Wirtschafts-
wissenschaftliche
Fakultät

Implementing the Advanced Encryption Standard

Benedikt Kluss, Hendrik Hagedorn

Seminar Thesis – February 5, 2020.
Research Group for Cyber Security

Principal Supervisor: Prof. Dr.-Ing. Thomas Hupperich
Associate Supervisor: Samaneh Rashidibajgan

Contents

1 Introduction

Nowadays, most communication is exchanged via channels which are usually unknown to the users or at least out of their control. Communicating under these insecure circumstances leads to a number of security concerns for all senders and recipients. For example, without any additional security measures, it is typically not possible to know if any third party was able to read or alter the piece of communication at hand. Whilst telecommunication has always been insecure to some extent, the potential value of eavesdropping has increased massively in recent times. The new abilities to automatically process huge amounts of data have lead to a completely new view on personal data in general. Even though more and more sensitive data is processed digitally anyway, at the some time more and more data is also perceived as sensible, since nowadays even the most trivial communication can be turned into value. This means that on a broad scope more than ever before, measures are needed to enable secure communication. Numerous cryptographic techniques exist which provide such functionality. One of the oldest subclasses of these is constituted by the so-called symmetric algorithms. These use the same secret keys for both the encryption and deciphering of data, thus ensuring the privacy and integrity of communication. Given the above-discussed background, they are very promising as they can be used with all kinds of digital communication and offer a high efficiency paired with reasonable security. One of the most prominent symmetric encryption algorithms is the Advanced Encryption Standard (AES). It has been introduced in 2001 and is still well established to the current day.

The work at hand presents the project documentation of implementing the AES as part of a project seminar, held by the Research Group for Cyber Systems at the University of Münster. It depicts the respective project by illustrating its phases from the initial start to the final evaluation of the outcome.

To do so, the following work begins by addressing the theoretical background in section 2. It will introduce symmetric encryption in general and the AES in particular, including its characteristics and a description of the underlying algorithm. The two subsequent chapters cover the project of implementing the AES itself. First, in section 3, the definition and planning of the project are outlined. This includes the definition and analysis of project requirements, a list of all required tasks, and a preliminary project schedule. In the following chapter 4, the project outcomes are evaluated by taking a closer look at the implemented functionality, the resulting project schedule and the key learnings from it. A detailed description of the project deliverables

is not included, but given in the associated implementation documentation. Finally, section 5, provides a conclusion of the overall project.

2 Introducing the AES

The following chapter explains symmetric encryption in general terms and contracts especially the Advanced Encryption Standard (AES) in section 2.2. It includes the historical background of AES, a description of the algorithm itself, and its security and threats.

2.1 Symmetric Encryption

The manner of symmetric encryptions is especially to preserve the privacy of secrets. This aim is followed by humans for a long time. In the era of computer technology, this need even intensifies, as a major part of communication is processed via insecure channels.

More specifically, it aims at achieving the four security goals of data privacy, integrity, authentication, and nonrepudiation [Den07]. These terms can be defined as:

- **Privacy:** Disguising the meaning of a transferred message.
- **Integrity:** Preserving the message's primary meaning.
- **Authentication:** Confirming the identity of the communicators.
- **Nonrepudiation:** A convincing validity of authentication.

Symmetric encryption, unlike asymmetric encryption, uses the same key for encryption and decryption. Using the same key results in higher efficiency than asymmetric approaches offer. Additionally, symmetric encryption is able to offer a certain degree of authentication [sym20]. This is provided as only the person knowing the key can be the author of the message at hand. Therefore, the required trust in the other parties ability to keep the key secret is a significant limitation that needs to be considered.

The model of symmetric encryption consists out of a plaintext, an encryption algorithm, a secret key, cipher-text, and a decryption algorithm [Sta05]. Prominent examples of symmetric algorithms are the Data Encryption Standard (DES), the Advanced Encryption Standard (AES), the Rivest Cipher (RC), the International Data Encryption Algorithm (IDEA), and the Blowfish [Sta05].

2.2 Advanced Encryption Standard

The DES was proposed in 1975 by IBM and declared as Federal Information Processing Standard. It was commonly used until the 90s, where several demonstrations of insecurity were published [Lan00]. DES is written in a block cipher design, using substitution and permutation of a grouping of symbols within its encryption process. Furthermore, DES uses additional XOR for encryption. Blocks of DES consists of 64 bit, the same input size holds for the key. One byte of the key is reserved for parity checking, which effectively decreases the key length to 56 bits. The DES algorithm uses a Feistel cipher and operates on half a block. The cipher is consisting of expansion to 48 bit, key-mixing, S-Box-lookup, and permutation, within each of its sixteen rounds. Especially the length of the key is a weakness of the DES. Due to this issue, standard brute-force attacks have been proven as a practical method to break any DES cipher [DK02]. Such an exhaustive key search has been used by many researchers to prove the weakness of the DES. For example, Distrubuted.Net manages to crack a DES encryption key in around 22 hours with a network of 100.000 computers in 1997 [KPP+09]. This led to the National Institute of Standards and Technologies (NIST) initiating a competition in the same year, searching for a new, Advanced Encryption Standard (AES).

Out of 15 submissions, the Rijndael Block cipher won in late 2000. The name combines the two last names of its inventors Vincent Rijmen and Joan Daemen. Their design is known for its security and especially its speed of processing which led to its election. The criteria for its election by the NIST were subdivided into three categories including the following qualities [Sta10]:

- **Security:** Actual security, randomness, soundness.
- **Cost:** Licensing requirements, computational efficiency, memory requirements.
- **Algorithm and implementation characteristics:** Flexibility, hardware and software suitability, simplicity.

2.2.1 Characteristics

The block cipher algorithm Rijndael uses keys of 128, 196, or 256 bit, depending on the demanded security level, and always performs them on 128 bit plaintext. Based on the selected security level, it repeats its process 10, 12, or 14 rounds, using the respective amount of round keys obtained through key expansion. Figure 2.1 shows the AES algorithm where each round consists of 4 layers: SubBytes, ShiftRows, MixColumn, and AddRoundKey. An exception of the last round is to omit the MixColumn layer which is shown in figure 2.1, as well. ShiftRows is the only permutation of AES, where the remaining are substitutions. The operations of AES are performed on a matrix of 4x4 dimensions. Each byte is interpreted as a polynomial of a finite field.

For example, the byte containing an unsigned value of 255 would be represented as:

$$\{11111111\} \Longrightarrow x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$$

2.2.2 Algorithm Description

The following section engages in the layers of AES, their security contributions for the whole cipher, and the dedicated key expansion. Only the encryption algorithm is explained as the decryption performs the inverses of the same basic operations. For the layers of the algorithm, the 128 bits are assumed as a two-dimensional array of byte entries, which is built column-wise.

SubBytes

The SubBytes layer substitutes each byte with a value from a look-up table. This table consists of 256 entries, each representing a substitute for a value in the range of one byte. It is designed to be invertible for decryption and resistant against known attacks due to a low correlation between input and output [Sta10]. Each entry is calculated by mapping the respective byte to its multiplicative inverse in Rijndaels finite field and applying an affine transformation on it subsequently.

ShiftRows

The diffusion operation of ShiftRows uses a left circular shift on every row, where each row is shifted of i positions (i as an index of an iteration over the rows starting with 0). Thereby each byte, starting at the second row, will be assigned to a new position in a different column. This results in a multiple of four distance along the column build vector over the whole text. This layer provides security against differential and linear cryptanalysis [DR02].

MixColumn

In contrast to ShiftRows, MixColumn operates as the name implies column-wise. Each entry is changed through the use of a matrix multiplication function performed on the whole column. Consequently, if one input byte changes, the complete output of the dedicated column changes. This results in high security, without MixColumn the AES would be easy to break [Den07]. As the MixColumn in the last round wouldn't rise the security level of AES [DR02], it is omitted to contribute higher efficiency.

The following multiplication displays the constant matrix that gets multiplied by the AES matrix to calculate the result of this layer.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \\ s_4 & s_8 & s_{12} & s_{16} \end{bmatrix} = \begin{bmatrix} s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \\ s'_4 & s'_8 & s'_{12} & s'_{16} \end{bmatrix}$$

Which can be inverted with the following multiplication by the decryption:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} * \begin{bmatrix} s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \\ s'_4 & s'_8 & s'_{12} & s'_{16} \end{bmatrix} = \begin{bmatrix} s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \\ s_4 & s_8 & s_{12} & s_{16} \end{bmatrix}$$

AddRoundKey

This layer finally applies a key to the input text. Therefore, it uses the expanded key to perform a bitwise XOR operation between the text and the round key. The security enhancement comes through the expansion of the original key, which will be explained in the upcoming paragraph.

Key expansion

To apply the different key sizes 128, 196, and 256 bits to the 128 bit text, the corresponding number of rounds is adjusted. This results in 10, 12, or 14 rounds requiring an equal number of round keys. To expand the original key, it gets separated into four-byte long words. First, a word-wise cyclic left shift gets applied. Afterwards, a round constant gets applied with an XOR. The round constant gets initialized to a value of one and gets multiplied by two in the Galois field every subsequent round. Then, each byte will be substituted like in the SubBytes layer using the same S-Box as well. The 256 bit key uses an additional substitution with the S-Box on every word where (word % 8 == 4) is true. This algorithm provides asymmetry for the round transformation and round itself and defends against attacks where single units of the cipher key are known [DR02].

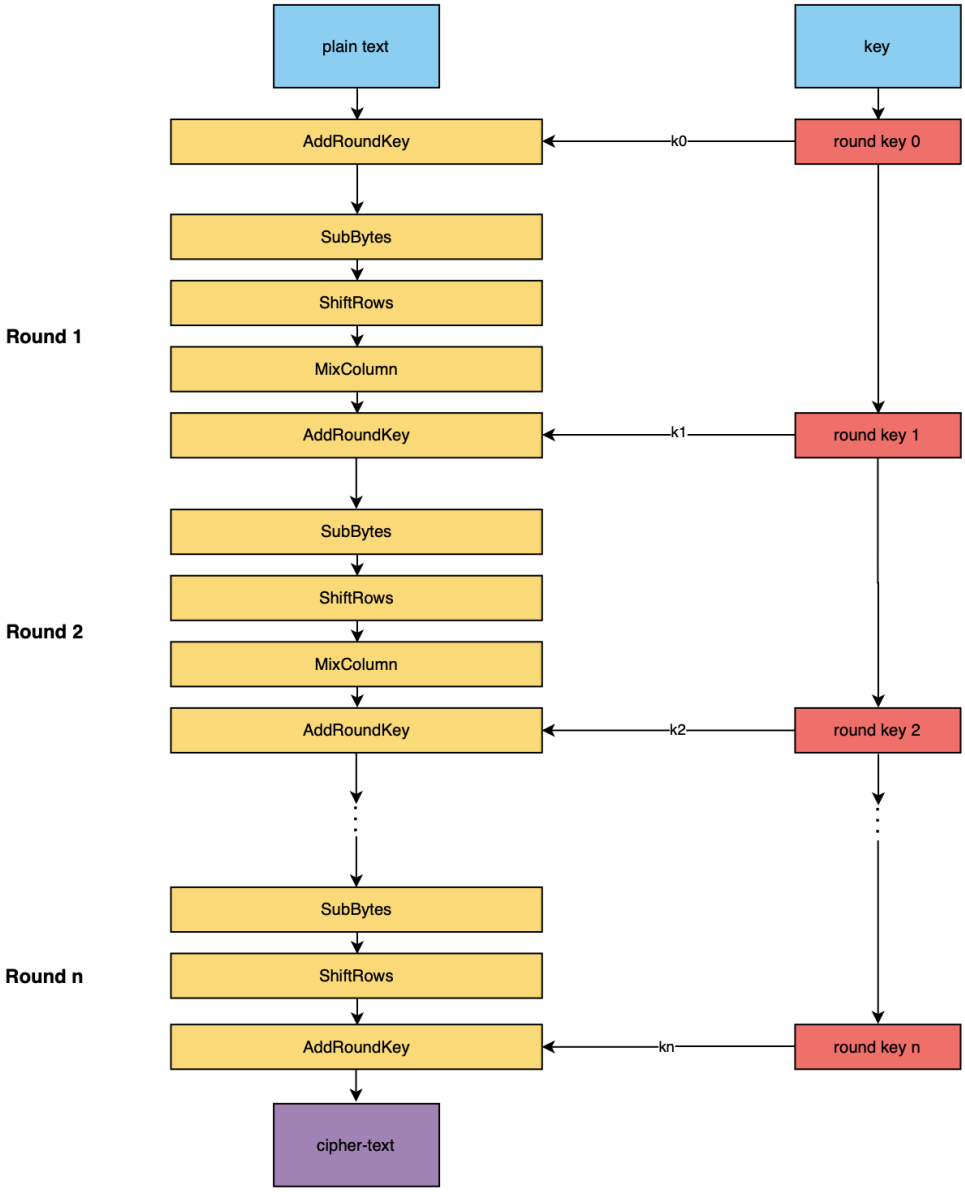

Figure 2.1: Structure of AES Rounds

3 Implementing the AES

In this chapter, the project of implementing the Advanced Encryption Standard is defined and planned. In section 3.1 the general project definition and planning are illustrated, before the resulting project backlog and schedule are presented in sections 3.2 and 3.3.

3.1 Project Definition and Planning

In order to illustrate the project definition and planning, the general project framework will be presented first, describing how the project will be structured and carried out. Subsequently, in subsection 3.1.2, potential product requirements are collected and analysed. Based on these requirements user stories are generated and, together with models of their underlying processes, presented in subsection 3.1.3.

3.1.1 Procedures & Framework

The creation of the project deliverables will be divided into equal slices of time, called sprints (the project schedule can be found on page 15). Different to Scrum [Glo10], which serves as a rough role model for how the project is carried out, here the user stories are not necessarily fulfillable within a single sprint. Instead, this guideline holds true for the product backlog items which the user stories are subdivided into (see section 3.2). Each sprint spans one week and is assigned one or multiple product backlog items upon its beginning. To carry out these sprints effectively, each entails three group meetings. The first meeting, which is scheduled at the beginning of each sprint, aims at selecting the work to be done and defining associated acceptance criteria. The second meeting takes place at half time and is concerned with reviewing the progress and intermediary results, discussing questions and issues that arose in the meantime. The final meeting, at the end of the sprint, evaluates the sprint outcomes. In this meeting, the results are examined and it is decided whether the tasks and goals have been fulfilled, or if their requirements and implementations have to be refined and added back to the product backlog and reworked in a following sprint.

Each task passes a four-stage process. The first stage is the requirements engineering, in which wished requirements are balanced against technical realisability. Based on these refined requirements the two stages of drafting and implementing the required work can be carried out. In the final stage, the implementation is tested, with all test-cases being collected and archived in a central test-routine. These four stages are all carried out at least once per task and sprint, potentially multiple times if one of the later steps leads to necessary adjustments in a previous one. As described above, a backlog item might also be worked over in one or more additional sprints, in each of which all of the four stages would be performed again.

In order to define the scope of the project and subdivide it all the way down to deliverable tasks, a number of steps have to be taken. First of all, potential requirements have to be collected, analysed, and rated. After the general product requirements are defined, user stories can be generated. As subdividing these abstract stories into meaningful tasks is not always as straightforward as expected, the underlying processes required to enable the stories are modeled first. From these more specific, nevertheless still abstract in terms of the actual implementation, models, the product backlog items can finally be derived. This process of defining the backlog entries is documented in the following, beginning with the requirements collection and analysis.

3.1.2 Requirements Analysis

The project team brainstormed potential requirements for the proposed implementation of the Advanced Encryption Standard and evaluated them subsequently. Following the MoSCoW approach, all requirements were prioritised by assigning them to one of the four categories must have, should have, could have, or won't have. Whilst requirements in the "must have" category are absolutely essential to the success of the project and therefore must certainly be met, those prioritised as "should have" entail a high relevance for the project value too, but are not equally critical for the overall success. In case they turn out not to be realisable, the project does not necessarily fail. Less priority is awarded to requirements rated as "could have", as these are less relevant to the overall project outcome and therefore are only considered if free capacities remain after realising the higher priority requirements. Requirements with the currently lowest priority are assigned the "won't have" category. Nevertheless, being assigned to this category does not imply that the given requirement is of less value to the project than those in any other class, but that their implementation is not critical within the current time frame and maybe postponed to future efforts [tG20]. The developed requirements are given in the following, grouped by their assigned MoSCoW-Categories:

1. **Must have**

a) Enable AES Encryption
The implementation of the AES must be able to encrypt 128bit texts with keys of 128 bit size.

b) Enable AES Decryption
The implementation of the AES must be able to decrypt 128bit texts with keys of 128 bit size.

2. **Should have**

a) High Portability
The product should be designed to facilitate high portability.

b) High Efficiency
The software should aim at high efficiency in terms of speed and storage capacity.

3. **Could have**

a) Handle Large Messages
The product could be designed to process in- and outputs of texts larger than 128 bits in size.

b) Dialogue User Interface
The product could offer the implemented AES functionality through a dialogue based user interface.

c) Enable Usage of 192 bit Keys
The product could enable (de-)ciphering with 192 bit keys.

d) Message Exchange Dialogue
The product could offer a dialogue interface, allowing the user to switch between en- and decryption mode in order to use a single continuous session for message exchanging.

e) Generate Secure Keys
The product could provide a key generation functionality, allowing the user to generate keys of desired size.

f) Diffie-Hellman Key Exchange
The product could include a Diffie-Hellman key exchange to enable the secure exchange of symmetric keys.

4. **Won't have**

a) Enable Usage of 256 bit Keys
The product will not be able to encrypt or decrypt with 256 bit keys.

b) Make Available as Library on GitHub
The product design will not aim at being made available as a library via GitHub.

3.1.3 User Stories & Processes

User stories were initially developed for all requirements rated as "must have" and "should have", saving resources by not emphasising "could have" requirements before it is clear if sufficient capacities for their fulfillment remain after finishing higher prioritised work. Where feasible, the processes underlying the user stories where modelled using the Business Process Model and Notation in order to further facilitate the subsequent definition of tasks. The results are given in the following:

1. **Role: Writer**
   "As writer I want to encode written data, to enable secure communication"
   Acceptance criterion: Supposed the writer entered a valid text and key, the data can be encrypted.
   Corresponding requirement: 1a. Priority: Must have.

2. **Role: Reader**
   "As reader I want to decode ciphered written data, to enable secure communication"
   Acceptance criterion: Supposed the reader entered a valid ciphered text and key, the data can be decrypted.
   Corresponding requirement: 1b. Priority: Must have.

3. **Role: User**

   a) "As User I want to be able to (de-)cipher written data on desktop machines."
   Corresponding requirement: 2a. Priority: Should have.

   The underlying requirement and hence the user story as well, do not result in a deliverable task. They rather imply the whole design process to increasingly consider portability.

   b) "As User I want to (de-)cipher text efficiently."
   Corresponding requirement: 2b. Priority: Should have.

   The underlying requirement and hence the user story as well, do not result in a deliverable task. Instead they have implications on the overall design process, to strongly emphasize efficiency in all stages.

3.2 Product Backlog

The following chapter lists the generated tasks and explains how they were derived from the user stories given above.

3.2.1 Encryption

The entries in this section of the product backlog are based on user story 1 and are all assigned the "must have"-priority. In order to facilitate subdividing the user story into meaningful tasks, the underlying process has been modelled first. The result is presented in figure 3.1. Taking a closer look at the model, it is striking that the key expansion has been represented within an individual lane, isolating it from the rest of the process. The reason for this can be found in the fact, that the key expansion is described as a distinct functionality within the AES and is shared with the decryption process described by user story 2, which is further subdivided into section 3.2.2 of the product backlog. Therefore, no corresponding backlog item is defined in both this and the following section, but instead the key expansion is specified in the general section 3.2.3. The items resulting from the rest of the depicted process are the following:

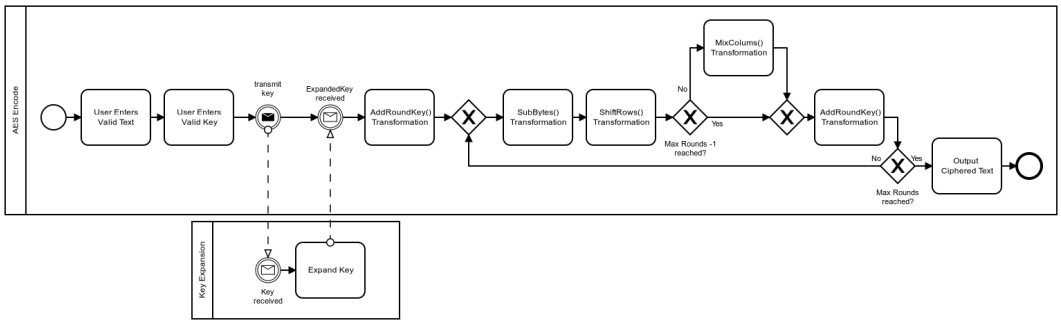

Figure 3.1: Underlying Process of User Story 1

1. **AddRoundKey() Transformation**
   The AddRoundKey() Transformation is one out of four transformations performed within AES encryption. It is described in section 5.1.4 of [oST01].

2. **SubBytes() Transformation**
   The SubBytes() Transformation is one out of four transformations performed within AES encryption. It is described in section 5.1.1 of [oST01].

3. **ShiftRows() Transformation**
   The ShiftRows() Transformation is one out of four transformations performed within AES encryption. It is described in section 5.1.2 of [oST01].

4. **MixColumns() Transformation**
   The MixColumns() Transformation is one out of four transformations performed within AES encryption. It is described in section 5.1.3 of [oST01].

5. **Encryption Control Structure**
   The encryption control structure is responsible for the application of the four transformations and specified in section 5.1 of [oST01].

6. **In- and Output**
   The in- and output task aims at enabling user interaction with the encryption implementation. This refers to the program being able to retrieve the required user inputs and provide useful outputs in turn.

3.2.2 Decryption

The entries in the following section are derived from user story 2 and their priority is rated "must have". Analogous to how the encryption tasks were created, the process underlying the AES decryption was modelled as well and is illustrated in figure 3.2. The model structure is based on the same decisions as explained above (see section 3.2.1), resulting in the key expansion being treated as an independent general module which will therefore be addressed in section 3.2.3 below. The backlog items derived from the remaining process are given in the following:

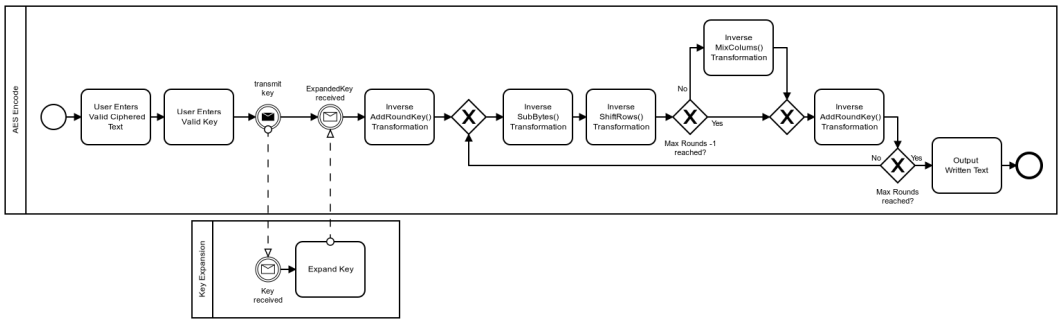

Figure 3.2: Underlying Process of User Story 2

1. **AddRoundKey() Transformation**
   The AddRoundKey() Transformation is one out of four transformations performed within AES decryption. It is described in section 5.3.4 of [oST01] and equals the corresponding transformation of the encryption process.

2. **Inverse SubBytes() Transformation**
   The inverse SubBytes() Transformation is one out of four transformations performed within AES decryption. It is described in section 5.3.2 of [oST01].

3. **Inverse ShiftRows() Transformation**
   The inverse ShiftRows() Transformation is one out of four transformations performed within AES decryption. It is described in section 5.3.1 of [oST01].

4. **Inverse MixColumns() Transformation**
   The inverse MixColumns() Transformation is one out of four transformations performed within AES decryption. It is described in section 5.3.3 of [oST01].

5. **Decryption Control Structure**
   The decryption control structure is responsible for the application of the four transformations and specified in section 5.3 of [oST01].

6. **In- and Output**
   The in- and output task aims at enabling user interaction with the decryption implementation. This refers to the program being able to retrieve the required user inputs and in turn output useful returns.

3.2.3 General

Tasks are assigned to the general category if they produce deliverables required by multiple user stories. In the project at hand, this only holds true for the Key Expansion task depicted below.

1. **Key Expansion**
   Based on User Stories 1 and 2. Rated must have.
   The Key Expansion entails cryptographic key-extension functionality required by both the encryption and decryption processes. It is specified in Section 5.2 of [oST01].

3.3 Project Schedule

Next up, after the definition of the tasks performed above, is the building of a preliminary schedule, illustrating the order and expected duration of fulfilling the project tasks. As the project definition performed in section 3.1 and the planning of how the project will be carried out are considered as part of the project itself, they will be added to the list of tasks. Based on the overall design decisions of the implementation (which are not explained here but in the related implementation documentation), the task of learning the programming language C is added as well. This is required, since it has been decided that the C is the language of choice for the project implementation, even though those conducting the project have got no prior knowledge in this area.

The project schedule, including the above mentioned additional tasks, is illustrated in figure 3.3. As mentioned previously, it is divided into eight sprints of equal size and already includes an allocation of the work to the involved persons Benedikt and Hendrik. As depicted, the first two sprints are expected to only contain work performed by both together. They entail the steps of project definition and planning in the first and learning C and implementing the key expansion in the second sprint. Afterwards, the implementation of the AES cipher layers begins and is equally split between the encryption and decryption part. The encryption tasks will be fulfilled

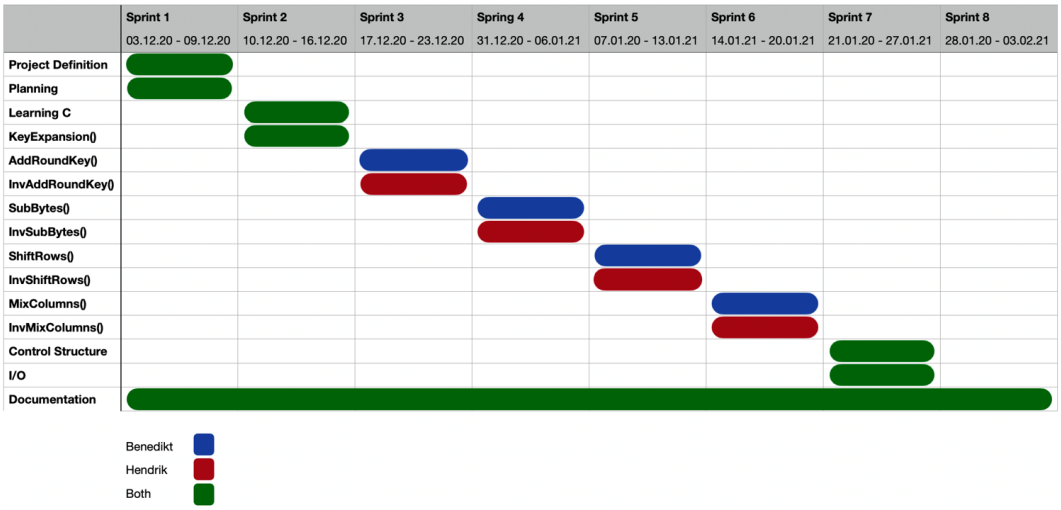

Figure 3.3: Anticipated Schedule for Implementing the AES

by Benedikt whilst Hendrik focuses on the decryption functionality. The tasks representing the four cipher layers are all allocated a single sprint, beginning with the AddRoundKey() tasks and followed by the (Inv-)SubBytes(), the (Inv-)ShiftRows(), and the (Inv-)MixColumns Transformations. After these tasks are accomplished, only two sprints are expected to be left. The seventh sprint will see Benedikt and Hendrik working on the same tasks for their respective implementations again, both implementing their respective control structures and in- and output interfaces. The project is concluded by the eight sprint, which will supposedly only focus on finishing the documentation, which was also included in every other sprint as it will be written continuously throughout the whole project.

As the figure illustrates, no additional milestones were included in the schedule. The reason for this is, that the backlog items themselves indicate a certain order in which they have to be implemented (based on the underlying algorithm), thus making it easy to plot the project progress. This is additionally facilitated by their manageable number. Nevertheless, the intermediary presentations scheduled every two sprints, provide an additional time-frame for scheduled reflection on the overall progress. Their purpose is to break all achieved progress down into small presentations which are strictly limited to ten minutes. Thereby the project team is halted to concentrate on the core achievements and their progress, keeping the scope clearly in sight.

4 Evaluation

Whilst the previous chapter described the beforehand project definition and planning, the actual project outcome will be reviewed in the following. This project evaluation is split into three parts, beginning with an examination of the accomplished project scope. Subsequently, the actual timetable, illustrating how the project was carried out, is presented and discussed. The chapter is concluded by a report on the key learnings.

4.1 Implemented Functionality

Recalling the requirements definition in chapter 3 once again, a project implementation can only be termed successful if it fulfills the requirements rated as "must have". According to the project definition this applies to both 128 bit AES encryption and decryption. Both requirements have been met, as the final deliverable offers the respective functionalities to the user.

High portability and efficiency of the implementation have been deemed important but not absolutely crucial for the project success and thus assigned the "should have" category. As the documentation specifies, the created deliverable can be used in several different technical environments and the requirement of high portability can therefore be considered as fulfilled. High efficiency was rather treated as a guideline throughout the project execution and has been set aside for the sake of simplicity and clarity on a number of occasions. An evaluation, whether or not the requirement has been met, is difficult without any benchmark it can be compared to. Therefore, no clear statement can be made about this.

Taking all these high priority requirements into account, it is fair to denote the project outcome as a success. Additionaly, two more functionalities from the "could have" category were included on top. The final version is able to process data of arbitrary sizes and offers its functionality via a user dialogue interface.

4.2 Final Project Schedule

The updated final version of the project schedule is given in figure 4.1. Comparing it with the anticipated schedule from the planning phase (see figure 3.3 on page

16) shows that the overall structure remained the same, but two deviations have occurred.

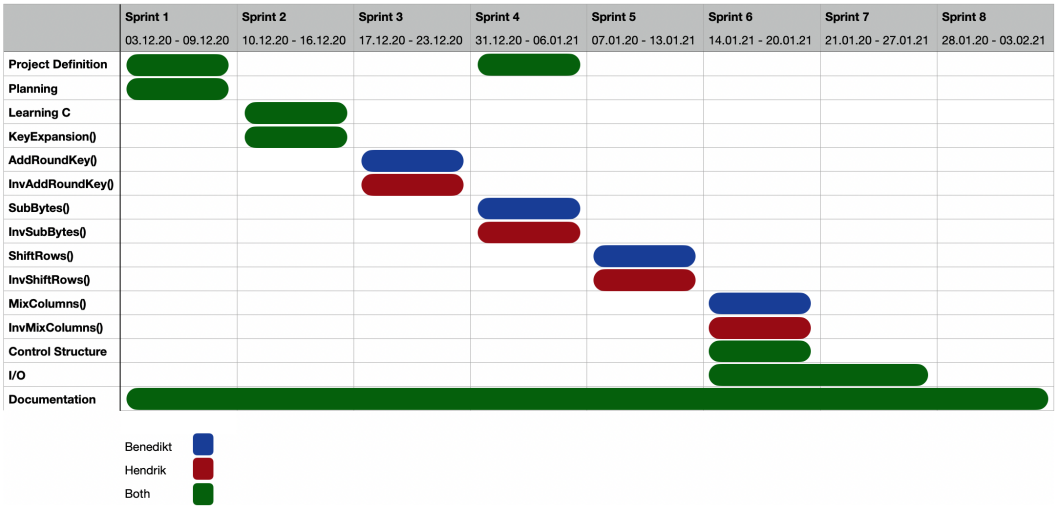

Figure 4.1: Final Schedule of Implementing the AES

The first deviation can be found with the project definition task, as it was not only executed in the first sprint but spanned the fourth sprint as well. An important task within the project definition was the setting up of the structure, the actual code-wise implementation would be developed in. This, for example, included the selection of a fitting IDE and selection of a framework to perform testing with. A misconception occurred in the estimation of the time required, as it showed that the selection and deployment of the test-framework was more complicated than expected. For this reason the project definition was added back to the backlog and reopened in sprint four, where the project structure was updated to include the testcases developed for the CTest environment.

The second divergence from the original schedule occurred for the exact opposite reason. In the mid-way meeting of sprint six, it turned out that the work dedicated to this time-slice in the initial sprint meeting was not as extensive as expected. Therefore, within this meeting, it was decided to already pull the Control Structure and I/O tasks to this sprint. Whilst the Control Structure was completed, only the basic I/O functionality was implemented within the sprint. This required the I/O to be put back into the backlog and the additional handling of special cases was finished in the subsequent sprint number seven.

Accordingly, no significant time deviations occurred throughout the project and it was successfully finished within the given time frame.

4.3 Key Learnings

The key learnings can be divided into two categories, the first referring to all skills linked with the conduction of a team-project. Collaborating in all phases of a project, from definition and planning to implementation and testing, has led to a lot of new learnings and improved knowledge and skills. This entails more general aspects like time-management and communication organisation as well as very specific learnings, like fully understanding the relevance and advantages of a clean feature-branch-workflow when collaborating via Git. Generally spoken, improving our skills concerning the use of Git and its integration with GitHub certainly constitutes one of the biggest learnings from this project. A lot of learnings also arose from the first-time application of knowledge acquired in previous university modules, like for example modelling structures and processes with applications like Visual Paradigm.

Something that we learned and certainly would do differently if we were to undertake the project again is the beforehand project definition. Initially, we precisely defined how our sprints will be organised and gave detailed guidelines for the execution of every single task. It turned out, that in a project conducted by a team as small as two and a scope as small as at hand, this excessive overhead is not necessary. In reality, a lot of communication took place alongside the sprint meetings, with the intermediary meeting often turning into multiple daily meetings.

The second category of key learnings is constituted by all new knowledge gained concerning the actual implementation of a software project. In the project at hand, this means learning a, to us, completely new programming language and acquiring the skills to build a complete project with it. We learned C from scratch, used CMake to build and CTest to test our project implementations. Whilst getting into C was an anticipated learning, CMake has turned out as a major takeaway which was not expected. Learning about its numerous capabilities, like project structuring, executable definition, and compiling was very valuable to us. Especially its abilities for the conditional deployment of programs, based on the used compilers and architectures, was highly-valued.

Additionally, we sought to improve our general coding skills. We scanned the literature for a consistent programming guide to answer our questions of what makes good code. Extensive answers were found in a book written by Robert C. Martin [MC19], whose guidelines we tried to implement whenever applicable. In our eyes this significantly improved our programming, resulting in a way more consistent and clear coding-style.

5 Conclusion

The project documented in this work aimed at implementing the Advanced Encryption Standard (AES). Based on the final implementation, it can be concluded that the project outcome can be rated as successful. The performed project definition and planning was based on a previously conducted literature search, dealing with the characteristics of the AES. By outlining its fundamentals and basic structures, the project plan was developed. Retrospectively, the defined tasks, derived from previously gathered and analysed requirements, and the predicted schedule on how they would be fulfilled, laid the foundation for the overall project outcome. The final evaluation showed, that all work was accomplished in the originally set time-frame, and that the delivered implementation exceeded the essential requirements. Due to the remaining capacities, additional functionality could be added to the project outcome. Nevertheless, several less critical requirements are still open and might be implemented in the future. For example, the usage of longer than 128 bit keys or a message-exchange dialogue could be enabled next.

List of Figures

Bibliography

[Den07] Tom St. Denis. *Cryptography for Developers*. Syngress Publishing, 1st edition, 2007.

[DK02] Hans Delfs and Helmut Knebl. *Introduction to Cryptography: Principles and Applications*. Springer-Verlag, Berlin, Heidelberg, 2002.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*. Springer, 1 edition, 2002.

[Glo10] Boris Gloger. Scrum. *Informatik-Spektrum*, 33(2):195–200, 2010.

[KPP+09] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, Andy Rupp, and Manfred Schimmler. How to break des for bc 8,980. 08 2009.

[Lan00] Susan Landau. Standing the test of time: The data encryption standard. *Notices of the American Mathematical Society 47(3)*, 2000.

[MC19] Robert C. Martin and James O. Coplien. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, Upper Saddle River, NJ [etc.], 2019.

[oST01] Information Technology Laboratory (National Institute of Standards and Technology). Announcing the advanced encryption standard (aes). In *Federal information processing standards publication ; 197.*, 2001.

[Sta05] William Stallings. *Cryptography and Network Security (4th Edition)*. Prentice-Hall, Inc., USA, 2005.

[Sta10] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, USA, 5th edition, 2010.

[sym20] Symmetric cryptography, 2020. `https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.14/gtps7/s7symm.html` as of 29.01.2021.

[tG20] t2informatik GmbH. Moscow-methode, 2020. `https://t2informatik.de/wissen-kompakt/moscow-methode/`as of 11.12.2020.

Declaration

I hereby declare that, to the best of my knowledge and belief, this Seminarthesis titled “Implementing the
Advanced Encryption Standard” is my own work. I confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references.

Münster, 5.2.2021

DATE

SIGNATURE

Hamm, 05.02.2021

Consent Form

for the use of plagiarism detection software to check my thesis

Full Name: Benedikt Kluss, Hendrik Hagedorn
Student Number: 464709, 417924
Course of Study: Information Systems
Title of Thesis: Implementing the
Advanced Encryption Standard

What is plagiarism? Plagiarism is defined as submitting someone else's work or ideas as your own without a complete indication of the source. It is hereby irrelevant whether the work of others is copied word by word without acknowledgment of the source, text structures (e.g. line of argumentation or outline) are borrowed or texts are translated from a foreign language.

Use of plagiarism detection software The examination office uses plagiarism software to check each submitted bachelor and master thesis for plagiarism. For that purpose the thesis is electronically forwarded to a software service provider where the software checks for potential matches between the submitted work and work from other sources. For future comparisons with other theses, your thesis will be permanently stored in a database. Only the School of Business and Economics of the University of Münster is allowed to access your stored thesis. The student agrees that his or her thesis may be stored and reproduced only for the purpose of plagiarism assessment. The first examiner of the thesis will be advised on the outcome of the plagiarism assessment.

Sanctions Each case of plagiarism constitutes an attempt to deceive in terms of the examination regulations and will lead to the thesis being graded as "failed". This will be communicated to the examination office where your case will be documented. In the event of a serious case of deception the examinee can be generally excluded from any further examination. This can lead to the exmatriculation of the student. Even after completion of the examination procedure and graduation from university, plagiarism can result in a withdrawal of the awarded academic degree.

I confirm that I have read and understood the information in this document. I agree to the outlined procedure for plagiarism assessment and potential sanctioning.

Münster, 5.2.2021

Date

Signature

Hamm, 05.02.2021