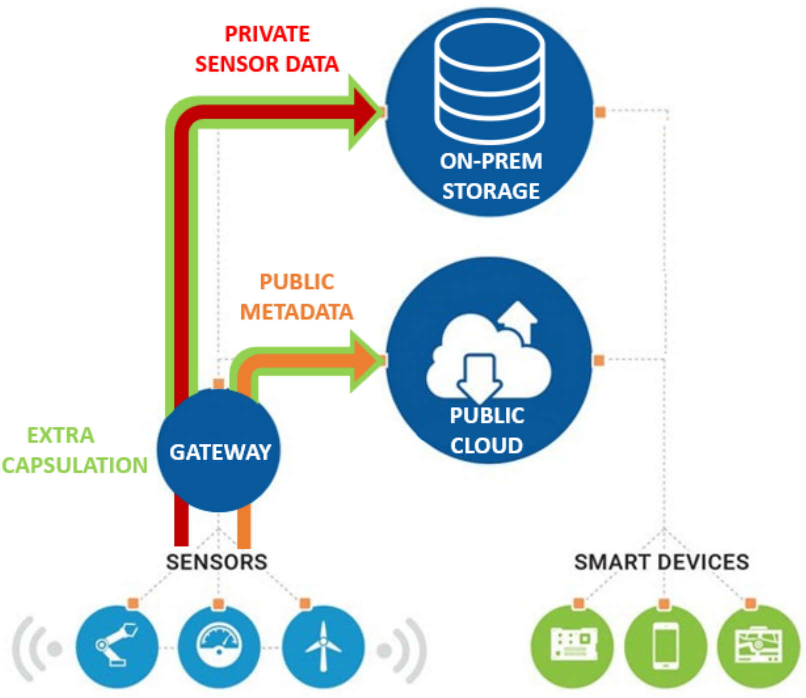

Figure 5.1: Sensor data path in IoT environment

We will be focusing on the encryption between the sensors and the on-premises storage, which is represented by the red arrow in Figure 5.1. The metadata sent to the cloud is to be encrypted and transferred according to the sensor manufacturer's preferences.

Communication on the smart devices' side is out-of-scope here. We will not discuss re-transmission of stored data, although the protocols proposed for sensor-to-storage and sensor-to-cloud communication can be adapted for communication with end devices, as they have much more computing and communication capabilities than lightweight sensors.

Sensor data should only be accessible to the two end points of the communication; the sensor and the storage. First, to ensure secure communications, the sensor must authenticate itself to the storage and vice-versa, with the gateway acting as a middle-man transferring messages from one to the other. During the authentication phase, the end points will synchronise keys (more details in Section 5.4). These keys will be session keys, as discussed in Section 2.2.3. Once proper authentication is achieved, the communication channel established is considered secure. The session key(s) will then be used to encrypt all messages between the sensor and storage to ensure confidentiality.

The extra green arrow in Figure 5.1 represents the possible use of an extra network protocol to transfer data from the gateway to the on-premises storage or the public cloud, like the use of TLS encryption or HTTPS Post requests. This extra protocol does not affect in any way encryption between the sensor and the storage, it is only an added encapsulation layer (as mentioned in Section 2.3) to ensure secure communications on this channel.

5.1 Authentication without a pre-shared secret

A lightweight sensor does not possess the required computing power to support the whole TLS protocol, which is one of the most common ways to establish secure communications among IoT devices. TLS's specifications require end devices to support many different cryptographic primitives and this means a lot of code space required to implement different support mechanisms of TLS use cases.

Nonetheless, authenticity is required between the two end-points. As we have seen in Chapter 4, there are many ways an attacker can falsify their identity, and this must be prevented. Authentication must be done through some other means here. It must also be renewed at regular intervals (the length of those intervals is up to the user) in order to avoid key recovery attacks.

One solution is to let the client run the key generation according to the specifications given by the sensor manufacturer, using a method relying on a PRNG. In this case there will be a drop in user friendliness during the authentication phase because the client will have to manually connect each sensor, running the authentication protocols specified by the manufacturer.

This option, however, does not satisfy authenticity of the end-points. There must be a pre-shared secret between the two sensor and storage in order for them to properly identify each other.

5.2 Pre-shared secret proposition: the master key

Another proposition, and our preferred mode of operation would be to embed a **master key** onto the sensor during manufacturing. This master key would be a random string of required length for further derivation of other keys used for authentication and encryption.

This master key could be used for symmetric or asymmetric authentication and encryption. In the case of symmetric cryptography, the master key would be the secret key, and be pre-shared between both entities (as discussed in Section 5.3.1). In the case of asymmetric cryptography, the master key would be the private key and the public key would be derived from that private key, in an RSA-like fashion. The private and public key pair would then be used for further DH exchanges as well as asymmetric key encryption. This would also require that the same practices be implemented storage-side.

5.2.1 Master key generation

The master key must be a randomly or pseudo-randomly generated key of suitable size for the client's security preferences. This key must be un-recoverable through analysis of the protocol design and kept secret as all other keys will be derived from it.

There are a two options when it comes to generating further encryption keys (this includes the public key if we are in the asymmetric cryptography option):

– Using a **KDF** to produce further keys, meaning the master key plays a role in the making of the keys but forward secrecy is respected.

– Producing random keys using a **PRNG**, which is more costly in computing power, and makes the master key obsolete.

The **first option** is preferred. Using a KDF is faster and cheaper and does not give any information on the master key from the derived keys and vice-versa.

This allows us to achieve forward secrecy in the possible implementation of a DH handshake.

5.2.2 Authentication

To achieve authentication, there needs to be a **pre-shared secret** between the sensor and storage. Using the master key, our idea is to have the sensor's public key be known to the storage and vice-versa before deployment of the sensor. In the case of symmetric key encryption, the master key and the secret key are one and the same, and this key is pre-shared between the two entities. That way, when communications begin, they can both broadcast their public keys and recognise that they are dealing with the right device on both ends.

5.2.3 Manufacturing & master key exchange

How do we pre-share the public keys in a secure way? One option would be to do it **during the manufacturing process**. First generate the master key and flash it onto the sensor. Following the asymmetric option, derive the public key from it then give that public key to the client along with the sensor. On the other side, the client must also generate a public key and share it so the sensor can store its value which will allow it to make that first connection to the storage. In the symmetric option, the master (secret) key is being shared with the client.

This solution could be developed into an adaptable Application Programming Interface (API) that the client could deploy onto their storage that allows them to reference the storage's public key from the asymmetric key pair used for connection but that may be a resource-heavy task for the manufacturer to implement.

5.2.4 Authentication through the cloud

Another authentication solution would be to **use the pre-existing authentication protocol between the sensor and the cloud** to share this public key or secret key. By using the same connection on the storage's side, one could also have the storage connect to the cloud to establish mutual authentication on either side, thus creating a safe channel through which the sensor can communicate the sensor's public key to the storage and vice-versa (or send the secret key in the case of symmetric cryptography).

After this step, the connection through the cloud is dropped and the sensor and storage can authenticate now knowing each other's public keys. After this phase, communications can ensue, with the steps further developed in Section 5.3 and Section 5.4. This sharing of a common secret is shown in Figure 5.2

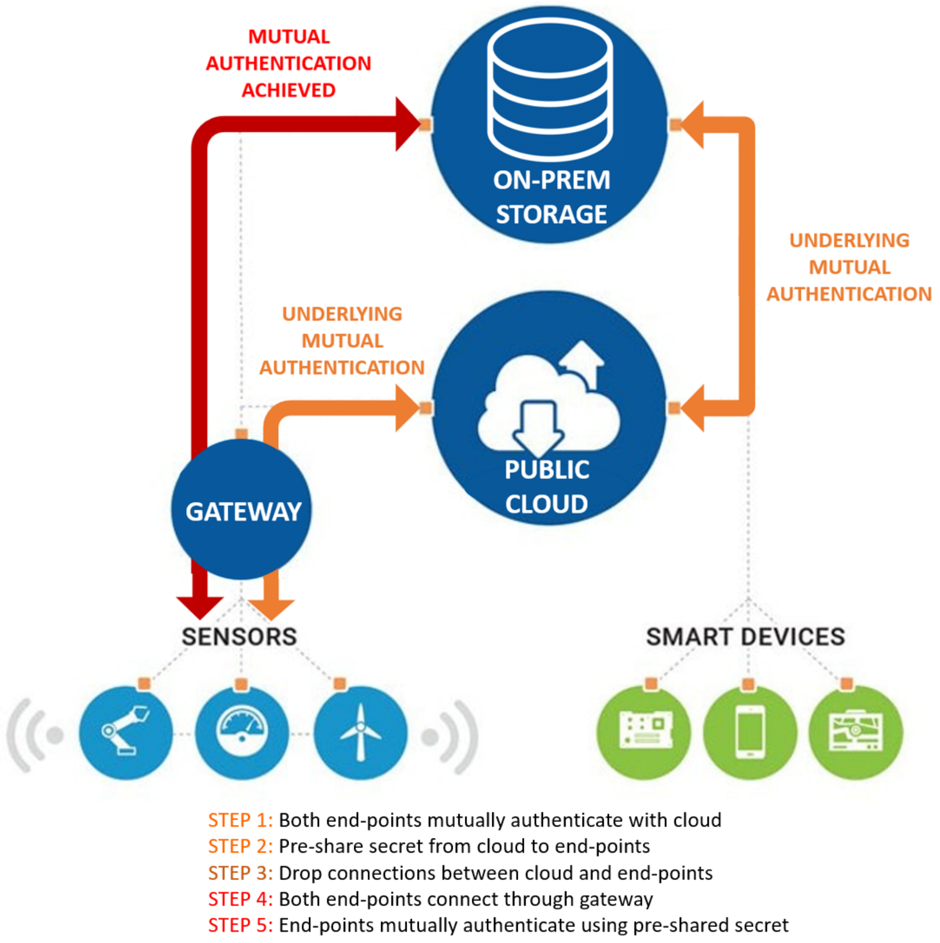
MUTUAL AUTHENTICATION ACHIEVED
ON-PREM STORAGE
UNDERLYING MUTUAL AUTHENTICATION
UNDERLYING MUTUAL AUTHENTICATION
PUBLIC CLOUD
GATEWAY
SENSORS
SMART DEVICES
STEP 1: Both end-points mutually authenticate with cloud
STEP 2: Pre-share secret from cloud to end-points
STEP 3: Drop connections between cloud and end-points
STEP 4: Both end-points connect through gateway
STEP 5: End-points mutually authenticate using pre-shared secret

Figure 5.2: Pre-shared secret through the cloud

This would allow for the return to classic cloud storage if the user ever wished for it or could be used as a backup storage for the cloud, as mentioned in Section 1.1. Also, if the user chooses to store data types on different storage devices, this option would enable us to differentiate between them by referencing their ID to the cloud.

In any of these solutions, a certain level of trust is required from the client to the sensor manufacturer. The manufacturer will see this original authentication phase unfold and must be trusted to let operations take place. Also, the manufacturer has access to the master key during manufacturing and are there during the original symmetric key pair generation, as they must communicate the public key to the client. This gives them access to future communications even if they claim not to store those keys and goes against the original need for user secrecy.

5.3 Encryption using the pre-shared secret

We will now look at how this master key can be used for further encryption. We will **incrementally** discuss improvements of these solutions in terms of security, feasibility and user friendliness in the following sections. At each step, we will show why each of the earlier propositions are not feasible or insecure and what steps we have taken to improve on them.

The following propositions follow the assumption that the pre-shared key or keys have been successfully delivered between the sensor and the storage, using the implementation of the master key and authentication through its derived public key, as discussed in Section 5.2. **Authenticity between the two entities is already validated.**

5.3.1 Symmetric key encryption

The first idea that comes to mind is the simplest; to **use the pre-shared symmetric key for encryption**. Today, a key of at least 128-bits is required to circumvent key retrieval attacks for symmetric cyphers as recommended by the NIST [BR19]. But the lightweight capabilities of the sensors may be more prone to other cryptographic protocols (see Chapter 6). These steps can be seen in Figure 5.3.

Following Section 5.2, the symmetric key used in this case would be the master key and would be common to both the sensor and the storage. In the diagram, K is the master key. At manufacturing, the key would be safely delivered to the client and be used to encrypt all communications.

Impact:

- Security
    - The symmetric key protocol needs to be properly secure, with a key of appropriate length.
    - The constraint of user privacy from the sensor manufacturer is not respected in this case, as they have access to the original symmetric key, the master key in this case.
    - Authenticity is respected here, but more because the two end-points are the only ones able to decipher the data. Adversaries cannot decipher their communications if sent in a broadcast manner but the entities have no way of properly identifying each other.

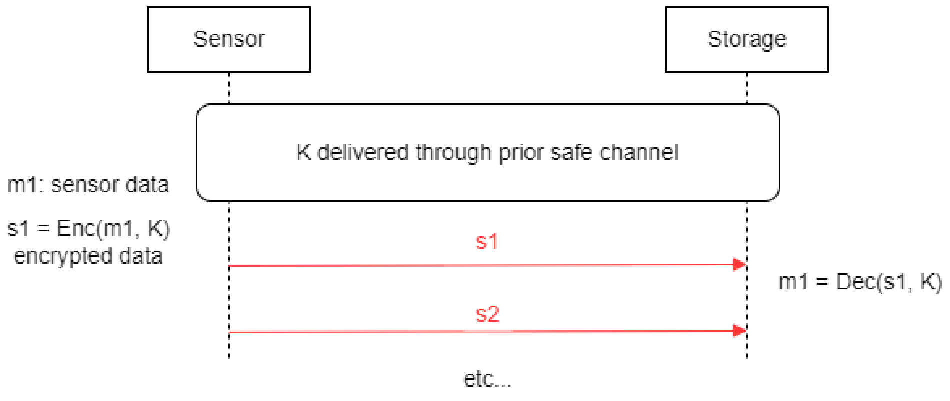

Figure 5.3: Symmetric Key Communication

- Feasibility: This solution is rather easy to implement.
- User friendliness: Encrypted communications are ready to send immediately after connection.

5.3.2 Asymmetric key encryption

Another simple design would be **using asymmetric encryption directly** from the key pairs used for authentication. The NIST suggests using a key of at least 256 bits when implementing asymmetric key encryption [BR19]. After the authentication phase using the public keys of each end-point, communications would be encrypted as described in Section 2.2.1. These steps can be seen in Figure 5.4.

Impact:

- Security:
  - It is about the same level of security as the symmetric key encryption, depending on the cypher chosen. See different options for these cyphers in Chapter 6.
  - The constraint of user privacy from the sensor manufacturer is still not respected in this case, as they have access to the master key.
  - Authenticity is now respected, as we are following the implementation of Section 5.2.
- Feasibility: Asymmetric key encryption may be quite heavy in terms of resources on the sensor's side.

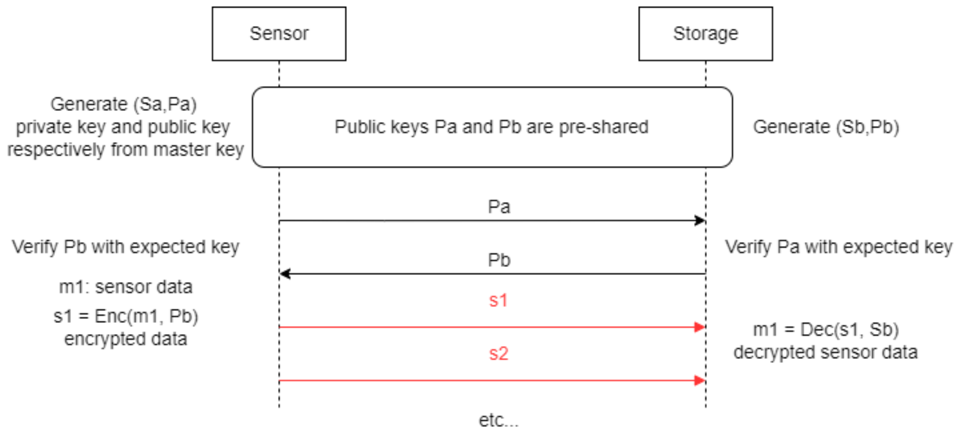
Sensor
Storage
Generate (Sa,Pa) private key and public key respectively from master key
Public keys Pa and Pb are pre-shared
Generate (Sb,Pb)
Pa
Verify Pb with expected key
Pb
Verify Pa with expected key
m1: sensor data
s1
s1 = Enc(m1, Pb) encrypted data
m1 = Dec(s1, Sb) decrypted sensor data
s2
etc...

Figure 5.4: Asymmetric Key Communication

– User friendliness: Encrypted communications are ready to send immediately after connection.

5.4 Key exchange & encryption using the pre-shared secret

Many different options for key exchange exist, using symmetric or asymmetric keys (see section 2.2 for more detail). The solutions presented in this section are an improvement over Section 5.3, which immediately used the master key or pairs of keys derived from the master key for encryption. We continue our incremental approach here with an added key exchange step, where we now use the master key to derive new keys for encryption.

5.4.1 Asymmetric handshake & symmetric key encryption

This implementation is a hybrid between asymmetric key cryptography and symmetric key cryptography. After authentication, the private and public key pairs from both entities are used to establish connection using a **DH handshake or ECDH key exchange protocol** (see Section 2.2.4). After this handshake, an agreed upon secret is calculated and then used for further communications, making computations less complex for the sensor.

We assume that a KDF is used here to generate the pair of symmetric keys used during the handshake, as specified in Section 5.2 to avoid key recovery. These steps can be seen in Figure 5.5.

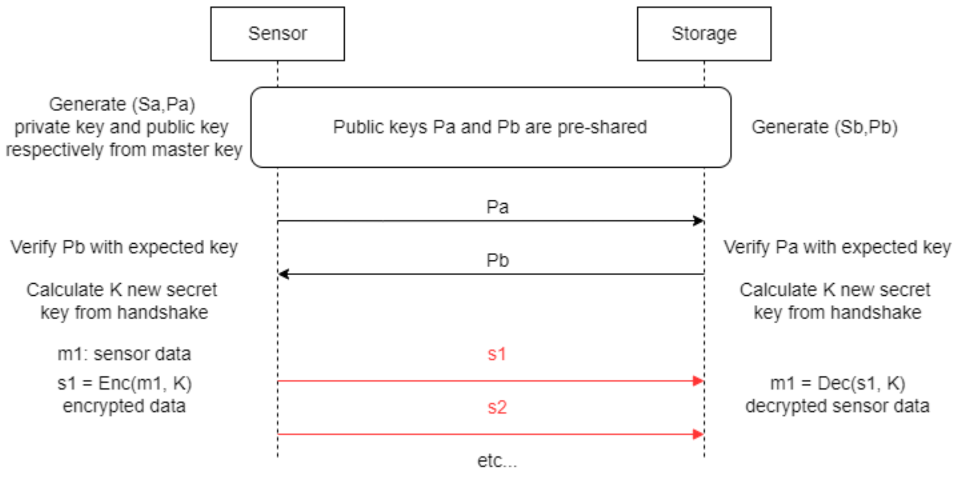
Sensor
Storage
Generate (Sa,Pa) private key and public key respectively from master key
Public keys Pa and Pb are pre-shared
Generate (Sb,Pb)
Pa
Pb
Verify Pb with expected key
Verify Pa with expected key
Calculate K new secret key from handshake
Calculate K new secret key from handshake
m1: sensor data
s1 = Enc(m1, K) encrypted data
s1
s2
m1 = Dec(s1, K) decrypted sensor data
etc...

Figure 5.5: Asymmetric Key Handshake + Symmetric Key Encryption

Impact:

- Security:
  - The constraint of user privacy from the sensor manufacturer is still not respected in this case, as they have access to the master key.
- Feasibility: An improvement is made on the asymmetric key encryption implementation as the use of a symmetric key for encryption is less resource intensive.
- User friendliness: This option requires the storage owner to implement the protocols required or that the sensor manufacturer deploy it on the storage.

5.4.2 Asymmetric handshake renewal

The solution of asymmetric key exchange followed by symmetric key encryption answers the problem of authentication but the master key was provided by the sensor manufacturer. Theoretically, if they wanted to decrypt sensor-to-storage communications they could, as they embedded the master key onto the sensor. This is a cynical view of the trust given to the sensor manufacturer, but if a client wants their data to be seen only by them, an extra step must be taken.

To keep communications private from the sensor manufacturer, a possible solution would be to **update public keys after the authentication phase**, using the original key generation protocol with the KDF. This time, key generation would not be done under the eyes of the manufacturer and the new key pair would be completely secret. The rest of the protocol would go on as planned. These steps can be seen in Figure 5.6. This would require a little more computing power from the sensor but is feasible.

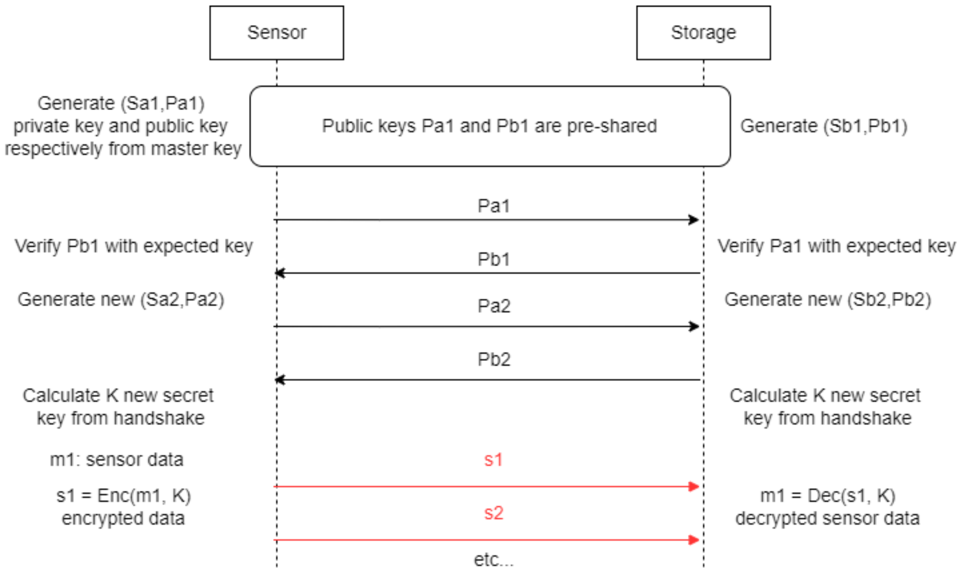
Sensor
Storage
Generate (Sa1,Pa1) private key and public key respectively from master key
Public keys Pa1 and Pb1 are pre-shared
Generate (Sb1,Pb1)
Pa1
Verify Pb1 with expected key
Verify Pa1 with expected key
Pb1
Generate new (Sa2,Pa2)
Generate new (Sb2,Pb2)
Pa2
Pb2
Calculate K new secret key from handshake
Calculate K new secret key from handshake
m1: sensor data
s1
s1 = Enc(m1, K) encrypted data
m1 = Dec(s1, K) decrypted sensor data
s2
etc...

Figure 5.6: Asymmetric Key Handshake with renewal + Symmetric Key Encryption

Impact:

– Security: User privacy is now achieved.

– Feasibility: This solution demands another calculation for key generation to be done by the sensor. Finding the most lightweight protocol for this in Chapter 6 is important.

– User friendliness: This succession of events is quite long to implement for the user. It should be automated or at least properly documented.

5.4.3 Session key & key renewal

This solution is a slight improvement following the key renewal solution. It answers the possibility of a connection drop during communications, be it accidental through some network failure event or intentional, to renew connections.

The solution is implemented as such: every time reconnection is required, a new DH or ECDH exchange is done to **compute a fresh symmetric key** used to encrypt this new session's communications. These steps can be seen in Figure 5.7. Re-connection should also be forced periodically to avoid giving attackers too much time and information for their brute-force attacks. Another potential reason would be that a node in the network is compromised and has leaked keys to an attacker, rendering all future communications unencrypted for that attacker [MCLD14].

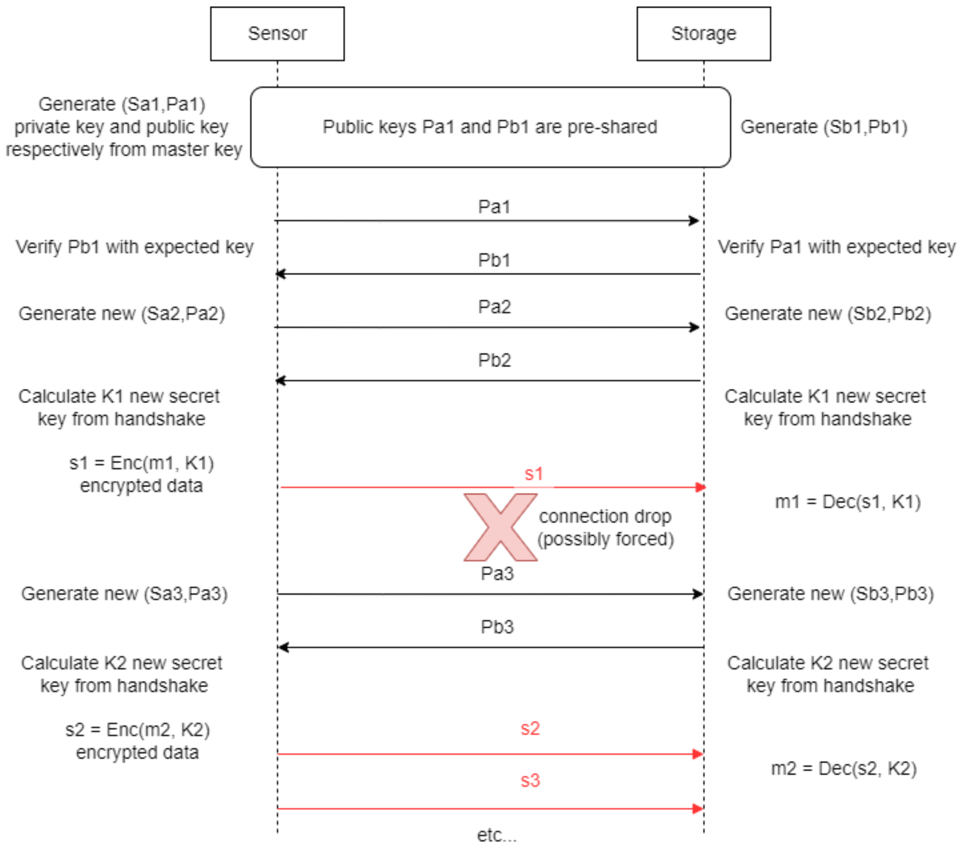
Sensor
Storage
Generate (Sa1,Pa1) private key and public key respectively from master key
Public keys Pa1 and Pb1 are pre-shared
Generate (Sb1,Pb1)
Pa1
Verify Pb1 with expected key
Pb1
Verify Pa1 with expected key
Generate new (Sa2,Pa2)
Pa2
Generate new (Sb2,Pb2)
Pb2
Calculate K1 new secret key from handshake
Calculate K1 new secret key from handshake
s1 = Enc(m1, K1) encrypted data
s1
m1 = Dec(s1, K1)
connection drop (possibly forced)
Pa3
Generate new (Sa3,Pa3)
Generate new (Sb3,Pb3)
Pb3
Calculate K2 new secret key from handshake
Calculate K2 new secret key from handshake
s2 = Enc(m2, K2) encrypted data
s2
m2 = Dec(s2, K2)
s3
etc...

Figure 5.7: Asymmetric Key Handshake with renewal + Symmetric Key Encryption with reconnection

The period at which this key renewal should be done is up to the user. This depends on the size of the key and the capabilities of the sensor. If the key is longer then renewal need not be done so often. Also, if the sensor is limited in terms of calculations, key renewal can not be done too often you have to trade off security for battery life. Another alternative would be to renew keys any time a sensor has downtime, for example if a sensor is only used during the day, set renewal every morning before it starts collecting data.

Depending on the cypher used, the possibility to re-use some parameters from the last session to avoid some setup costs could be done. For example, some cyphers use metadata to calculate new keys, counters and such, in order to make renewal costs less than initialization.

Impact:

- Security: This solution avoids threats related to key re-use attacks.
- Feasibility: It remains the same.
- User friendliness: The need for a ”home-brewed” protocol or an API developed by the manufacturer for the client to deploy becomes quite clear as this is a lot of steps to follow.

5.5 Separating private and public data

In the Section Section 5.3 and 5.4 we focused on sensor-to-storage communications. The data going to the cloud is metadata about sensor operations and its ID. This data should not allow the sensor manufacturer to decrypt communications between the sensor and the storage. By this we mean the already established cloud in our use case (see the orange arrow in Figure 5.1). In this section, we will discuss the intricacies of separating data for storage and data to the cloud.

Differentiating data types must be done sensor-side. All communications go through the gateway but they are already encrypted at that step. The destination for the data is referenced in the network packet sent by the sensor and the gateway redirects the packet to its proper destination. The gateway does not look at the data to tell where it must be forwarded to.

Any time the sensor takes a reading, it will encrypt it, save the encrypted data in its RAM and encapsulate it in a packet with the storage as destination. If it needs to send analytic data to the cloud, the cloud’s address will be referenced in the packet.

5.6 Key storage

Storing keys is a very important part of key management. Even with an unbreakable encryption cypher, if an attacker can retrieve keys on a device, then they can decrypt all communications.

Bad practices for key storage are implementations like hard-coding keys into source code or storing them in environment variables and leaving them in configuration files that do not require permissions. Keys should also be stored separately from the data they encrypt and decrypt, avoiding attackers getting lucky by gaining access to one and having access to both.

When available, the best key storage options for a system are:

- **A Hardware Security Module (HSM)**. These devices are dedicated cryptographic processors used for secure key storage, encryption/decryption, digital signatures and authentication purposes. They can be attached onto a device or accessible through a local network.
- **Key Vaults**, such as those used in Amazon Web Services' Key Management Service and Microsoft Azure's Azure Key Vault. These are key storage options proposed by public cloud providers.
- **Secure Storage APIs**, for example the *ProtectedData* class [Mic] offered by Microsoft's .NET framework.

These options provide safe management and storage of keys, they are tamper-proof and can offer the possibility of key generation as well. If these options are not available, **keys should at least be stored in an encrypted fashion**, using a KEK. That KEK can also be derived from the master key and should be stored apart from the other keys, preferably on a secure other device. If that is not an option, another less safe option would be to use **obfuscating code** to hide the private or secret key. The idea is to use very confusing code used to decipher the stored encrypted keys, making attackers have to go through a tedious reverse-engineering phase when trying to obtain these keys.

In our use case, the devices affected by key storage are the end devices for communications; sensors, storage and cloud. Each of these has different key storage capabilities. For example, the sensor does not have access to key vaults and neither does the on-premises storage if it is not a private cloud utilizing the aforementioned services.

The best option for key storage sensor-side would be to use a HSM. If this is not possible, the microcontrollers that the sensors run on should store keys in Static Random Access Memory (SRAM) when using them because it allows small access time, although it is not a very secure solution, as this would require the device to always be turned on. The key should be loaded into SRAM from flash or other non-volatile memory for instance. Some newer microcontrollers are also designed with key storage in mind. In the case of the MAX32558 [Int], this is done using 256-bit AES key storage.

5.7 Discussion

In this Chapter we discussed how to achieve authenticity and confidentiality of sensor data in order to combat the threats exposed in Chapter 4.

To achieve confidentiality, we designed the use of a **master key** which can be used to derive all keys needed for further encryption in sensor-to-storage communications. After a step-by-step look at proper encryption methods, we arrived at Section 5.4.3, which shows the most complete communications protocol we have designed using the master key.

To achieve authenticity, there was a need to have a pre-shared secret between the sensor and the storage. This pre-shared secret was either the entities' respective public keys in the case of asymmetric cryptography or the master key itself, being a secret key in the case of symmetric cryptography. To share this common secret, two options were outlined; **sharing the secret at the manufacturing phase**, as discussed in Section 5.2.3 and **sharing the secret through the pre-existing connection to the cloud**, as discussed in Section 5.2.4. These two options are compared in terms of security, feasibility and user-friendliness Table 5.1.

Comparison term	**Sharing the secret at manufacturing**	**Sharing the secret through the cloud**
Security	It must be done in a secure room and it relies on an untrusted method of delivering the secret to the storage on the client's side.	It relies on the secure pre-existing mutual authentication protocol with the cloud which is considered safe.
Feasibility	It requires more input from the client at manufacturing, either physically or through some secure sharing protocol.	It requires the storage to be able to have cloud access.
User-friendliness	In the case of asymmetric keys, it requires the client to use the same key pair as the one referenced at manufacturing.	It allows the client to switch back and forth with cloud storage. It is not possible if the client wants to deploy their sensors in a network having no internet access.

Table 5.1: Comparison of pre-shared secret options

The solution for mutual authentication of sensors and storage using the master key that we have proposed in Section 5.2 is valid but requires some coordination from the sensor manufacturer and the client when it comes to sharing the pre-shared secret during the manufacturing phase. This **requires a lot of trust** from both entities and could potentially slow down the manufacturing process drastically. Therefore, this solution is adapted to clients who order sensors by the hundreds if not thousands. For a client who only wants a few sensors to implement in their home and does not like having to deal with the manufacturer's cloud, this solution is not financially valuable for the manufacturer. We suggest the client either use the already existing cloud protocol or the authentication-through-cloud solution suggested in Section 5.2.4.

The solution we have proposed in Section 5.4.3 runs authentication to then use a symmetric key for less encryption and decryption calculations, which allows for lower latency. In the next chapter, we will be looking in detail at what lightweight cyphers can be implemented to run authentication and encryption for our purposes.

Chapter 6

Lightweight Cypher Study

As we have seen in Chapter 5, there is a need for secure, lightweight encryption cyphers, both symmetric and asymmetric, depending on the preferred option. An AEAD alternative would also be welcome as it adds more security. We have also talked about using a key exchange protocol, but there is no need for a lightweight alternative as ECDH key exchange is not too demanding for a sensor [RMF+15].

But how lightweight a protocol do we need? A typical microcontroller used in lightweight sensors today would have the specifications of a CC1350 MCU [Ins] or a MAX32558 [Int]. As stated by Trappe Howard and Moore [THM15], implementing even the RC5 algorithm [Riv95], one of the most simple block cyphers to exist, requires 32-bit rotations which already pushes the limits of these 32-bit microcontrollers.

Cryptographic algorithms also require RAM and lookup tables to store calculations or keys needed for implementation. These resources are also required by the sensor to operate core functionality which means devoting them to security functions for too much time results in a performance drop for the sensor. This is why finding a proper lightweight cryptographic protocol is crucial for encrypting IoT communications.

The NIST holds workshops and conferences to find the best cryptographic primitives. The primitives go through rigorous testing, attack-resistance trials, benchmarking and a full cryptanalysis of the primitives is done. At the end, the best candidates get standardised and become part of the NIST standards, which gives them a highly reputed seal of approval. The NIST holds a particularly interesting workshop focused entirely on lightweight cryptography [TMC+21]. The last workshop was held on May 9-11 2022 and defined a list of finalists which will be further evaluated and possibly standardized in 2023. We will study some of these finalists and compare them in terms of performance and security.

We have also found articles referencing lightweight cyphers; a study by Danda et al. [DSJ20] and another by Hammad et al. [BZ17]. These studies differentiate cyphers using different metrics, notably classifying them by GE (Gate Equivalence) requirements. Although this unit of measurement is dependent on CMOS technology implementation of the cypher, it gives a generally good idea of resource requirements.

We will look at different promising cyphers from these sources and see how they are implemented, as well as what makes them lightweight.

6.1 Keccak

Keccak [Tea] is a hash function based on the sponge construction principle. The sponge construction, as seen in Figure 6.1 takes a permutation f that transforms a fixed-length input into an output of the same length. It adapts this function into a variable-length input and output construction by breaking a variable-length input into blocks which it will XOR with the previous block that was run through the function *f*. This is the absorbing phase. The squeezing phase consists of running the same iterations but this time taking out the blocks one by one, leaving us with an output of desired length.

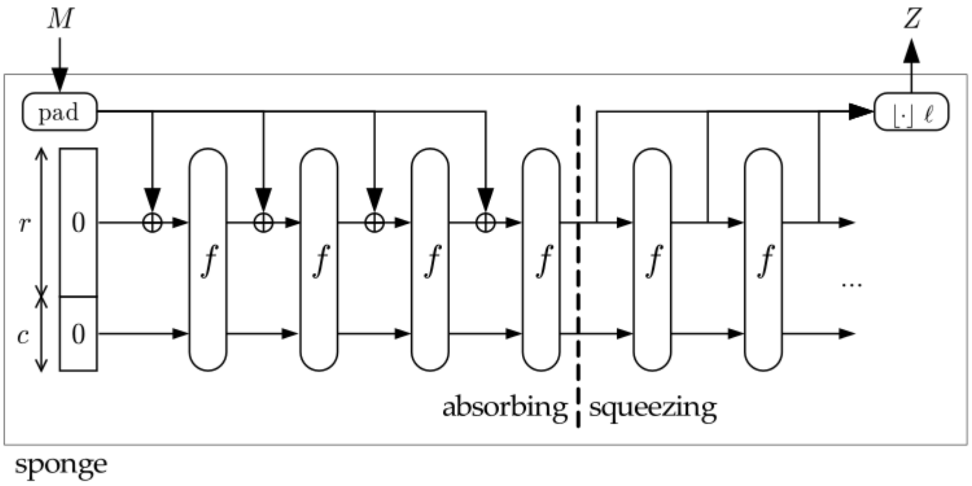

Figure 6.1: Sponge construction diagram

The originality in Keccak is the permutations f that are fed into the sponge construction. Keccak works by using a certain permutation chosen from a set of seven Keccak-f[b] permutations, with b being the width of the permutation. Although the *Keccak-f[1600]* permutation is more commonly used, the *Keccak-f[200]* and *Keccak-f[400]* permutations seem to give the best results in terms of required computing power, particularly when using a serialized architecture hybrid of Keccak [KY10].

For example, by serializing the common *Keccak-f[200]* permutation, an area of 2.52 kGE is achieved, with throughput of 8 Kbps, 900 cycles per block and a power consumption of 5.6 μW/MHz.

Keccak can also be used for authentication and authenticated encryption by using the duplex method [BDPV12] and thus could suit our needs. Other encryption cyphers presented in this chapter make use of lightweight hash functions to operate, and Keccak could also be a good candidate for them. It is the winner of the SHA-3 competition and has been standardized in the NIST standard FIPS 202. We think this is a viable candidate for lightweight encryption of sensor data.

6.2 A5/1

A5/1 is a very widely used protocol to this day, particularly used in Global System for Mobile communications (GSM), a standard for mobile communications and the predecessor to 3G. It is very lightweight due to the key size being only 64 bits, which makes it suffer on a security standpoint. A5/1 is a shift cypher, meaning it is a symmetric key cypher. It can also be used for key generation.

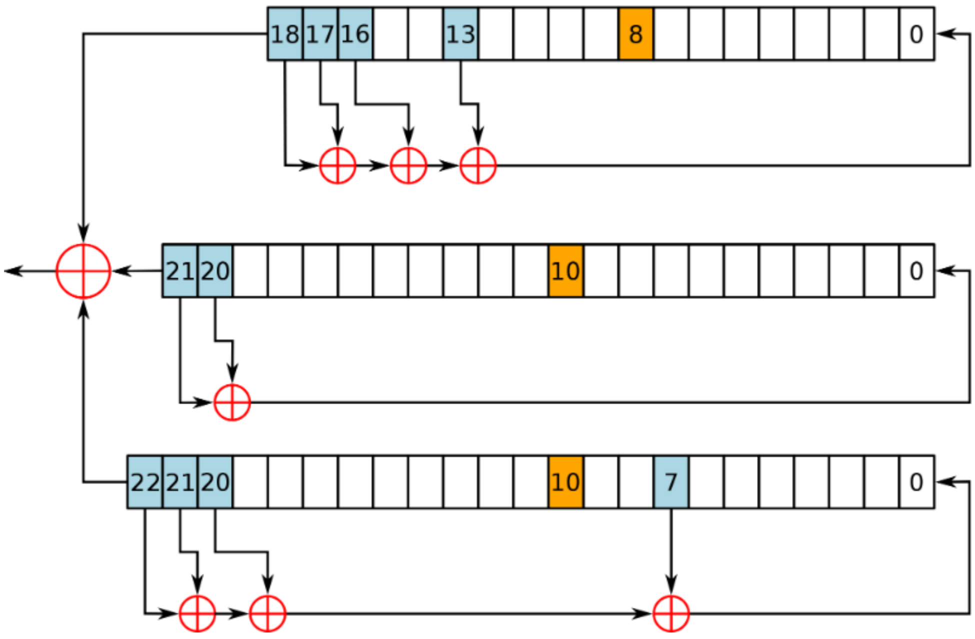

Figure 6.2: A5/1 LFSR keystream generation

A5/1 functions as such; It encrypts 114-bit chunks of data by XOR'ing them with a generated keystream. The keystream is generated from 3 Linear-Feedback Shift Registers (LFSRs), as seen in Figure 6.2. They are initialised with all bits at 0. Each of these LFSRs has a clocking bit, which are examined at each cycle to determine the majority bit. If this majority bit is the same as a register's clocking bit, that register is clocked. A register being clocked means that the i^{th} bit of the secret key is added to the least significant bit of that register. This is done for 64 cycles, one cycle for each bit of the secret key. This is then repeated for the 22 bits of the frame number. This use of the majority bit for clocking ensures pseudo-randomness by having all 3 registers shift asynchronously.

Today, a 64-bit key can be cracked in many different ways and in a few minutes using a powerful enough computer [BSW00]. A5/1 is easy to implement but does not seem robust enough when compared to other alternatives, especially given the key-size used. We chose to mention it to show that one of the more lightweight alternatives we found (according to Dhanda et al. [DSJ20]) achieves this because of its drastic trade-off in security.

6.3 APE

Authenticated Permutation-based Encryption (APE) is a block cypher that relies on the use of permutations from lightweight hash functions such as Quark [AHMN10], Spongent [BKL+11] or PHOTON [GPP11]. It has an adaptable key-size and promising throughput and area usage results [Ele13].

Figure 6.3 shows APE's mode of encryption. p is the chosen hash function in the construction with A the associated data (extra security measures like a MAC or a Nonce) and M the message (sensor data in our case). The associated data is XOR'ed block-by-block with part of the previous hash then combined with the key and run through the hash p once again. This is then repeated with the message.

APE was designed to withstand Nonce-misuse attacks. These attacks consist of exploiting a cryptographic system where the same Nonce is being re-used for different messages using the same encryption key, going against our recommendations in Section 2.2.4. APE does this by encrypting the Nonce along with all other data as can be seen in Figure 6.3. APE has achieves an average area of 2000 GE, a throughput of 2 Kbps and a latency of 2000 cycles.

6.4 PHOTON-Beetle

PHOTON-Beetle is one of the NIST lightweight candidates at their most recent workshop [TMC+21]. It relies on the PHOTON [GPP11] hash, particularly the P_{256} permutation, meaning the PHOTON permutation with a hash output of 256 bits.

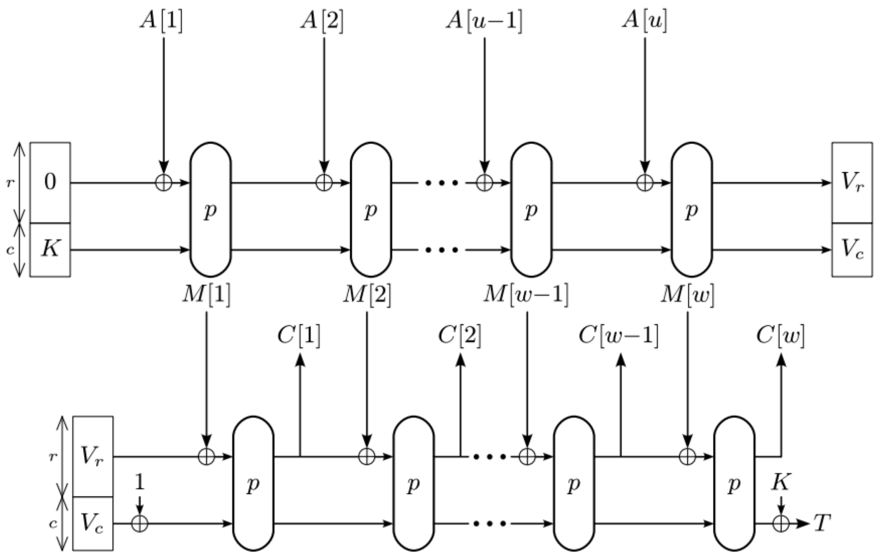

Figure 6.3: APE Encryption

This permutation is a sponge function AES-like matrix permutation. PHOTON-Beetle can be used both for authenticated encryption, using the PHOTON-Beetle-AEAD family of functions and for hashing, with the PHOTON-Beetle-Hash family. PHOTON-Beetle achieves an estimated are of 1736 GE and a latency of 1716 clock cycles.

As seen in Figure 6.4, for PHOTON-Beetle-AEAD encryption, the permutation f is the P_{256} permutation. It works similarly to APE, in that the nonce, key and associated data are processed much in the same way, by calculating the first r bits and so-on. The originality of PHOTON-Beetle is its use of P_{256} permutation and the fact that the cyphertext is not simply output after the squeezing phase, but is run through another function p. p is a linear function which takes the combined feedback of the permutation output (called the state) and the cyphertext block to generate the next permutation input [BCD+19]. This use of the permutation output stops the attacker from being able to control the next output if they have access to the input, which adds another level of security to existing sponge cyphers.

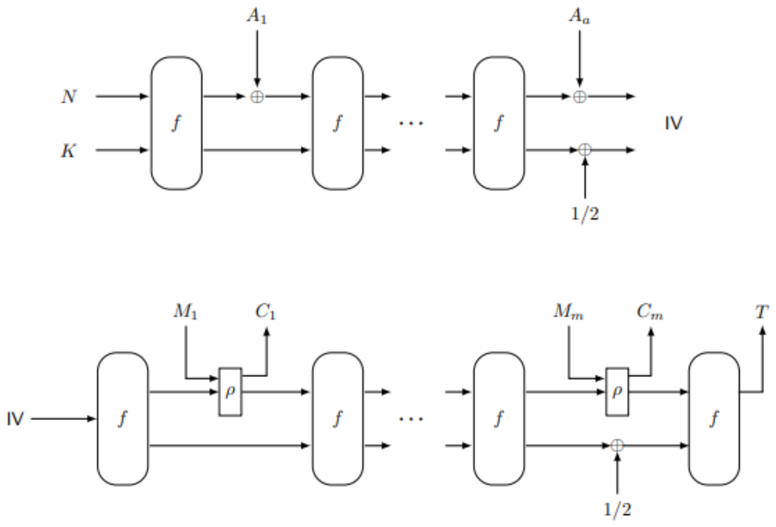

Figure 6.4: PHOTON-Beetle-AEAD encryption [BCD+19]

6.5 Discussion

In this chapter, we attempted to make a literature study and comparison of existing lightweight cyphers, to complement our implementation suggestion of Chapter 5. A lack of time limited our research, and only a few cyphers could be included. Other interesting cyphers we intended to discuss included SFN [LLZZ18], HashOne [MMSS16], ASCON [DEM+], Elephant [Via22] and Xoodyak [DW].

From what can be seen of the candidates of the recent NIST lightweight cryptography workshop [TMC+21], there is a real need for such encryption methods, and the preferred encryption type seems to be through the use of AEAD. Newer cyphers seem to be built with one specific attack in mind and circumventing that attack, all while keeping lightweight requirements. The aim for gate equivalence is around 1000-1500 GE.

PHOTON-Beetle and APE seem to be very worthy cyphers, both in terms of security and resource requirements. We cannot however make a full comparison as we do not have access to all resource metrics, and comparing only two cyphers seems a little futile.

Chapter 7
Conclusion

The use of small sensors to monitor simple activities is growing extremely fast. Companies developing such sensors need to be at the cutting-edge of technology and must take into account manufacturing costs, sensor capabilities as well as the clients' needs when developing sensors. With clients relying on these sensors more and more, and with data privacy being at the forefront of information security discussions today, more and more clients want to rely on their own on-premises infrastructure to communicate with sensors and store their data.

7.1 Results

We have discussed the possible on-premises storage options for user-friendly sensor data storage in Chapter 3. We also evaluated the security requirements for an on-premises alternative to cloud data storage, through the use of a STRIDE threat model in Chapter 4. These threats relied on unauthorised data access and unauthorised connection to devices of our IoT network. To combat these threats, we proposed two methods of master key delivery in Chapter 5, these being the authentication-through-cloud approach and the sharing-at-manufacturing approach. We also discussed the most effective key exchange solution as an alternative to the existing sensor-to-cloud protocol, which brought up our literature study of different user-friendly low-level encryption protocols in Chapter 6.

7.2 Discussion

The storage options discussed in Chapter 3 are **all viable** in our case, although each has its trade-offs between security and aese of use. The real difference in implementation comes at Chapter 5 where we discuss key management. The main problem in this thesis was achieving authentication without using the cloud's automatic sync to the sensor. Our propositions were those of sharing the secret at manufacturing and sharing the secret through the cloud.

As we have said in Section 5.7, **sharing the secret at manufacturing** requires a lot of coordination between the manufacturer and client, and raises other threats. This is also more costly to the manufacturer and not a viable solution for clients only ordering a few sensors. On the other hand, **using the pre-existing cloud protocol** relies on the client's network being connected to the internet, which may not be the case as it is a motivation for clients to have on-premises storage in the first place.

The encryption and key exchange steps of Section 5.3 and Section 5.4 are quite straight-forward when it comes to designing a cryptographic system. When it comes to key storage in Section 5.6, **using a HSM is our top recommendation**. If a HSM is not an option, the next best thing is using a KEK to **encrypt all keys** used for encryption and authentication. Keys should never be stored in the clear and should be kept well apart from data. The overview of lightweight cyphers in Chapter 6 does not go into enough depth into the comparison between their different resource consumption, meaning we cannot make trustworthy suggestions.

7.3 Future work

The logical next step for this thesis is **implementation**. A proof-of-concept implementation of our proposed cryptographic scheme on a microcontroller communicating to a device imitating a server would be our test bench. Testing different promising cyphers that were mentioned in Chapter 6 to see resource consumption results and comparing them would give us a clearer idea of what implementation is best. The next step could be to **run known attacks** against cryptographic systems to see if we could recover any information.

The heart of this thesis has also been the **relationship between manufacturer and client**, and exploring the level of trust clients are willing to give to manufacturers would be another interesting option, by interviewing clients and their IT departments, to see where better communications can be made between companies to ensure proper data security and clarity between those parties.

References

[AGGL22] I. Aldasoro, L. Gambacorta, *et al.*, «The drivers of cyber risk», *Journal of Financial Stability*, vol. 60, p. 100 989, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1572308922000171?casa_token=b3Sd6g3iidMAAAAA:eK0T4rHJ3MGxK5t0Gru0MXs2qM511hEP4BH9P8jqRnv2fvw2UvpGW k5jsVoRg.

[AHMN10] J.-P. Aumasson, L. Henzen, *et al.*, «Quark: A lightweight hash», vol. 26, Aug. 2010, pp. 1–15. [Online]. Available: https://www.iacr.org/archive/ches2010/62250001/62250001.pdf.

[BCD+19] Z. Bao, A. Chakraborti, *et al.*, «Photon-beetle authenticated encryption and hash family», *NIST Lightweight Compet. Round*, vol. 1, p. 115, 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/PHOTON-Beetle-spec.pdf.

[BDPV12] G. Bertoni, J. Daemen, *et al.*, «Duplexing the sponge: Single-pass authenticated encryption and other applications», in *Selected Areas in Cryptography*, A. Miri and S. Vaudenay, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 320–337. [Online]. Available: https://keccak.team/files/SpongeDuplex.pdf.

[BKL+11] A. Bogdanov, M. Knežević, *et al.*, «Spongent: A lightweight hash function», Sep. 2011, pp. 312–325. [Online]. Available: https://www.iacr.org/archive/ches2011/69170311/69170311.pdf.

[BR19] E. Barker and A. Roginsky. «Transitioning the use of cryptographic algorithms and key lengths». en. (Mar. 2019), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf.

[BSW00] A. Biryukov, A. Shamir, and D. A. Wagner, «Real time cryptanalysis of A5/1 on a PC», in *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, B. Schneier, Ed., ser. Lecture Notes in Computer Science, vol. 1978, Springer, 2000, pp. 1–18. [Online]. Available: http://jf.morreeuw.free.fr/security/url21.pdf.

[BZ17] M. E. R. Baraa Tareq Hammad Norziana Jamil and M. R. Z'aba, «A survey of Lightweight Cryptographic Hash Functions», *International Journal of Scientific & Engineering Research*, vol. 8, no. 7, pp. 806–814, Jul. 2017. [Online]. Available: https://www.ijser.org/researchpaper/A-survey-of-Lightweight-Cryptographic-Hash-Function.pdf.

[CVE] CVE Program, *CVE*. [Online]. Available: https://cve.org.

[DEM+] C. Dobraunig, M. Eichlseder, *et al.*, «New ascon implementations», [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf.

[Dis] Disruptive Technologies, *Disruptive Technologies - Tiny Wireless Sensors & IoT Infrastructure*. [Online]. Available: https://www.disruptive-technologies.com.

[DRA16] P. Dey, R. S. Rohit, and A. Adhikari, «Full key recovery of acorn with a single fault», *J. Inf. Secur. Appl.*, vol. 29, no. C, pp. 57–64, Aug. 2016. [Online]. Available: https://doi.org/10.1016/j.jisa.2016.03.003.

[DSJ20] S. S. Dhanda, B. Singh, and P. Jindal, «Lightweight Cryptography: A Solution to Secure IoT», *Wirel. Pers. Commun.*, vol. 112, no. 3, pp. 1947–1980, 2020. [Online]. Available: https://doi.org/10.1007/s11277-020-07134-3.

[DW] O. Dunkelman and A. Weizman, «Differential-linear cryptanalysis on xoodyak», [Online]. Available: https://csrc.nist.gov/csrc/media/Events/2022/lightweight-cryptography-workshop-2022/documents/papers/differential-linear-cryptanalysis-on-xoodyak.pdf.

[Ele13] Elena Andreeva and Begül Bilgin and Andrey Bogdanov and Atul Luykx and Bart Mennink and Nicky Mouha and Kan Yasuda, «APE: authenticated permutation-based encryption for lightweight cryptography», *IACR Cryptol. ePrint Arch.*, p. 791, 2013. [Online]. Available: http://eprint.iacr.org/2013/791.

[GPP11] J. Guo, T. Peyrin, and A. Poschmann, «The photon family of lightweight hash functions», in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 222–239. [Online]. Available: https://eprint.iacr.org/2011/609.pdf.

[HMLY05] R. Hasan, S. Myagmar, *et al.*, «Toward a threat model for storage systems», Nov. 2005, pp. 94–102. [Online]. Available: https://www.researchgate.net/publication/221103837_Toward_a_threat_model_for_storage_systems.

[Ins] T. Instruments, *Cc1350 - simplelink™ 32-bit arm cortex-m3 multiprotocol sub-1 ghz & 2.4 ghz wireless mcu with 128kb flash*. [Online]. Available: https://www.ti.com/product/CC1350.

[Int] M. Integrated, *Max32558 - deepcover secure arm cortex-m3 flash microcontroller*. [Online]. Available: https://www.maximintegrated.com/en/products/microcontrollers/MAX32558.html.

[Kis13] R. Kissel, *Glossary of key information security terms*, en, May 2013. [Online]. Available: https://www.nist.gov/publications/glossary-key-information-security-terms-1.

[KL14] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, 2nd. Chapman & Hall/CRC, 2014.

[Kre] T. Krenn. «Storage basics: Das, san and nas at a glance». (), [Online]. Available: https://www.thomas-krenn.com/de/tkmag/expertentipps/storage-grundlagen-das-san-und-nas-im-ueberblick/.

[KY10] E. B. Kavun and T. Yalçin, «A lightweight implementation of keccak hash function for radio-frequency identification applications», in *Radio Frequency Identification: Security and Privacy Issues - 6th International Workshop, RFIDSec 2010, Istanbul, Turkey, June 8-9, 2010, Revised Selected Papers*, S. B. Ö. Yalçin, Ed., ser. Lecture Notes in Computer Science, vol. 6370, Springer, 2010, pp. 258–269. [Online]. Available: https://www.researchgate.net/publication/221622639_A_Lightweight_Implementation_of_Keccak_Hash_Function_for_Radio-Frequency_Identification_Applications.

[LLZZ18] L. Li, B. Liu, *et al.*, «SFN: A new lightweight block cipher», *Microprocess. Microsystems*, vol. 60, pp. 138–150, 2018. [Online]. Available: https://doi.org/10.1016/j.micpro.2018.04.009.

[MCLD14] I. Mansour, G. Chalhoub, *et al.*, «Secure key renewal and revocation for wireless sensor networks», Sep. 2014, pp. 382–385. [Online]. Available: https://www.researchgate.net/publication/283702981_Secure_key_renewal_and_revocation_for_Wireless_Sensor_Networks.

[Mic] Microsoft, *Protecteddata class.* [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.protecteddata?redirectedfrom=MSDN&view=netframework-4.8.

[MMSS16] P. M. Mukundan, S. Manayankath, *et al.*, «Hash-one: A lightweight cryptographic hash function», *IET Inf. Secur.*, vol. 10, no. 5, pp. 225–231, 2016. [Online]. Available: https://doi.org/10.1049/iet-ifs.2015.0385.

[NIS] NIST, *Compliance faqs: Federal information processing standards (fips).* [Online]. Available: https://www.nist.gov/standardsgov/compliance-faqs-federal-information-processing-standards-fips#:~:text=are%5C%20FIPS%5C%20developed%5C%3F-,What%5C%20are%5C%20Federal%5C%20Information%5C%20Processing%5C%20Standards%5C%20(FIPS)%5C%3F,by%5C%20the%5C%20Secretary%5C%20of%5C%20Commerce..

[PB19] C. Paulsen and R. Byers, *Glossary of key information security terms*, en, Jul. 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.7298r3.pdf.

[Riv95] R. L. Rivest, «The rc5 encryption algorithm», in *Fast Software Encryption*, B. Preneel, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 86–96. [Online]. Available: https://people.csail.mit.edu/rivest/Rivest-rc5rev.pdf.

[RM77] Z. Ruthberg and R. McKenzie, «Audit and evaluation of computer security», en, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, Oct. 1977. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nbsspecialpublication500-19.pdf.

[RMF+15] O. Raso, P. Mlynek, *et al.*, «Implementation of elliptic curve diffie hellman in ultra-low power microcontroller», in *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, 2015, pp. 662–666. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7296346?fbclid=IwAR0ybQX5hYUf3C_55kr2Sv5V2c_m9alALzPeUjUH4BrTMzR-jLgk11-YveM.

[Ros] M. Rosulek. «The joy of cryptography». (), [Online]. Available: https://joyofcryptography.com.

[Sho14] A. Shostack, *Threat Modeling: Designing for Security*, 1st. Wiley Publishing, 2014.

[Sin21] S. Sinha. «State of IoT 2021: Number of connected IoT devices growing 9% to 12.3 billion globally, cellular IoT now surpassing 2 billion», IoT Analytics. (Sep. 22, 2021), [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/ (last visited: Nov. 7, 2021).

[SPA+21] A. K. Sikder, G. Petracca, *et al.*, «A survey on sensor-based threats and attacks to smart devices and applications», *IEEE Communications Surveys & Tutorials*, vol. PP, pp. 1–1, Mar. 2021. [Online]. Available: https://www.researchgate.net/publication/349901174_A_Survey_on_Sensor-Based_Threats_and_Attacks_to_Smart_Devices_and_Applications.

[Tea] K. Team, *Keccak*. [Online]. Available: https://keccak.team/keccak.html.

[TGP21] S. K. Tripathi, B. Gupta, and K. S. Pandian, «An alternative practical public-key cryptosystems based on the dependent rsa discrete logarithm problems», *Expert Systems with Applications*, vol. 164, p. 114 047, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420308149.

[THM15] W. Trappe, R. Howard, and R. S. Moore, «Low-energy security: Limits and opportunities in the internet of things», *IEEE Security Privacy*, vol. 13, no. 1, pp. 14–21, 2015. [Online]. Available: https://www.researchgate.net/publication/272385802_Low-Energy_Security_Limits_and_Opportunities_in_the_Internet_of_Things.

[Tho84] K. Thompson, «Reflections on Trusting Trust», *Commun. ACM*, vol. 27, no. 8, pp. 761–763, 1984. [Online]. Available: https://doi.org/10.1145/358198.358210.

[TMC+21] M. S. Turan, K. McKay, *et al.*, «Status report on the second round of the nist lightweight cryptography standardization process», en, NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, Jul. 2021. [Online]. Available: https://www.nist.gov/news-events/news/2021/07/status-report-second-round-nist-lightweight-cryptography-standardization.

[UVSL18] V. E. Urias, B. Van Leeuwen, *et al.*, «Applying a threat model to cloud computing», in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8585471.

[Via22] L. Vialar, «Fast side-channel key-recovery attack against elephant dumbo», *Cryptology ePrint Archive*, 2022. [Online]. Available: https://eprint.iacr.org/2022/446.pdf.