



University of
Zurich^{UZH}

Security Whitepaper for IoT-Enabled Hardware Temperature Logger

*Andreas Knecht
Zürich, Switzerland
Student ID: 11-916-152*

Supervisor: Thomas Bocek, Bruno Rodrigues
Date of Submission: May 15, 2018

Abstract

This work analyzes the strength of security measures in the cold-chain monitoring system developed by modum. Critical components are identified and analyzed against an extensive risk analysis. In addition, technical details are documented and discussed to enable an independent security audit.

Kurzfassung

Die Sicherheitsmerkmale der Kühlketten-Temperaturüberwachungslösung von modum werden analysiert. Kritische Komponenten des Systems werden identifiziert und mithilfe einer extensiven Risikoanalyse analysiert. Ausserdem werden technische Details dokumentiert und erörtert, um einen unabhängigen Audit der Sicherheitsmerkmale zu ermöglichen.

Acknowledgments

I want to thank Thomas Bocek, Malik El Bay, Stefan Weber and Mike Taylor for their input during turbulent times at modum.

Contents

Abstract	i
Kurzfassung	iii
Acknowledgments	v
1 Introduction	1
1.1 Description of Work	2
1.2 Thesis Outline	3
2 Background	5
2.1 Digital Signatures	5
2.2 Certificates	6
2.3 Chain of Trust	6
2.4 Certificate Revocation List	7
2.5 System Outline	7
3 Analysis	9
3.1 Logger Device	10
3.1.1 General Logger Interaction	10
3.1.2 Hardware Security Module	12
3.1.3 Risk Analysis	13
3.1.4 Mitigation Steps	21
3.1.5 Logger Production	24

3.2	Smart Contract	24
3.2.1	General Function of the Smart Contract	25
3.2.2	Risk Analysis	26
3.2.3	Mitigation Steps	29
3.2.4	Limitations	31
4	Technical Details	33
4.1	Merkle Tree for Signature Generation	33
4.2	Time Synchronization Protocol	34
4.3	Algorithms	34
4.3.1	Signature	35
4.3.2	Encryption	36
4.3.3	Hashing	36
4.3.4	Message Authentication Codes	36
4.3.5	Key Derivation	36
4.3.6	Random Number Generation	37
4.4	Key Management	37
4.4.1	Certificate Layout	37
4.4.2	Keys	38
5	Summary and Conclusions	45
	Bibliography	47
	Glossary	49
	List of Figures	50
	List of Tables	51
A	Large Graphics	55

Chapter 1

Introduction

Goods distribution practices (GDP) regulations in Switzerland and the European Union mandate the monitoring of temperature during all shipments of pharmaceutical goods [1, 2]. The modum system is a system under development to satisfy this requirement by the GDP [14]. The modum system uses Blockchain technology to improve this vital part of supply chain processes.

A logger device records temperatures during shipments of pharmaceutical goods. A client device reads out the logger devices when a shipment has arrived at the destination and uploads the data to the backend system. A Blockchain Smart Contract assesses the recorded temperatures, autonomously makes a decision about the shipment's compliance and stores this decision and the hash of the recorded temperatures immutably. A backend system is used to forward data between the client device and the smart contract, as well as to provide seamless integration for customers.

The Smart Contract acts as a single, autonomous authority deciding on the shipment's compliance status and storing the result immutably. Since the same smart contract code can be reused by many pharmaceutical companies and logistics providers, a single audit of the Smart Contract code can reduce the need for thorough audits at each logistics provider, reducing the burden on the logistics providers, pharmaceutical companies and the regulation agencies. It is also feasible in the future to, for example, automate release of funds based on a shipment's compliance status.

The modum system is currently under development and scheduled for release on the market in Summer 2018. The system aims to be very secure. This work supports this goal by providing a definition of security in the context of the modum system, analysing the system's compliance with that definition and providing a high-level and detailed description of security relevant processes and protocols in order to enable security audits by individual experts. At least one such audit is planned before release on the market.

1.1 Description of Work

The modum system is marketed to logistics and pharmaceutical companies to allow them to track that no harmful temperature deviations have occurred to medicinal products while they were being shipped. At the core of the system's use case stands patient safety. The system increases the safety of treatment with medications from a patient's perspective because medications that have been exposed to harmful environmental conditions can be discovered and disposed of before the patient consumes them. If a medication has been exposed to harmful environmental conditions, it may become ineffective or, worse, harmful to the patient. Thus, incorrect operation of the modum system can lead to a patient receiving ineffective or harmful treatment. Thus, the core risk of the system is any malfunction that results in a patient receiving ineffective or harmful treatment, endangering the patient. Any such malfunction must be mitigated. This can be summarized in two high-level goals:

1. No malfunction of the modum system must result in a patient consuming medications that have been exposed to harmful conditions.
2. No malicious tampering with the system must allow hiding of a recorded event where medications have been exposed to harmful conditions.

Figure 1.1: High Level Risks

Based on [6], we define safety of an IT system as the lack or mitigation of failure cases that would impose a big risk. We define security of an IT system as the lack of possible ways where malicious action can invoke a failure that would impose a big risk. Thus, when malicious actions are expected, it is critical to view security as an integral part of the system's safety, include maliciously invoked failures in the analysis of the system's safety and mitigate them accordingly.

According to above definitions, the first point in Figure 1.1 is a safety goal, while the second point is a security goal (and as such, also a safety goal).

Key concepts of IT security often include **availability**, **confidentiality**, **integrity** and **authenticity**.

This work doesn't cover availability because it is impossible to prohibit a malicious actor from withholding temperature measurements – e.g. by refusing to read out a temperature logger at the end of a shipment – on a technical level. This scenario could be covered on a contractual level when signing up customers. When no data can be retrieved for a shipment that was monitored by the modum system, the transported goods should be destroyed since compliance with the GDP cannot be proven without data.

Confidentiality is ensured in the client, backend and Smart Contract subsystems (see Section 2.5) through use of user- and role management, encryption and hashing of sensitive data. The confidentiality on the logger device is not prioritized in the first version of the modum system for reasons of compatibility and ease of use. Anyone without range of the

Bluetooth Low Energy (BLE) radio of the logger device can interact with it and read out temperature and shipment data.

Integrity and authenticity of recorded temperature data are core concerns and achieved by signing all temperature data with individual keys on each logger device. Thus, temperature data can be traced back to the individual logger device that recorded it and cannot be modified without knowledge of the secret key on a logger device.

1.2 Thesis Outline

Chapter 2 explains key concepts of cryptography that are used in the modum system and gives a system outline of the modum system. Chapter 3 gives an analysis of the security of the modum system. It starts by identifying key components that must be secure to mitigate the high level risks 1 and 2 in Figure 1.1. The analysis is structured in a risk analysis and a list of mitigation steps for each of the previously identified key components. Chapter 4 documents technical details, such as the exact binary format used to encode certificates that are handled on the logger devices. The goal is to enable security audits by independent experts.

Chapter 2

Background

This section explains key concepts used in this work, that are often misunderstood and that are used extensively in the modum system. The goal is to provide a common ground for understanding of Chapter 3.

2.1 Digital Signatures

A digital signature is a piece of data that is calculated from a message and a private key to ensure authenticity and integrity of the message after transmitting it through an insecure channel. The signature can be sent with the message or separately. It is important to note that the message is not changed by the signature algorithm, it is not encrypted, and can be read by anyone reading on the insecure channel. However, any change of a single bit in the message will result in the signature verification failing. Additionally, the signature provides a means to verify the authenticity of the signer of the message because successful verification of a signature with a given public key proves the sender was in possession of the corresponding private key. The signature verification algorithm has a binary output: whether the verification failed or not.

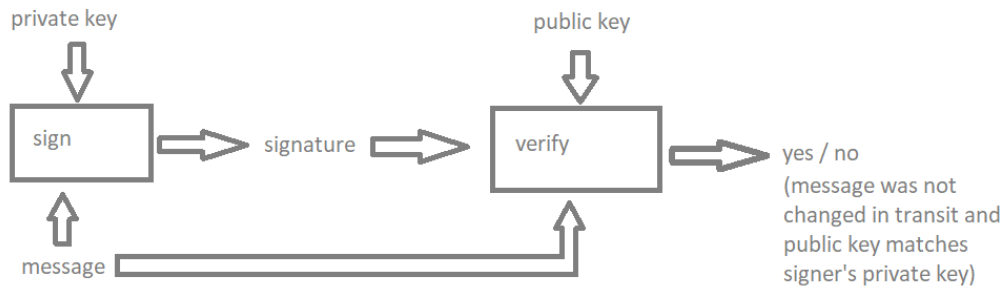


Figure 2.1: Digital Signatures

2.2 Certificates

To ensure the correct handling of digital signatures, certificates are used. Certificates contain the public key of a keypair, as well as additional information. Each certificate is signed by a trusted authority that vouches that the information contained in the certificate is correct and that they verified the authenticity of the owner the private key for a certificate. It is very important to store the intended use of a certificate in it, especially if the corresponding private key is easy to steal by an attacker.

In the modum system, for example, not all keys are equally secure from theft. Some keys are, for example, stored in a smartphone application and are considered relatively easy to steal and abuse. At the same time, private keys on the Logger Devices are used to sign temperature measurements. The certificates that are distributed to the Logger Devices contain the intended use of signing temperature measurements and are signed like this by a trusted authority. Thus, private keys that are used in different components of the modum system, such as the smartphone application, cannot be used to sign temperature measurements because the corresponding certificate does not state that as their intended purpose.

2.3 Chain of Trust

Because each certificate is signed by a trusted authority that may itself present a certificate signed by another trusted authority, a chain of signatures is formed that reaches up to the

so-called root certificate authority. This allows to extend trust to unknown intermediate authorities. If the root certificate authority proves trust in the intermediate authority by signing their certificate, in most cases certificates signed by the intermediate can be trusted because the trusted root certificate authority vouches for their trustworthiness.

In the modum system a chain of trust is used to mitigate the consequences of theft of private keys. For example, a Provisioning Key is used to sign the certificates of the Logger Devices recording temperature measurements. The modum Master Key is used to sign the Provisioning Key, but is otherwise kept very secure because theft of the Master Key would be catastrophic. The Provisioning Key will be used to sign all Logger Device certificates and is as a consequence required to be stored on a machine with a larger attack surface. However, theft of a Provisioning Key is not as detrimental as theft of the Master Key. Nonetheless, Logger Device certificates can be verified to be indirectly approved by the modum Master Key by checking that their certificate is signed with a Provisioning Key, whose certificate is signed with the Master Key.

2.4 Certificate Revocation List

When theft of a private key is noticed, the corresponding certificate is added to a certificate revocation list. This tells all users verifying the chain of trust to stop accepting a certain certificate and all the certificates signed by it, and certificates signed by those certificates and so on.

In the modum system the certificate revocation list is kept in a Smart Contract so that all modum Smart Contracts that perform signature verification can ensure the integrity of the certificates used.

2.5 System Outline

Figure 2.2 visualizes the general architecture of the modum system. The system consists of the modum temperature logger devices, a mobile client, a web client, a HTTP server with a PostgreSQL database attached and an Ethereum full node to interact with the Ethereum network.

The logger device records temperature measurements when it is placed together with a shipment of medicinal products. The recording is started and measurements are read out via the mobile client via Bluetooth Low Energy (BLE). Information about the started shipments and measurements are uploaded to the HTTP server via a RESTful API. The HTTP server stores a copy of the shipment information and measurements in the PostgreSQL database, as well as in an Ethereum Smart Contract via the full Ethereum node. The Smart Contract autonomously makes a decision about the shipment's legal compliance given the temperature measurements and stores this decision, as well as all other shipment information immutably. The shipment information can then be viewed and analyzed conveniently via the web client and the mobile client. Access to shipment

information is granted for users of the company who sent the shipment and the company who received it. The data displayed in the web and mobile client is queried from the PostgreSQL database. The PostgreSQL database stores the information, thus, for ease of access, while the Blockchain stores it securely and immutably. All audits on the data should be verified against the information stored in the Blockchain. [3] contains additional information about the system and business processes.

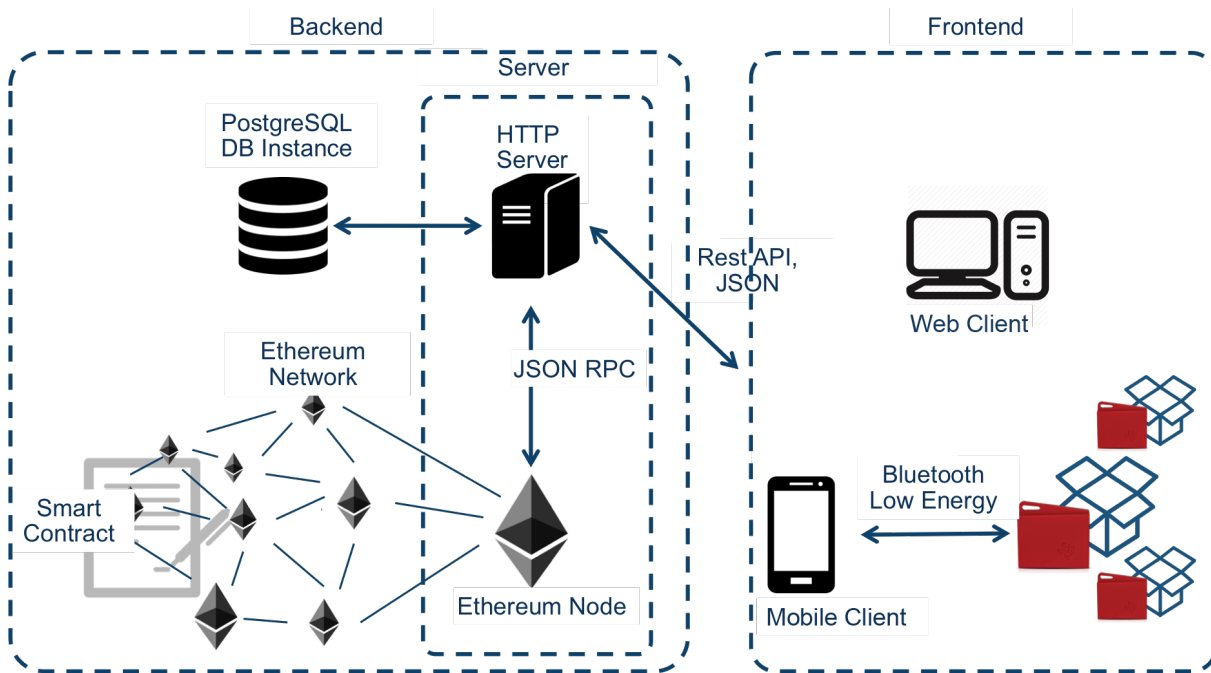


Figure 2.2: System Architecture Diagram

Chapter 3

Analysis

In order to mitigate high level risk 2 in Figure 1.1 it must be impossible for an attacker to modify temperature data that was recorded by a modum temperature logger and it must be guaranteed that the right action is taken based on the data, i.e. release to the customer or destruction of the transported goods. If no data is available for a shipment or if the integrity and authenticity of the data cannot be verified, the transported goods must also be destroyed.

Modification of the data is easily prohibited through means of digital signatures. If the private key signing the shipment data can be kept securely on the logger device, no one can modify the shipment data without invalidating the signature. However, the modum system isn't intended to be used in a way that the head of regulatory affairs at each customer manually verifies all signatures by the logger devices. Instead, an automated component, the Smart Contract, performs these signature checks and returns a binary output, whether to release or destroy a shipment.

Thus, there are two critical components. The logger device must ensure the authenticity and integrity of the data it records and its secret keys must be hard to steal. The Smart Contract must only accept trusted input data from authentic logger devices and always output the correct compliance decision. If these two critical components correctly mitigate security risks, the binary output compliance decision of the Smart Contract can be trusted.

Compromise of the other components, including the client, backend or PostgreSQL database, as depicted in the system outline (Section 2.5), cannot compromise the authenticity and integrity of data since the private keys used to sign all data is not known on those systems. However, other private keys can still be stolen from those systems, as discussed in Section 4.4.2, and, generally, compromise of those systems still poses a risk to availability of the system and confidentiality of the contained data. Thus, proper server security practices must be maintained.

In the following analysis, the two critical components, the logger device and Smart Contract, are analyzed separately. This analysis is partially based on findings in [4] and [5]. Since these works, much more details about the logger device production process, deployment at the customer and the concrete logger hardware have been finalized, and the implementation of security measures proposed here incorporates the latest findings.

3.1 Logger Device

Medical devices are subject to an array of quality standards [17, 18, 19]. These standards target the safety of the medical devices, e.g. they aim to ensure that medical devices are highly reliable, made of safe materials (if implanted) and that malfunction is detectable.

On the other hand, not a lot of focus has been placed on the security of medical devices [?]. Examples of security aspects include the confidentiality of patient data and the resilience to hacking of a medical device. An insulin pump is an example of a device vulnerable to hacking, where it was found that a hacker could administer unauthorized doses of insulin to patients [16]. Clearly, lack of security is a safety flaw often overlooked in medical devices.

In addition to conforming to industry-typical safety standards, it is a goal of the currently ongoing design and development of the modum system’s logger devices that they provide a higher level of security than most medical devices.

3.1.1 General Logger Interaction

Every logger has a real-time-clock that is synchronized with a trusted time source during production. The time can be resynchronized later when the logger is in the field. The real time clock is to assign to each shipment an associated start and stop timestamp. Only a time-synchronized logger can be used to track shipments and a reboot of the logger resets time-synchronization, yielding rebooted loggers unusable.

Table 3.1 lists the shipment parameters that can be set for each shipment.

Name	Description	Required
Tacking Number	The shipment identifier allocated by the logistics provider	optional
Interval	The time period between two temperature measurements	mandatory
Alarm Criteria	The temperature thresholds that the medicines must remain in for the duration of the shipment	mandatory
Options	Additional options for the temperature recording (such as flight mode (no Bluetooth advertising))	mandatory
Operator IDs	Identifiers for the operators starting and reading out a shipment (for reasons of traceability)	optional

Table 3.1: Shipment Parameters

In addition, Table 3.2 details the additional data that is generated by the logger for each shipment.

Name	Description
Temperature Log	The logged temperature measurements
Nonce	A unique identifier for the shipment as generated by the logger
Status	Flag for whether an alarm was triggered according to the set alarm criteria, flag whether the stored temperature measurements reached the maximum capacity of the flash memory used to store them (temperature log is cropped at the end).
Start & Stop Timestamp	Timestamps when the shipment was started and terminated

Table 3.2: Shipment Data Generated by the Logger

Shipment parameters can only be set when no recording is currently ongoing.

The nonce is a 16 byte randomly generated value that is different for each shipment. Since tracking numbers may not be unique between different logistics providers and aren't even guaranteed to be unique beyond 6 months within certain logistics providers, the tracking number cannot be used to uniquely identify a shipment whereas the nonce can.

The operator ID is set if the enable command (to start a recording) is written using an authentication MAC (see Section 4.3.4.1). Otherwise, the operator ID is set to 0, indicating an anonymous logger interaction. Since only the enable command is authenticated, a man-in-the middle could inject other shipment parameters before forwarding the correctly MACed enable command. However, the operator ID is only used for traceability of operators, doesn't affect authenticity and integrity of logger data (which is signed using the logger's private ECC key) and the range of Bluetooth Low Energy (BLE) communications is limited. Thus, this tradeoff was selected for this feature.

A logger can be read out (i.e. the temperature log retrieved) when the recording is stopped. A logger can be read out repeatedly and without limitation in time (but not after a new recording has been started). Writing any shipment parameter resets the temperature log to an empty one to ensure consistency between a shipment's temperature log and its parameters.

A signature is generated over all temperature measurements and shipment parameters to ensure the integrity and authenticity of the shipment data. The data covered by the signature includes the temperature measurements, tracking number, interval, alarm criteria, status flag `is_full` (but not alarm triggered), nonce, start & stop timestamp, fingerprint of the timestamping server certificate and operator IDs. Logger identifiers such as the MAC address don't need to be part of the signed data since they are integrated in the certificate issued for the logger public key (see Section 4.4.2.5). Section 4.1 gives full details on the data structure used for obtaining the hash that is input to the signature generation algorithm.

The signature is generated when the user reads the correspondig BLE characteristic, if

no recording is currently ongoing. Thus, with multiple reads of the BLE characteristic multiple signatures can be generated for a single shipment. A timestamp is included in the signed data every time a signature is generated. The time used for timestamping is taken from the logger's internal real-time-clock. Since the temperature log is erased whenever a shipment parameter is written by the user, consistency of the signed data is achieved.

3.1.2 Hardware Security Module

Each logger is equipped with a hardware security module (HSM). The HSM is designed to generate and store secret keys and to resist tampering, decapping and probing so that these keys cannot be retrieved. This tamper-resistance is assumed to be unbreakable in this work and, thus, the secret keys irretrievable.

Since the Ethereum platform uses ECC with the curve `secp256k1` and only signatures using that curve can be verified in Smart Contract code, a HSM was selected that supports this curve, despite most HSMs on the market only supporting different curves.

The advantages of using the HSM over implementing cryptography in software on the MCU are as follows:

- Prohibit cloning of loggers. I.e. if cryptography is implemented in software on the MCU, stealing a logger's secret keys allows producing an arbitrary number of fake loggers that produce authentic signatures using the same secret keys. If a HSM is used, even if all other logger components can be subverted and the HSM can be made to sign arbitrary temperature measurements, only one logger device is affected. No clones can be manufactured since the secret keys cannot be retrieved from the HSM.
- Cryptographic computations may be more power efficient since they are accelerated by dedicated hardware.
- Cryptographic computations may be faster since they are accelerated by dedicated hardware.
- The burden of correctly implementing cryptographic algorithms doesn't lie with modum.

The MCU has to authenticate itself to the HSM for every use of the HSM before it can be used (this lasts until the MCU cuts power to the HSM or terminates the secure channel session). The HSM also authenticates itself toward the MCU (mutual authentication) to prevent man-in-the-middle attacks. A pre-shared symmetric key and challenge-response are used for authentication (see Section 4.3.2). The pre-shared-key is different for each logger and HSM. The key is only kept in RAM. Thus, a logger reset kills the key, yielding reverse-engineering efforts harder.

3.1.3 Risk Analysis

This analysis covers the risks of an operator or attacker attempting to remove unfavorable measurements from the temperature measurements log or attempting to modify the logger device to not record unfavorable measurements. The risk analysis is divided in three classes of attacks: those without direct access a logger device (e.g. those in charge of regulatory affairs), those with access to one (or multiple) logger devices (e.g. the operators) and those with access to one (or multiple) logger devices and the knowledge and intention to crack them open.

3.1.3.1 Manipulation Without Logger Access

Risk No.	Description	Mitigation Steps
1	Change temperature data before upload to Smart Contract	1
2	Upload temperature data from a different logger	2
3	Replay old temperature data from the same logger	3,4

Table 3.3: Risks of Manipulation Without Logger Access

1: Change temperature data before upload to Smart Contract Temperature data is modified to remove unwanted temperature measurements.

2: Upload temperature data from a different logger Temperature data recorded by a different logger is uploaded because it contains less unwanted measurements than the logger that actually monitored a shipment. This could include storing a log of only favourable temperatures and always uploading that for all shipments.

3: Replay old temperature data from the same logger An old temperature log is uploaded instead of the correct one for a given shipment. The log was recorded by the same logger device.

3.1.3.2 Manipulation With Logger Access

No.	Description	Mitigation Steps
4	Separate logger from shipment	
5	Switching loggers	2,3
6	Restarting logger (with different parameters)	3,4,5
7	Replaying previous BLE communication	4,6
8	Critical bug in logger firmware	14

Table 3.4: Risks of Manipulation With Logger Access

4: Separate logger from shipment The logger that is monitoring a shipment is transported apart from the shipment, e.g. in a temperature controlled box in the truck driver’s cab.

This risk cannot be mitigated on a software level alone. The easiest mitigation approach is to define processes at the logistics companies that check for these violations. An alternative approach could be to minimize the size of the logger device and physically fix it to the packaging of the medication that is being monitored.

5: Switching loggers A logger where unfavorable temperatures are suspected is swapped with a different one. This is equivalent to risk 2 (Upload temperature data from a different logger).

6: Restarting logger (with different parameters) A logger where unfavorable measurements are suspected is restarted partways through the shipment, optionally with different parameters.

7: Replaying previous BLE communication A previously recorded BLE communication is replayed towards the logger or the client.

8: Critical bug in logger firmware A critical bug in the logger firmware could, for example, allow changing of shipment parameters after a shipment has already been started. For example, the alarm criteria could be changed after the shipment has arrived.

3.1.3.3 Manipulation With Electronics Access

Manipulations that target logger electronics incur much higher costs and work than the other two categories. However, since some pharmaceutical shipments are worth multiple

millions of USD (usually on large consignments, e.g. in ship containers), there may be a strong incentive for logistics providers to find a means to hide unwanted temperature measurements on the logger to avoid having to destroy expensive shipments. Thus, this class of manipulation must also be assessed for risk.

For parties with physical access to the logger and the knowledge and means to tamper with the hardware, it is easy to obtain a dump of the logger's flash memory (of the MCU – the HSM's flash memory is designed to be resistant against such attacks).

The block protection feature of the MCU is designed to disable the debug interface and prohibit the read out and write of flash memory through any other interface than the on-chip software [20]. But this feature had a weakness in the previous version of the MCU used [21] that allowed a full flash memory dump despite this feature.

Even with block protection enabled, a full erase of the MCU is always possible via the debug interface [20], and this disables block protection as part of the erase. This is usually not a problem because after a full erase there is nothing left on the chip to protect and the chip could be reused for a completely different purpose. But if an attacker manages to obtain a dump of flash memory, they can perform a full erase of the chip and subsequently reflash the firmware. They then have the modum firmware on the MCU with debugger access enabled and block protection disabled, allowing efficient debugging and reverse-engineering.

Even if the block protection feature works as designed, it is feasible to decap the MCU using mechanical filing and/or acid etching [22]. A flash memory dump may be obtained using optical or other methods [23, 25]. [24] gives an estimate of the time required at a university lab of 4 hours.

It is assumed in this work that decapping destroys the MCU in the process. But for obtaining a firmware image, this is OK since the image can be then flashed onto another MCU.

Obtaining a dump of RAM contents is harder. Probing techniques exist to read and/or manipulate electrical signals of busses inside a decapped chip [26]. Probing requires exact a priori knowledge which lines to probe – or a number of devices that can be destroyed before finding success. [26] suggests that probing can be done with a fairly cheap home setup, but the knowledge of the MCU required to read specific data from RAM incurs a steeper “learning curve” than dumping flash memory.

While the equipment required for microprobing - microscope, micropositioners, probes, amplifiers - may sound expensive, the second-hand market makes all attack equipment available even for individuals. This makes it important to research on low-cost protection mechanisms.

<https://www.sec.ei.tum.de/en/research/invasive-attacks/>

The time required to reverse-engineer a binary dump of the firmware varies wildly with the experience of the person attempting it. But it is fairly easy to look for patterns in small parts of code execution, e.g. looking for cryptographic-key-sized areas of RAM to

change right before activity on the bus to the HSM. Thus, we assume that an attacker can effortlessly reverse-engineer a dumped firmware binary.

No.	Description	Mitigation Steps
9	Resoldering HSM onto different logger (or fake logger)	(7),(13),(16),(17),(18)
10	Reverse-engineering firmware dump	(7),(16)
11	Reflashing a logger with new firmware	(17),(18)
12	Cloning a logger	10,17,18
13	Modifying flash memory of a logger	(17),(18)
14	Obtaining a firmware binary dump / full flash dump	(15),(17),(18)
15	Manipulation of measurements in flash	12,(17),(18)
16	Readout of data in RAM	(17),(18)
17	Manipulation of data in RAM	(17),(18)
18	Replacement of components	9,(17),(18)
19	Manipulation of data on the bus	9,13,(17),(18)
20	Tampering with the quartz	(17),(18)
21	Replacing the modum master key certificate	11,(17),(18)

Table 3.5: Risks of Manipulation With Electronics Access

Assumptions In summary, the following analysis is conducted with the following assumptions:

- The HSM is impenetrable
- Decapping destroys a MCU in the process, i.e. data can be extracted from it, but the hole in the MCU can't be re-filled with epoxy and the unit used like dacapping hadn't been performed. This is especially the case when microprobing is used and the glass coating of the data lines has to be destroyed for the probes to connect. (Even more so when layers of the IC have to be removed to access layers beneath – *the simple MCU used in the modum logger likely doesn't have many layers [26]*)
- Dumping flash memory and RAM is feasible at moderate cost for a determined attacker
- Dumping RAM is more expensive and harder for an attacker than dumping flash (since it requires microprobing and more knowledge of the MCU architecture)
Author's note: if a review finds this not to be the case, keys used to authenticate between MCU and HSM can be stored in flash.
- Reverse-engineering a dumped firmware image is feasible

Scenarios In order to discuss the risk imposed from these techniques we add the following scenarios that an attacker might wish to attempt as additional risks:

No.	Description	Mitigation Steps
22	Manipulate measurements of a single shipment	12,(17),(18)
23	Craft a modified logger from the given hardware where all safeguards are disabled and measurements for each shipment can be modified easily (for internal use or with intent to sell it to interested logistics companies)	(10),(16),(17),(18)
24	Craft a large number of non-genuine loggers that look and function like the modum logger, but where all safeguards are disabled and measurements for each shipment can be modified easily	10,(17),(18)
25	Reuse the logger hardware for a completely different purpose (e.g. home automation)	(17),(18)

Table 3.6: Additional Scenarios of Manipulation With Logger Access

9: Resoldering HSM onto different logger (or fake logger) An attacker can resolder the HSM onto a different logger. He can retrieve the bus encryption key for the HSM from the MCU by destructive means, reflash a modified firmware onto another logger’s MCU and swap the co-processors. Thus, one logger is destroyed and another is created that has modified firmware and can still make use of the co-processor.

A combination attack of 9, 14, 10, 11 and 16 is hard to prevent technically. It requires a lot of work from an attacker. However, the reverse-engineering (10) effort only has to be invested once, after which an attacker can combine 9, 11 and 16 repeatedly on multiple loggers.

Since the key used for authenticating the MCU towards the HSM is different for each logger, a RAM dump (16) has to be obtained for each HSM used in this scenario. This also means that for every logger modified in this scenario, another one is destroyed. This suggests, that tracing efforts by modum to try and discover unexpected occurrences of loggers disappearing can be beneficial.

10: Reverse-engineering firmware dump The assumption is that this cannot, ultimately, be prevented. The effort is slowed down by the fact that the internal state machine of a freshly booted logger isn’t in a usable state, the HSM authentication key is lost on reset and the logger isn’t time-synchronized after reset.

One way to get the logger in a usable state is to obtain a full RAM dump from another logger, load it via the debug interface and then resume code execution of the logger firmware. The logger time will be wrong because it depends on the internal state of the RTC peripheral in addition to RAM.

11: Reflashing a logger with new firmware Reflashing a logger after erasing all memory can't be prevented.

12: Cloning a logger This is largely equivalent to 24: Craft a large number of non-genuine loggers that look and function like the modum logger, but where all safeguards are disabled and measurements for each shipment can be modified easily.

13: Modifying flash memory of a logger Assuming the block protection feature works as designed and the firmware doesn't have any exploitable security vulnerabilities that allow an attacker to modify flash, the only way to achieve this is in obtaining a full memory dump of the logger through means of decapping, erasing all memory of a different MCU and reflashing it with the previously obtained dump. Thus, one logger is destroyed to obtain a memory dump and an arbitrary number of loggers and other MCUs can be reflashed with the (modified) dumped image.

14: Obtaining a firmware binary dump / full flash dump This is assumed to be feasible. If done through means of decapping, it destroys one logger in the process.

15: Manipulation of measurements in flash This is largely equivalent to 22: Manipulate measurements of a single shipment.

16: Readout of data in RAM This is assumed to be tricky, but feasible. It destroys one logger in the process.

17: Manipulation of data in RAM A RAM dump of one logger can be loaded into the RAM of a different logger via the debug interface before resuming code execution on the new logger.

18: Replacement of components Replacing, for example, the temperature sensor with a bogus one that returns favorable measurements on the I2C bus.

19: Manipulation of data on the bus Similar to 18. Not that hard on the I2C busses (used for the temperature sensor and accelerometer). Hard on the bus between the MCU and HSM because it is authenticated and encrypted.

20: Tampering with the quartz Similar to 18, for example, replace the quartz with a signal generator to slow down or speed up the logger's internal clock.

There is no concrete attack scenario for this.

21: Replacing the modum master key certificate If the modum master key certificate can be replaced, then the chain of trust used to verify signed data by modum is compromised. This could be used to flash arbitrary firmware over-the-air (OTA). Since the master key certificate is stored in the HSM, this is considered unfeasible.

22: Manipulate measurements of a single shipment The measurements that a logger records are stored in flash. But a hash over all measurements is continuously updated in the HSM. The nonce of a shipment is also generated on the HSM. No measurements can be modified without generating a new nonce and clearing the previous measurements. Under the assumption that the HSM is impenetrable this attack is impossible.

23: Craft a modified logger from the given hardware where all safeguards are disabled and measurements for each shipment can be modified easily The extent of this attack is intermediate to critical (depending on how many loggers are subverted and how easy it is to subvert further loggers after finding out how to subvert one).

Only existing loggers can be affected because subverted loggers still require the presence of a HSM from a genuine logger. The key used for authenticating the MCU toward the HSM is only kept in RAM. Thus, reverse-engineering efforts will fail at first because reflashing a new MCU with a firmware dump doesn't retain RAM.

The steps required once for this attack are:

- Cracking open the casing
- Obtaining a flash dump (14)
- Reverse-engineering the firmware binary (10) in order to understand the use of the HSM authentication key, remove the logging of unfavorable measurements and to present the MAC address and serial numbers that appear in the logger certificate stored on the HSM.

The steps required for subverting each subsequent logger are:

- Cracking open the casing of two loggers
- Obtaining a RAM dump of one logger (16)
- Re-soldering the HSM onto the other logger (9)
- Flashing the other logger with modified firmware (11)
- Time-synchronizing the other logger
- Sealing the other logger in a mended genuine casing (after cracking it open) or in a genuine-looking self-manufactured casing

Because the HSM cannot be cloned, the number of manipulated loggers that can be produced is limited by the number of genuine loggers the attacker has access to.

24: Craft a large number of non-genuine loggers that look and function like the modum logger, but where all safeguards are disabled and measurements for each shipment can be modified easily This is equivalent to risk 12: cloning a logger.

Since the HSM can't be cloned and no secret keys extracted from it, it's impossible to craft a large number of fake loggers that can produce a genuine signature for arbitrary measurements. Loggers can be crafted that look and behave like modum loggers and customers may be defrauded by selling those, but those loggers cannot produce genuine signatures.

25: Reuse the logger hardware for a completely different purpose (e.g. home automation) This is hard to prevent (one could fill the casing with epoxy), but we don't care about this since it is no threat to business goals.

3.1.4 Mitigation Steps

No.	Description	Mitigates Risks No.
1	Signing data on the logger	1
2	Each logger has an individual key pair	2,5
3	The data signed on the logger for a shipment includes all shipment parameters	3,5,6
4	The logger generates a random, unique shipment identifier every time a shipment parameter changes, as well as when a shipment is started	3,5,6,7
5	Shipment parameters can only be changed when the logger is not recording	6
6	Time syncing is performed through a protocol resistant to replay attacks	7
7	A freshly rebooted logger cannot be used for shipment tracking	(9),(10)
8	The recording enable characteristic write is optionally authenticated	
9	Implausible measurements are cross-checked with other sensors	18,19
10	The ECC keypair private key is stored on a HSM	12,(23),24
11	The modum master key certificate is stored on the HSM	21
12	Measurements are hashed in real time on the HSM	15,22
13	Bus communication between the MCU and the HSM is authenticated and encrypted	(9),19
14	Independent code review for logger firmware	8
15	All cryptographic operations are computed on the HSM	(14)
16	The key used to authenticate the MCU toward the HSM is different for each logger	(9),(10),(23)
17	Tamper-evident casing	(9),(11),(12),(13),(14),(15), (16),(17),(18),(19),(20),(21), (22),(23),(24),(25)
18	Trace loggers until recycling	(9),(11),(12),(13),(14),(15), (16),(17),(18),(19),(20),(21), (22),(23),(24),(25)

Table 3.7: Mitigation Steps for Logger Risks

1: Signing data on the logger The temperature data that the logger device records is signed using ECC cryptography using the secp256k1 curve.

2: Each logger has an individual key pair Each logger is provisioned with an individual key pair. Thus, data originating from one logger cannot be substituted with another logger's data.

3: The data signed on the logger for a shipment includes all shipment parameters Including all shipment parameters in the signed data ensures that the temperature data can only be viewed in context of the parameters that surrounded the corresponding shipment, including shipment id and temperature alarm criteria.

4: The logger generates a random, unique shipment identifier every time a shipment parameter changes, as well as when a shipment is started Avoids collisions of shipment identifiers allocated by logistics companies. Also, prevents multiple attempts to record favorable temperatures by setting the same shipment parameters on the logger because the nonce changes every time.

5: Shipment parameters can only be changed when the logger is not recording Prevents changing shipment parameters after the recording has been started, e.g. right before readout.

6: Time syncing is performed through a protocol resistant to replay attacks Prevents time-syncing the logger to a time in the past by replaying an old time-syncing communication.

7: A freshly rebooted logger cannot be used for shipment tracking Makes reverse-engineering the firmware a bit harder. Ensures that logger reboots are detected when the battery level becomes too low.

8: The recording enable characteristic write is optionally authenticated Allows optional traceability for operators interacting with loggers. Makes attempts to pose as another operator non-trivial.

9: Implausible measurements are cross-checked with other sensors Other temperature sensors on the logger are built into the accelerometer and directly into the MCU. Cross-checking the values mitigates switching of components on the logger and manipulation on the I2C bus.

10: The ECC keypair private key is stored on a HSM Prevents the compromise of the key used to sign temperature measurements.

11: The modum master key certificate is stored on the HSM Prevents compromise of the root-of-trust.

12: Measurements are hashed in real time on the HSM Prevents attacks on flash memory to modify unfavorable measurements.

13: Bus communication between the MCU and the HSM is authenticated and encrypted Makes subverting the HSM to sign arbitrary measurements hard.

14: Independent code review for logger firmware Could find bugs that allow circumventing specified restrictions in logger interaction, as well as other problems.

15: All cryptographic operations are computed on the HSM Is required to not allow the corresponding keys to ever leave the HSM. Might mitigate some side-channel attacks.

16: The key used to authenticate the MCU toward the HSM is different for each logger Makes the cost work required to steal a single MCU – HSM authentication key questionable.

17: Tamper-evident casing The logger device is sealed in a casing that is ultrasonically welded shut. Testing has found that it is hard to crack open the casing without leaving visual evidence of damage. Each casing is laser-engraved with a serial number on the front and back half. Thus, both the front and back are individual for each logger. This prevents an attack where one logger is sawed open while damaging the front half only and one logger is sawed open damaging the back half only, yielding an undamaged front and back each. Thus, cracking open the casing without leaving visual evidence of damage or sanding off the laser-engraved labelling is hard. The modum casings are produced by injection molding, thus, 3D-printing a custom casing will likely not yield visually similar results.

18: Trace loggers until recycling This allows detecting too many loggers disappearing inexplicably, which may be used in attempted physical attacks.

3.1.5 Logger Production

During logger production the MCU is flashed with the logger firmware and the HSM is configured and provisioned with the key material and certificates. Risks during production include theft of sensitive key material and injection of arbitrary software.

No.	Description
26	Theft of sensitive material during production
27	Manipulation of software or data during production

Table 3.8: Risks of Manipulation During Logger Production

26: Theft of sensitive material during production The most sensitive material, i.e. the ECDSA measurement signing key, is generated directly on the HSM and cannot be stolen. However, other sensitive material, such as a binary firmware image can be stolen.

27: Manipulation of software or data during production A malicious actor could inject arbitrary firmware during logger production or replace the modum master key root certificate with another certificate. This fundamentally undermines all assumptions about the operation of the logger and negates all security measures.

This cannot be completely mitigated on a technical level.

3.2 Smart Contract

The role of the Smart Contract in the modum system is such that it has full authority over a shipment's compliance status. This places a high requirement of trust on the input data that the Smart Contract bases its decision on. The limited availability of trusted data from the world beyond the Blockchain is a common problem with Smart Contract technology [27, 28]. In the modum system the Smart Contract only accepts data that is signed by an authentic modum logger device and has the modum master key hardcoded to verify the logger's certificate along the chain of trust back to the master key.

The modum system aims to be Blockchain agnostic. Concretely, care is taken in the specification and implementation of the logger, client and backend to not rely on any specific Blockchain platform. The goal is to easily be able to migrate to a different Blockchain platform should the need arise.

The Smart Contract adds to the security already provided by the logger devices in that it provides an easily auditable platform for immutable storage of compliance decisions and hashes over the associated data, as well as an easily auditable central authority for making the compliance decision. In addition, the Smart Contract data is accessible worldwide and relieves the need to deploy additional infrastructure.

3.2.1 General Function of the Smart Contract

Once a Smart Contract is deployed, its code is immutable. Thus, neither its interface nor behavior can be changed. The member variables of a Smart Contract define its state. They are persisted in the Blockchain across calls to the contract's interface. If the code doesn't allow for multiple modifications of state variables, they are immutable. All calls to the Smart Contract interface are executed on the machines of many miners, thus ensuring the consistency of the result of the computation, as well as incurring a transaction fee which is paid in Ether via the unit of Gas. Smart Contracts can have constant methods which must not modify state (but can read it), which can be executed locally and don't incur any transaction fee. In the modum system constant methods are used to allow generation of a compliance decision and verification of the stored hash against confidential input data (since the constant method is executed locally, the input data is not shared with miners).

In the modum system one Smart Contract is created per shipment. The same contract code is always used, which auditors should regularly check by comparing against their own compiled binary. The public modum master key is hard-coded in the Smart Contract and the contract code can verify the authenticity of the uploaded data along the chain of trust back to the master key. During construction, the Smart Contract is supplied a hash of all shipment parameters that are known at the beginning of the shipment, which is stored immutably in the contract state. This includes tracking number, interval, alarm criteria, nonce, start timestamp, fingerprint of the timestamping server certificate and the starter operator ID. At the end of a shipment a compliance decision is computed via a call of a constant method. Then, via non-constant method call, the Smart Contract is supplied a hash of all shipment data (see Section 3.1.1), the previously computed compliance decision and a signature of the hash (and timestamp). All of these are stored immutably in the contract state, i.e. this function call is only allowed once per Smart Contract. A constant method call allows verification of the stored hashes against the confidential shipment data, the certificate of the logger that signed the data and the provisioning certificate completing the chain-of-trust back to the master key. If the hashes don't match the shipment parameters or shipment data, if the signature isn't valid for the temperature report hash, if the certificate can't be verified along the chain of trust to the master key (including checking the certificate types (see Table 4.2)), or if the stored compliance decision doesn't match the shipment data, an exception is thrown and the auditor should investigate the issue.

A second Smart Contract keeps a list of fingerprints of revoked certificates. The Smart Contract code only allows adding to the list of revoked certificates, i.e. a revocation cannot be undone. Any signature and certificate verification of a Temperature Smart Contract consults the Certificate Revocation Smart Contract and fails if a certificate in the chain was marked as revoked.

3.2.2 Risk Analysis

No.	Description	Mitigation Steps
28	Loss of Ethereum availability	(19),21
29	Loss of confidentiality through Smart Contract data	22
30	Compromise of the owner key of the Smart Contract	19,20,23
31	Hack / buggy contract code	21,23,24,26,30
32	Temperature data can be overwritten	25
33	Modification of the master key	23
34	Duplicates of the modum contract are deployed	20,23
35	Replay of old data	27,29
36	Hard fork	19,28

Table 3.9: Risks of Manipulation of the Smart Contract

28: Loss of Ethereum availability There are scenarios where a transaction could fail to execute or to be created. The cost of operations in the Ethereum virtual machine could change, Ethereum could be abandoned by traders and miners for another system, or it could be forgotten to top up the Ether balance of the Ethereum node broadcasting the modum transactions to the Blockchain. But the integrity of the data generated by the loggers is still protected by the signature computed on the HSM. Thus, a loss of Ethereum availability is only a (temporary) inconvenience for stakeholders verifying the integrity of logger data and the corrections of the compliance decision. The Ether balance could be topped up and the transaction reattempted. Since the modum system is designed to be Blockchain agnostic, moving future operations to another blockchain system is easy. The stakeholders, regulatory agencies and auditors will have to be notified to monitor another Smart Contract on another platform – and they should re-audit the Smart Contract whenever any change is made to it – but there is only a temporary disruption. In addition, if Ethereum is abandoned by traders and miners, nothing changes the integrity of old data stored in its Blockchain by modum. The data still cannot be changed without the immense work of re-mining an alternative branch and anyone with a copy of the Blockchain can still look up the stored data.

29: Loss of confidentiality through Smart Contract data Since only a hash of confidential data is sent to the Smart Contract, no confidential information is leaked via the Smart Contract. Section 4.1 details the data structure used to hash all the data. Since hashing only provides pseudonymity, e.g. a hashed operator ID will always be the same for the same operator, some data is hashed together with a salt to ensure full confidentiality. The only information that can be gathered from the Smart Contract is whether a shipment (not traceable to any specific shipment nor logistics company) complied with the criteria.

Thus, for example, the percentage of compliant shipments among all shipments can be computed from public Smart Contract data.

30: Compromise of the owner key of the Smart Contract The account that owns the Smart Contract associated with a shipment doesn't have to possess a lot of Ether at any point in time. It can be topped up regularly to ensure a balance that allows it to operate for a few days. Thus, theft of Ether is not a big concern. But compromise of the owner key allows any party to upload manipulated data to Smart Contracts for shipments currently underway, as well as to deploy new Smart Contracts.

Uploading manipulated data to existing Smart Contracts has limited use since the data needs to be signed by an authentic modum logger and the nonce needs to match for the shipment. Thus, no data can be injected that doesn't immediately fail integrity checks. Failing integrity checks on purpose could be used for denial-of-service, and would require immediate migration to another Smart Contract (implying notification of stakeholders and re-auditing the new Contract), but cannot lead to catastrophic failure.

Deploying new Smart Contracts with the modum key, could, for example, introduce Smart Contracts that appear to perform valid compliance checking, while actually not verifying the signature of uploaded data, thus allowing upload of arbitrary data. Regular checking of the binary code of the deployed contracts makes it possible to discover this attack very quickly, however. As soon as the attack is discovered, the use of this key must be discontinued, a new key generated and all stakeholders notified.

In the future, it should be evaluated to have the logger devices themselves sign transactions for the Smart Contract. The logger devices can have individual ECC-secp256k1 key pairs generated on the HSM whose public keys are then imported as an Ethereum address and some funds transferred to. The benefit is that only the HSM knows the private key and can sign transactions. The transactions can be broadcast to the Blockchain by any node, so the loggers don't necessarily have to become Ethereum nodes nor know about the Blockchain at all. They just sign data conforming to the Ethereum transaction format. Drawbacks are increased complexity of the message generated on the logger, as well as the burden of distributing Ether over to all those accounts. Another drawback is the limited Blockchain agnosticism of that solution. A future work or thesis could implement such a system.

31: Hack / buggy contract code To date, no vulnerabilities have been found in the modum Smart Contract code. The Smart Contract code will always be public to allow its public audit. Besides the modum Smart Contract code, vulnerabilities could also be present in other places, including the Ethereum code, Ethereum virtual machine and Solidity compiler. Since equivalent logic to the Smart Contract implementation is implemented in the modum backend and executed for all shipment uploads and temperature reports, any deviation from the expected output of the Smart Contract will be noticed and the appropriate steps taken. This might involve migrating to another Blockchain platform.

In addition, modum employees constantly monitor public channels for any news about hacks or problems within the Ethereum ecosystem. If any problem is found, the Blockchain agnosticism makes it possible to migrate to another platform.

32: Temperature data can be overwritten If through an Ethereum vulnerability or a bug in the contract code, state variables, that are meant to be immutable, can be overwritten, state data can be modified to correspond to manipulated temperature data. If the vulnerability is discovered, a migration to a fixed Smart Contract or other Blockchain platform should be performed immediately. Note that even such a vulnerability cannot change the history of state data, which is fixed in the Blockchain through the immense work required for mining and, thus, the old unmanipulated data can be retrieved directly from the Blockchain. A mitigation to discover such manipulation should be to regularly check the Smart Contract data against copies in the modum backend.

33: Modification of the master key If the public modum master key that is hardcoded in the Smart Contract can be changed, the integrity and authenticity checking of input data is compromised and manipulated data can be uploaded to the Smart Contract. Since it is hardcoded, however, this would be noticeable in the change of the binary code of the contract, which an auditor is required to regularly check for.

34: Duplicates of the modum contract are deployed Anyone can compile the publicly available modum Smart Contract code and deploy it. The use of that is limited since the Smart Contract only accepts authentic data signed by a modum logger. Thus, the Smart Contract couldn't be directly used to verify shipments from a competitor's logger device data. If someone modified the Smart Contract code to, for example, hard code another master key the binary contract code changes and can be spotted through comparison of the contract binary with one's own compiled version.

35: Replay of old data If the receiver of a shipment reads out unfavorable measurements from a logger, he may be tempted to upload data from an old temperature log instead. This doesn't work because a hash of the shipment parameters is already sent to the Smart Contract when its creation is initiated by the sender of the shipment. Thus, the parameters (especially the nonce) of the replayed shipment data won't match the hashed data uploaded by the sender.

If sender and receiver cooperate in replaying the Smart Contract inputs of an old shipment, the hashes provided by the sender and receiver can easily be checked against previously used hashes. Since a hash collision is sufficiently impossible with the hash function used (SHA-256), any hash collision indicates such a replay attack. Modum checks shipment parameters uploaded by senders and receivers for collisions, but the auditor and regulatory agency should also regularly check for hash collisions indicating this attack. The same applies if the receiver creates a new Smart Contract (through the modum API, for example) initialized with the shipment parameters of the data to be replayed, and subsequently uploads his replayed data there. Checking for hash collisions allows easy spotting of the

attack. In addition, this would cause the Smart Contract created by the sender to become orphaned (shipment never concluded). Modum should regularly look for orphaned Smart Contracts that might indicate problems.

36: Hard fork If, after a hard fork, one branch loses most of the mining power supporting it, manipulation might become easy on that branch. modum's communication strategy should ensure that all stakeholders understand which branch modum will continue to use after a hard fork.

3.2.3 Mitigation Steps

No.	Description	Mitigates Risks No.
19	Shipment data is stored immutably in the Smart Contract	(28),30,36
20	The Smart Contract checks the signature of uploaded data and validity of the logger certificate	30,34
21	Modum system is Blockchain agnostic	28,31
22	No unhashed data stored in the Smart Contract	29
23	Regular checking of Smart Contract instances	30,31,33,34
24	Monitoring Ethereum / Blockchain news channels	31
25	Spot testing Smart Contract data	32
26	Local evaluation of shipment data	31
27	Checking for hash collisions	35
28	Stakeholder mentoring	36
29	Checking for orphaned Smart Contracts	35
30	Smart Contract code is publically auditable	31

Table 3.10: Mitigation Steps for Smart Contract Risks

19: Shipment data is stored immutably in the Smart Contract Once a shipment's data has been reported, it cannot be changed. Since the data is stored in the Blockchain, the integrity of past data is protected even when major problems force a migration to a new platform in the future.

20: The Smart Contract checks the signature of uploaded data and validity of the logger certificate Since the Smart Contract verifies the signature of uploaded

data and the chain-of-trust back to the master key, only authentic data from a modum logger is accepted.

21: Modum system is Blockchain agnostic Being Blockchain agnostic allows migration to another platform if the Ethereum platform becomes unreliable.

22: No unhashed data stored in the Smart Contract Since all shipment data except the compliance decision is hashed before upload to the Smart Contract, the disclosure of confidential data is minimized.

23: Regular checking of Smart Contract instances Regular checking of the deployed binary of Smart Contract instances should be performed by stakeholders to verify that modum keeps deploying the same once-audited Smart Contract for all shipments. Modum should periodically spot-check the binary of deployed Smart Contracts to catch if the Ethereum key was compromised.

24: Monitoring Ethereum / Blockchain news channels Any hacks or problems in the Ethereum ecosystem must be quickly assessed for damage potential necessary mitigation steps taken.

25: Spot testing Smart Contract data Periodically testing Smart Contract data could reveal modification of data that is supposed to be immutable.

26: Local evaluation of shipment data Local evaluation of shipment data in addition to the evaluation by the Smart Contract reveals when the behavior of the implementations diverges.

27: Checking for hash collisions Checking for hash collisions of uploaded shipment data easily marks replay attacks of old data.

28: Stakeholder mentoring The goal of the modum system is to reduce trust required in modum and to allow stakeholders instead to trust the autonomous Smart Contract that they audit. However, using a Blockchain platform comes with certain risks, such as hard forks, that modum must make sure that stakeholders understand. Mentoring stakeholders in their understanding of the technologies used is, thus, a core strategy for ensuring the safety of the system.

29: Checking for orphaned Smart Contracts Accumulation of too many orphaned Smart Contracts (i.e. Smart Contracts that were created for some shipment, but no temperatures reported) can indicate problems and should be regularly checked for by modum.

30: Smart Contract code is publically auditable The code of the Smart Contract is publically auditable for presence of bugs and conformity to specified behavior. All stakeholders should audit the modum Smart Contract.

3.2.4 Limitations

For confidentiality and cost reasons the actual compliance decision of the Smart Contract is computed locally in a constant method. A hash of all data to arrive at the compliance decision is stored immutably in the Smart Contract. Thus, it is feasible to send the wrong compliance decision to the Smart Contract, hoping that the auditor will not re-calculate the hash over the original shipment data. Traces of such an attack can never be erased since they are stored publicly in the Blockchain. In order to mitigate this limitation, the logger could compute the compliance decision and sign it together with the shipment data. If the Smart Contract verifies the signature in a non-constant method (expensive), there is no vector for manipulation except switching out the contract code. This would, however, limit flexibility in the interpretation of the temperature measurements to arrive at the compliance decision. Zero knowledge proofs could be a theoretical solution to allow verification of shipment compliance without revealing the shipment data.

Chapter 4

Technical Details

The purpose of this section is to allow review of protocol specification details, algorithms and data structures. None of the attacks discussed in Chapter 3 are necessary if the system can easily be subverted through a flaw in a protocol.

4.1 Merkle Tree for Signature Generation

In order to protect the confidentiality of shipment parameters and temperatures when they are sent to the Smart Contract, all data is hashed using a Merkle tree structure. The structure allows to reveal different subsets of shipment parameters to different stakeholders while still providing them with all the data to calculate the hash and verify it against the signature generated by the logger. Concretely, the values of some shipment parameters are revealed, while for other parameters only the root hash of the Merkle subtree is provided to the stakeholder.

Figure A.1 visualizes the Merkle tree structure to obtain a hash of all shipment data at the end of a shipment. Figure A.2 visualizes the Merkle tree structure to obtain a hash of the shipment parameters known at the start of a shipment. Nodes are concatenated and hashed to form the parent node.

Since hashing only provides pseudonymity, e.g. a hashed operator ID will always be the same for the same operator, confidential parameters are hashed together with a random 8 byte salt to ensure full confidentiality. When revealing the parameter, the salt is also revealed to allow verification that the given input data produces the correct subtree root hash. A different salt is used for different shipment parameters.

One subtree root hash is filled with random data and reserved for future extension of the structure. For example, if we want to add acceleration measurements to the data structure, we can replace the reserved node (reserved (random number)) with a subtree (subtree root) as depicted in Figure 4.1.

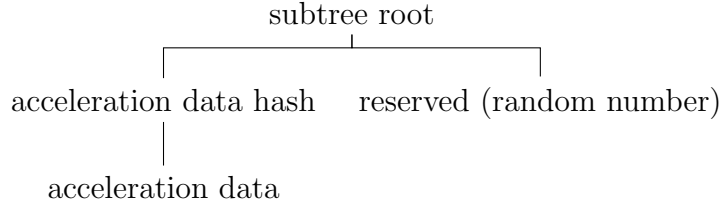


Figure 4.1: Extension of the Merkle Tree Structure

Including additional data in the authenticity and integrity checks can be valuable even when there is no intention to immediately modify the Smart Contract to check this. One such scenario is for experimental data that is not yet ready for inclusion in the Smart Contract.

4.2 Time Synchronization Protocol

All loggers are synchronized with a trusted time source during production. Additionally, they can be time-synchronized in the field. A random 16 byte challenge is generated on the logger device when a time synchronization request is received. The challenge is valid for 20 seconds, after which time synchronization using that challenge is no longer possible. The challenge is forwarded to the time synchronization server which signs a timestamp of the current time and the challenge with his private key. The timestamp, signature and the server certificate are returned to the logger. If the logger can verify the server certificate using the master key and the signature is correct over the timestamp and challenge for the given certificate and if no more than 20 seconds have passed since generating the challenge, the timestamp is accepted and the logger time set to the timestamp.

The protocol was chosen over NTP since NTP is not encrypted nor authenticated by default. NTP could be carried over DTLS (the TLS equivalent over UDP), but NTP server support is questionable.

4.3 Algorithms

This section details the algorithms used in the modum system.

In this section, the following notation is used:

$S_{Key}(Data)$ denotes signature using the Signature Algorithm (Section 4.3.1) using key Key over data $Data$.

$H(Data)$ denotes a SHA-256 hash over data $Data$.

$||$ denotes concatenation.

$HMAC(Data, Key)$ denotes the HMAC-SHA-256 algorithm over data $Data$ with key Key to obtain an 8 byte MAC tag.

$HKDF(Info, IKM)$ denotes HKDF using HMAC-SHA-256 to generate a 32 byte key from initial key material IKM with additional info parameters $Info$.

All operations are generally defined in Big Endian.

4.3.1 Signature

The algorithm used for all signatures is ECDSA with the secp256k1 curve and is applied over a SHA-256 hash of the input data. A timestamp is included in all signed data.

The signature algorithm is used for the following use cases:

- Signing shipment data (see also the Merkle tree structure described in Section 4.1)
- Issuing certificates (see also the certificate layout described in Section 4.4.1)
- Signing the time syncing response on the time syncing server (see also the time synchronization protocol in Section 4.2)

A future use case is the signing of over-the-air (OTA) updates.

4.3.1.1 Cryptograms

The cryptograms are computed as follows:

Signature over Shipment Parameters and Data The signature is computed on the HSM as $Sig = S_{LoggerKey}(H(TS||MTR_1))$ where TS denotes the 8 byte timestamp of signature generation and MTR_1 denotes the root hash of the merkle tree over all shipment data according to Section 4.1.

Issue of a Certificate The signature is computed as $Sig = S_{IssuerKey}(H(CertData))$ where $IssuerKey$ denotes the issuer's key, e.g. the modum master key, and $Cert$ denotes the certificate data as described in Section 4.4.1.

Timesync Server Signature The signature is computed on the time synchronization server as $Sig = S_{TSServerKey}(H(Ch||TS))$ where Ch denotes the 16 byte challenge (as returned by reading the timesync characteristic (which also starts the timer limiting the validity of the challenge)) and TS denotes the 8 byte timestamp (in ms since epoch).

4.3.2 Encryption

Encryption is used for the communication between the MCU and the HSM. These cryptograms conform to the Global Platform Standard Secure Channel Protocol '02'. The security level of the channel is set to MAC and encrypt.

Future use cases are encryption of over-the-air (OTA) updates using AES-CTR and encryption of the BLE traffic using according to the BLE specification.

4.3.3 Hashing

All hashes are computed using SHA-256. The hashes for the Merkle tree structures in Figures A.1 and A.2 are computed on the HSM. Each required data point is input into the HSM as soon as its known, e.g. each temperature measurement as soon as it's been measured.

4.3.4 Message Authentication Codes

HMAC-SHA-256 is used to optionally authenticate the operator that starts or stops a recording. MACing is also for the communication between the MCU and the HSM, according to Global Platform Standard Secure Channel Protocol '02'.

4.3.4.1 Cryptograms

The cryptograms are computed as follows:

Operator Authentication for Start & Stop The start or stop message is computed on the client or on the logger HSM respectively as

$M = En || OpID || HMAC(Ch || En || OpID, OpKey)$ where Ch denotes the challenge, which is the currently set nonce on the logger (as returned by reading the nonce characteristic), En denotes the enable/disable command (1 byte 0x00 for disable or 0x01 for enable), $OpID$ denotes the 16 byte operator id and $OpKey$ denotes the operator key as described in Section 4.3.5.1.

4.3.5 Key Derivation

HKDF is used to derive the operator key that is used as a pre-shared key to authenticate an operator to the logger (see Section 4.3.4.1). The operator only downloads the derived key onto the client (on first login), whereas the logger device can re-derive the key using the stored operator authentication master key and the operator id. Thus, only the modum backend and the logger devices know the operator authentication master key.

A future use case is the derivation of a symmetric key used to encrypt OTA updates.

4.3.5.1 Cryptograms

The cryptograms are computed as follows:

Operator Key The operator key is computed on the modum backend or on the logger HSM respectively as $Key = HKDF(MK, OpID)$ where $OpID$ denotes the 16 byte operator id and MK denotes the pre-shared master key.

4.3.6 Random Number Generation

The following random numbers are generated on the logger:

- Shipment nonce
- Time synchronization challenge
- Salts used in the Merkle tree structure
- Generation of the logger ECC key pair, as well as the pre-shared symmetric key used for MCU – HSM secure channel establishment

All these random numbers are generated directly on the HSM using its internal secure random number generator.

4.4 Key Management

This section gives an overview over all the keys and certificates used in the modum system.

4.4.1 Certificate Layout

A custom certificate format is chosen over X.509 for all certificates handled on the logger for reasons of compactness.

Size (B)	Name	Required
64	Public key	mandatory
1	Certificate type (purpose) byte	mandatory
16	16 byte prefix of SHA-256 hash of additional data (such as logger serial number)	optional (depending on the certificate type byte)
8	Timestamp (date of issue) (ms since epoch)	mandatory
64	Signature over hash of all certificate data	mandatory

Table 4.1: Basic Certificate

Table 4.1 visualizes the basic layout of all certificates handled on the logger. The signature algorithm is ECDSA with SHA-256 with the secp256k1 curve. The expiration date is implicitly derived from the date of issue for all certificate types. The size of the certificate is implicit depending on the certificate type and certificates are encoded as a direct concatenation of their fields. ASN.1 is not used on the logger. The type of the certificate is defined for all logger, client, backend and Smart Contract protocol endpoints and the receiver checks that the length of the data matches the length of the expected certificate.

Valid values for the certificate type are given in Table 4.2.

Value	Description
0x01	Time Synchronization Server Certificate
0x02	Logger Certificate
0x03	Provisioning Certificate

Table 4.2: Certificate Types (in hex)

A certificate's fingerprint is calculated as the 16 byte prefix of the SHA-256 hash of its public key.

4.4.2 Keys

The following keys are deployed in the modum system. Figure 4.2 gives an overview over the relation between keys. Parent-child node relation means that the certificate of the child node key is issued by the parent node key (in case of asymmetric keys) – or that the child node key is derived from the parent node key (in case of symmetric keys).

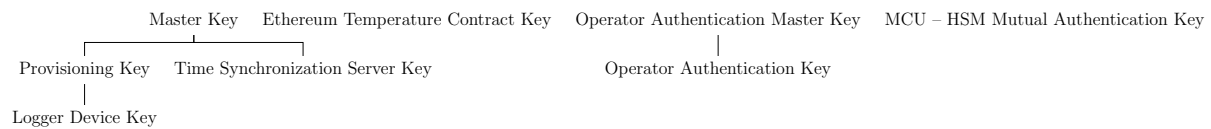


Figure 4.2: Keys Used in the modum System

4.4.2.1 Master Key

This key is the root of trust in the modum system. The modum loggers store the master key in the HSM. The Smart Contract has the master key hard coded. All signature verification is performed along the chain of trust back to the master key.

Type ECDSA asymmetric key pair

Storage and Use The private key is stored on an offline computer and only used to issue certificates based on certificate signing requests (CSR) where the CSRs and signed certificates are transferred to and from the offline computer only via USB drive.

Handling of Compromise This fundamentally undermines all security measures in the modum system. Thus, the master key must be handled with utmost care.

Associated Certificate The associated certificate is a X.509 certificate with company, country and email address information. The validity is 7 years. The certificate is signed by the Verisign certificate authority. The public master key is hardcoded into all modum systems that perform signature checking. Thus, the chain of trust only needs to be evaluated up to the modum master key for all operations in the regular operation of the modum system.

The certificate is published on the modum website and on GitHub.

4.4.2.2 Provisioning Key

This key is used to issue certificates for new loggers during their production.

Type ECDSA asymmetric key pair

Storage and Use The private key is stored on the provisioning server. For each logger produced, a connection is established from the logger device production facility to the provisioning server. The connection is protected by HTTPS, VPN and IP address whitelisting. The logger public key is transmitted, the certificate issued and stored in a database operated by modum, as well as returned to the production facility to be stored on the logger.

The remote connection from the production facility is preferred over distributing the provisioning private key directly to the production facility. This approach allows to inspect the logs of signing requests and to deny signing of logger device certificates when misuse is suspected.

Handling of Compromise Compromise of this key allows the creation of seemingly authentic loggers.

A new provisioning key is generated and the fingerprint of the old one added to the certificate revocation Smart Contract.

Associated Certificate The certificate conforms to the basic certificate format (Table 4.1) without the additional data field. The validity is 6 months from the date of issue. The certificate is signed by the modum master key.

This certificate must be supplied to the Smart Contract along with the logger certificate for each temperature report to allow verifying the chain of trust back to the master key.

4.4.2.3 Time Synchronization Server Key

This key is used to time-synchronize a logger device during production or in the field. The time synchronization protocol is detailed in Section 4.2.

Type ECDSA asymmetric key pair

Storage and Use The private key key is stored on the time synchronization server.

Handling of Compromise Compromise of this key allows syncing the modum Logger Devices with arbitrary timestamps. This then allows to record temperature measurements that are timestamped with the timestamps chosen by the attacker. It does not allow the manipulation of temperature measurement values, however.

A new Logger Device Timestamping Key is generated and the fingerprint of the old one added to the certificate revocation Smart Contract.

Associated Certificate The certificate conforms to the basic certificate format (Table 4.1) without the additional data field. The validity is 6 months from the date of issue. The certificate is signed by the modum master key.

The certificate fingerprint is stored on logger device on successful time synchronization and included in the data that is signed when the measurements are read out. Thus, it can be verified by the Smart Contract that the timestamps for the measurements are based on a valid synchronization.

4.4.2.4 Ethereum Temperature Contract Key

This key is used by the server running the full Ethereum node to create and update the Temperature Smart Contracts. The Certificate Revocation Smart Contract can be operated by the same key. Since revocations of certificates cannot be undone, compromise of the key doesn't pose a large threat to the Certificate Revocation Smart Contract.

Type ECDSA asymmetric key pair

Storage and Use The private key resides on the server machine that runs the Ethereum node.

Handling of Compromise This is discussed in Section 3.2.2 (paragraph '29: Compromise of the owner key of the Smart Contract').

Associated Certificate This key doesn't need an associated certificate. A plaintext message citing the currently used Ethereum Temperature Contract Key in form of an Ethereum address is signed with the modum master key and published on the modum website and on GitHub.

4.4.2.5 Logger Device Key

This key is generated inside the logger device HSM during the production process. It is used to sign temperature measurements to ensure that they cannot be manipulated and that they stem from an authentic modum Logger Device.

Type ECDSA asymmetric key pair

Storage and Use The private is generated directly on the HSM of the logger device during production and cannot be retrieved. Furthermore, a different key pair is used for each logger device.

Handling of Compromise Compromise is assumed to be unfeasible (i.e. economically not viable) given the assumptions about the HSM. If a logger device's casing is visually damaged, the fingerprint of the certificate is added to the certificate revocation Smart Contract.

Associated Certificate The certificate conforms to the basic certificate format (Table 4.1). The associated data field is filled with a 16 byte prefix of the SHA256 hash of the concatenation of logger MAC address, MCU serial number, HSM serial number, temperature sensor serial number, accelerometer serial number, device serial number as assigned by the manufacturing contractor, production lot number, device hardware revision and the hardware expiration date (derived from the battery shelf life and temperature sensor calibration validity, computed during manufacture). These values are stored unhashed in the modum backend when the certificate is issued. The certificate is signed by the provisioning key currently in use.

The certificate is stored immutably on the logger's HSM.

4.4.2.6 Operator Authentication Master Key

This is a pre-shared symmetric key between modum servers and the logger devices. It is used to derive operator keys (see Sections 4.4.2.7 and 4.3.5.1) to enable operators to authenticate towards the logger. The operators only receive the derived key.

Type Symmetric pre-shared key

Storage and Use The key is stored on the modum backend server and on the logger HSM.

Handling of Compromise Since compromise of the operator authentication feature doesn't interfere with any other feature of the modum system, a pre-shared key is acceptable for this use case. The operator authentication feature should be discontinued if the key is compromised.

4.4.2.7 Operator Authentication Key

This is a pre-shared symmetric key between the client and the logger devices. It allows an operator to authenticate his interaction with the logger devices. The feature is used for traceability of logger interaction and is optional.

Type Symmetric pre-shared key

Storage and Use The key is only stored on the operator's client. The logger derives the key from the stored master key (see Section 4.4.2.6) and the operator id that is submitted during an authenticated write.

Handling of Compromise See Section 4.4.2.6.

4.4.2.8 MCU – HSM Mutual Authentication Key

This key is used to authenticate and encrypt connections between a logger's MCU and HSM. This makes it hard to reuse the HSM on another device or to reuse the HSM after tampering with the logger.

Type Symmetric pre-shared key

Storage and Use The key is stored in the logger's RAM and on the HSM. The key is randomly generated during logger device production.

Handling of Compromise This scenario is discussed in Section 3.1.3.3.

Chapter 5

Summary and Conclusions

A security analysis of the modum system was conducted. The logger device and Smart Contract were identified as the critical components in the system. The signature over shipment data computed by the logger device ensures data integrity and authenticity as long as the secret key cannot be stolen or the HSM subverted. The Smart Contract acts as a single point of authority over a shipment's compliance decision and stores the associated data immutably.

The conducted analysis finds that for both, the logger device and the Smart Contract, almost all attacks can be mitigated. The remaining attacks on the logger device are costly, require a lot of knowledge and work and have a limited extent – i.e. manufacturing a large number of fake logger devices is impossible.

The attacks on the Smart Contract are easily mitigated by regular checks of the deployed Smart Contracts. Migration to another Blockchain platform is possible, thanks to the Blockchain agnostic nature of the modum system.

Overall, evidence suggests that the modum system is secure against all but very costly attacks. The production environment must be trusted and attack vectors in the production environment cannot be completely mitigated on a technical level.

The documented technical details in Section 4 allow an independent reviewer to gain a good understanding of the specification details of cryptographic structures in the modum system. Together with the description of security measures in Chapter 3, it should provide a valid starting point for a security review of the system. A review of specification details from Chapter 4, high level decisions from Chapter 3 and a code review of the Smart Contract and logger firmware covers most surfaces where security vulnerabilities could be found.

Bibliography

- [1] Arzneimittel Bewilligungsverordnung, <https://www.admin.ch/opc/de/classified-compilation/20011780/index.html>, Visited May 2018.
- [2] Guideline of 5 November 2013 on Good Distribution Practice of medicinal products for human use, <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:C:2013:343:0001:0014:EN:PDF>, 1.5.2016, Visited May 2018.
- [3] A. Knecht: Securing Goods Distribution with Smart Contracts and Sensors, University of Zurich, October 2016.
- [4] A. Knecht: modum.io Software on the CC2650 SensorTag, University of Zurich, February 2017.
- [5] M. Heimgartner, L. Ruppen, Y. Belevskiy: Security and Front-End Enhancements for Blockchain-based Cold Chains, University of Zurich, September 2017.
- [6] What's the difference between safety and security?, TÜV Nord, <https://www.tuev-nord.de/explore/en/explains/whats-the-difference-between-safety-and-security/>, Visited May 2018.
- [7] Elliptic Curve Cryptography, https://en.wikipedia.org/wiki/Elliptic_curve_cryptography, Visited May 2018.
- [8] Elliptic Curve Digital Signature Algorithm, https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm, Visited May 2018.
- [9] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>, Visited May 2018.
- [10] Block Cipher Modes of Operation, https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation, Visited May 2018.
- [11] HMAC: Keyed-Hashing for Message Authentication, IETF, <https://www.ietf.org/rfc/rfc2104.txt>, Visited May 2018.
- [12] HMAC-based Extract-and-Expand Key Derivation Function (HKDF), IETF, <https://tools.ietf.org/html/rfc5869>, Visited May 2018.
- [13] X.509, <https://en.wikipedia.org/wiki/X.509>, Visited May 2018.

- [14] modum <https://modum.io>, Visited May 2018.
- [15] O. Jones: The Need for Increased Investment in Medical Device Security, <https://www.tripwire.com/state-of-security/featured/need-increased-investment-medical-device-security>, Visited May 2018.
- [16] J&J warns diabetic patients: Insulin pump vulnerable to hacking, Reuters, <https://www.reuters.com/article/us-johnson-johnson-cyber-insulin-pumps-e/jj-warns-diabetic-patients-insulin-pump-vulnerable-to-hacking-idUSKCN12411L>, Visited May 2018.
- [17] ISO 13485:2016, <https://www.iso.org/standard/59752.html>, Visited May 2018.
- [18] ISO 9001:2015, <https://www.iso.org/standard/62085.html>, Visited May 2018.
- [19] GAMP5 Guidance Documents, ISPE, <https://www.ispe.org/publications/guidance-documents>, Visited May 2018.
- [20] nRF52832 Product Specification v1.4, Nordic Semiconductors, http://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf, Visited May 2018.
- [21] K. Brosch, Firmware dumping technique for an ARM Cortex-M0 SoC, <http://blog.includesecurity.com/2015/11/NordicSemi-ARM-SoC-Firmware-dumping-technique.html>, Visited May 2018.
- [22] Decapping ICs (removing epoxy packaging from chips to expose the dies), Applied Science, <https://www.youtube.com/watch?v=mT1FStxAVz4>, Visited May 2018.
- [23] The Decapping Project, http://members.iinet.net.au/~lantra9jp1_nbn/gurudumps/decap/index.html, Visited May 2018.
- [24] Where to store sensitive keys in a device?, Crypto Stackexchange, <https://crypto.stackexchange.com/questions/48003/where-to-store-sensitive-keys-in-a-device>, Visited May 2018.
- [25] S. P. Skorobogatov: Copy Protection in Modern Microcontrollers, https://www.cl.cam.ac.uk/~sps32/mcu_lock.html, Visited May 2018.
- [26] P. Maier, K.Nohl: Low-Cost Chip Microprobing, https://events.ccc.de/congress/2012/Fahrplan/attachments/2247_29C3-Dexter_Nohl-Low_Cost_Chip_Microprobing.pdf, Visited May 2018.
- [27] J. Slobodnik: How Oracles connect Smart Contracts to the real world, <https://medium.com/bethereum/how-oracles-connect-smart-contracts-to-the-real-world-a56d3ed6a507>, Visited May 2018.
- [28] Darius: How Ethereum contract can communicate with external data source, <https://medium.com/aigang-network/how-ethereum-contract-can-communicate-with-external-data-source-2e32616ea180>, Visited May 2018.

Glossary

AES Advanced Encryption Standard

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ASN.1 Abstract Syntax Notation One – Common way to serialize multi-field (and nested) data in cryptographic applications.

BLE Bluetooth Low Energy

CBC Cipher Block Chaining

CSR Certificate Signing Request

DTLS Datagram Transport Layer Security

ECC Elliptic Curve Cryptography

ECDSA Elliptic Curve Digital Signature Algorithm

Flash Memory Non-volatile memory used on the logger device (same as in many USB drives)

GDP Goods Distribution Practices

HMAC Hash-Based Message Authentication Code

HKDF A Key Derivation Function based on HMAC

HSM Hardware Security Module

HTTPS Hypertext Transfer Protocol Secure

IP Address Address in the Internet Protocol

IT system Information Technology System

MAC Message Authentication Code

MAC Address Media Access Control Address – A 6 Byte address individual for each logger

MCU Microcontroller Unit

NTP Network Time Protocol

OTA Over-the-Air Update

RAM Random Access Memory – volatile memory used on the logger device

RTC Real Time Clock – the internal clock of the logger device that keeps track of time during the entire life of the logger device

SHA-256 Secure Hash Algorithm Version 2, 256 bit variant

TLS Transport Layer Security

UDP User Datagram Protocol

USB Universal Serial Bus

USD United States Dollar

VPN Virtual Private Network

X.509 Certificate A very commonly used certificate format standard.

List of Figures

1.1	High Level Risks	2
2.1	Digital Signatures	6
2.2	System Architecture Diagram	8
4.1	Extension of the Merkle Tree Structure	34
4.2	Keys Used in the modum System	39
A.1	Merkle Tree Structure for All Shipment Data	56
A.2	Merkle Tree Structure for Shipment Parameters	56

List of Tables

3.1	Shipment Parameters	10
3.2	Shipment Data Generated by the Logger	11
3.3	Risks of Manipulation Without Logger Access	13
3.4	Risks of Manipulation With Logger Access	14
3.5	Risks of Manipulation With Electronics Access	16
3.6	Additional Scenarios of Manipulation With Logger Access	17
3.7	Mitigation Steps for Logger Risks	21
3.8	Risks of Manipulation During Logger Production	24
3.9	Risks of Manipulation of the Smart Contract	26
3.10	Mitigation Steps for Smart Contract Risks	29
4.1	Basic Certificate	38
4.2	Certificate Types (in hex)	38

Appendix A

Large Graphics

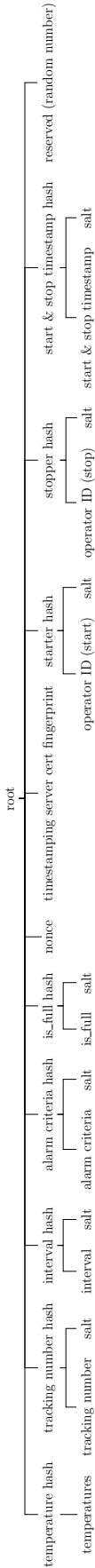


Figure A.1: Merkle Tree Structure for All Shipment Data

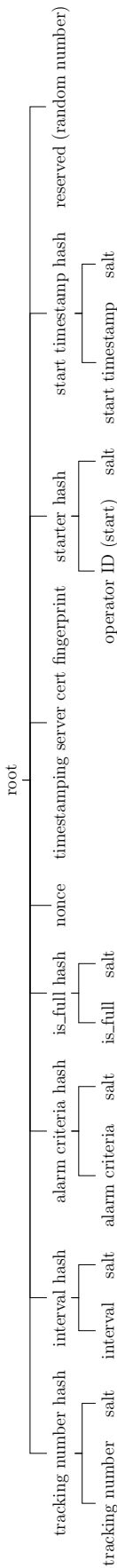


Figure A.2: Merkle Tree Structure for Shipment Parameters