Disclaimer

Statement for open documents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780108. Any dissemination of results here presented reflects only the consortium view.

D7.1 Preliminary specification of FENTEC prototypes

Document Identification			
Status	Final	**Due Date**	30/09/2018
Version	1.0	**Submission Date**	27/09/2018

Related WP	WP7	**Document Reference**	D7.1
Related Deliverable(s)	D7.2, D7.3, D7.5, D7.7	**Dissemination Level(*)**	PU
Lead Participant	WALLIX	**Lead Author**	Henri Binsztok (WALLIX)
Contributors	ATOS, KUD, FUAS, WALLIX, XLAB	**Reviewers**	Clément Gentilucci (FUAS) Henri Binsztok (WALLIX) Mariem Krichen (WALLIX)

Keywords:
Preliminary Technical Specification report, technical specification, use-case, digital currency, web analytics, video surveillance, IoT, crypto API

This document is issued within the frame and for the purpose of the FENTEC project. This project has received funding from the European Union's Horizon2020 under Grant Agreement No. 780108. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

(*) Dissemination level.-PU: Public, fully open, e.g. web; CO: Confidential, restricted under conditions set out in Model Grant Agreement; CI: Classified, Int = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.

Document Information

List of Contributors	
Name	Partner
Eduardo Gonzalez Real, Alberto Crespo Garcia, Álvaro García Recuero, Miguel Angel Mateo Montero	ATOS
Robert Bowers, Jean-Bernard Fischer, Yolan Romailler, Brecht Wyseur	KUD
Henri Binsztok, Norman Scaife	WALLIX
Miha Stopar	XLAB

Document History			
Version	Date	Change editors	Changes
0.1	03/06/2018	WALLIX	First content
0.2	13/07/2018	WALLIX	Updated after telco
0.3	23/08/2018	WALLIX	Updated after telco
0.4	24/08/2018	ATOS	ATOS section added
0.5	24/08/2018	WALLIX	First complete draft of WALLIX section
0.6	05/09/2018	WALLIX	Updated after telco
0.7	11/09/2018	XLAB	Added XLAB comments
0.8	12/09/2018	WALLIX	Updated after telco
0.9	14/09/2018	ATOS	First complete draft of ATOS section
0.10	17/09/2018	KUD	First complete draft of KUD section
0.11	17/09/2018	WALLIX	Submitted for internal review
0.12	21/09/2018	WALLIX	Corrections from internal review
0.13	25/09/2018	WALLIX	Final changes from use-case partners
0.14	26/09/2018	WALLIX	Final draft for submission to Project Coordinator
1.0	27/09/2018	WALLIX	Submission to EU

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable Leader	Norman Scaife (WALLIX)	27/09/2018
Technical Manager	Michel Abdalla (ENS)	27/09/2018
Quality Manager	Diego Esteban (ATOS)	27/09/2018
Project Coordinator	Francisco Gala (ATOS)	27/06/2018

Table of Contents

Document Information . . . 1
Table of Contents . . . 2
List of Figures . . . 3
List of Acronyms . . . 4
Executive Summary . . . 5
1 Introduction . . . 6
1.1 Structure and Methodology . . . 7
2 Enhanced Digital Currency Prototype Technical Specification . . . 8
2.1 Introduction . . . 8
2.2 Overview . . . 8
2.3 Design . . . 19
2.4 Operating Environment . . . 24
2.5 Evaluation Metrics . . . 25
2.6 Meeting Requirements . . . 26
2.7 Conclusions . . . 27
3 Data Collection Prototype Technical Specification . . . 28
3.1 Introduction . . . 28
3.2 Overview . . . 28
3.3 Design . . . 33
3.4 Operating Environment . . . 41
3.5 Evaluation Metrics . . . 41
3.6 Additional Requirements . . . 43
3.7 Outstanding Issues . . . 44
3.8 Meeting Requirements . . . 44
3.9 Conclusions . . . 45
4 IoT Prototype Technical Specification . . . 46
4.1 Introduction . . . 46
4.2 Overview . . . 46
4.3 Design . . . 49
4.4 Operating Environment . . . 55
4.5 Evaluation Metrics . . . 55
4.6 Additional Requirements . . . 56
4.7 Outstanding Issues . . . 57
4.8 Meeting Requirements . . . 57
4.9 Conclusions . . . 59
5 Role of the FENTEC library in the technical specifications . . . 60
6 Conclusion . . . 62
7 Next steps . . . 63
References . . . 64

List of Figures

1 Establishment of exchange . . . 11
2 Cash owner setup . . . 12
3 Merchant setup . . . 13
4 Customer setup . . . 14
5 Taler withdrawal . . . 15
6 Taler payment . . . 16
7 Audit . . . 17
8 Functional dependencies . . . 20
9 Web Analytics relationship between modules . . . 29
10 Web Analytics basic architecture schema . . . 32
11 Web Analytics software dependencies between components . . . 36
12 The functional architecture of the prototype . . . 48
13 The physical and logical architecture of the prototype . . . 51

List of Acronyms

Acronym	Description
ABE	Attribute Based Encryption
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Deployment
CLI	Command Line Interface
CP-ABE	Ciphertext Policy Attribute-Based Encryption
DDH	Decisional Diffie-Hellman
DDOS	Distributed Denial of Service
DMCFE	Decentralized Multi-Client Functional Encryption
FE	Functional Encryption
FFMPEG	Fast Forward Motion Picture Experts Group (file format)
FPGA	Field Programmable Gate Array
GDPR	General Data Protection Regulation (EU)
HD	High Definition
IND-CPA	INDistinguishability under Chosen Plaintext Attack
IV	Independent Variable
IoT	Internet of Things
KP-ABE	Key Policy Attribute-Based Encryption
LWE	Learning with Errors
NALU	Network Abstraction Layer Unit
OOP	Object Oriented Programming
PII	Personally Identifiable Information
QoS	Quality of Service
REST	REpresentational State Transfer
TDD	Test Driven Development (programming methodology)
UI	User Interface

Executive Summary

This deliverable defines an initial technical specification for each of the FENTEC use-cases. It is intended to provide a basis for the preliminary implementation work on the prototype due to commence in M9. However, due to the innovative nature of applying Functional Encryption to the selected use-cases there are many technical difficulties to be overcome to ensure that FE is both necessary and sufficient for each case. This document represents the results of the initial analyses of comparing the exact technical nature of FE with the security and performance requirements of each use-case[4].

The ATOS digital currency use-case is specified as a comprehensive set of relationships between the various actors in the scenarios. In fact, two scenarios are presented and they require different cryptographic support. Both are presented in this document. This use-case will be built on top of the GNU TALER[3] system and as such the use-case is dependent upon that platform. This choice was based upon our existing library support for that platform. Secondly, being an established platform or system, a significant advantage to the use-case is the stability and credibility of the schemes and software of the Taler community. Their specification is presumably sufficient to start working on the integration of our ABE schemes and the pilot development in later deliverables as part of the FENTEC project. However, there remains a number of questions related to the performance of the use-case and this may need to be addressed during development. A minimal set of success criteria are presented.

The WALLIX web analytics use-case has been specified using simple component relationships. The number and complexity of these components is relatively simple for this use-case but it uses recent results from FE which have not been applied in this manner before. The main point of this use-case is to validate the use of distributed key methods for web analytics situations. WALLIX will use a test-based methodology for developing and implementing the use-case in line with its existing open source application Awless which will be used as a platform for this work. The specification is complete enough to warrant implementation work which will be the next step. Success metrics have been difficult to quantify, however, given that the exact cryptographic protocol to be used has not yet been fully decided. Some basic rough guidelines for judging the success or failure of the use-case have been provided.

The KUD IoT surveillance camera use-case is specified in terms of a set of components, each of which is defined as a set of input parameters and a sequence of instructions for completing their functionality. A detailed analysis of the architecture is important here since, structurally, this use-case is quite complicated. The specification of this use-case is also more advanced than the others since detailed software interfaces are also presented. However, the construction of this use-case needs to be broken into phases since there are some questions about the nature of the cryptography required and the hardware requirements for performance reasons. In essence, this use-case is highly innovative and requires some care in its development. Again, measurements for the success of the use-case are provided although the performance targets are hard to quantify and may have to relate to current perceptions within the industry.

Work on the FENTEC library by XLAB is already well advanced, at least sufficient to support the development of all three prototypes specified in this document. The cryptographic requirements have been coordinated with the academic work being carried out as part of FENTEC and some comments are provided in Section 5 as to how the library meets the cryptographic requirements of the use-cases.

1 Introduction

The aim of this document is to provide preliminary technical specifications to allow the implementation of the use-case prototypes to proceed. However, the development of these technical specifications requires a greater degree of implementation detail and the making of concrete design decisions than was present in the original project proposal. It is intended to continue the work carried out by the requirements analysis. However, it should be noted that this document is only expected to overcome the initial obstacles to proceeding with the implementation work. Obviously, more design decisions and technical obstacles will be overcome during the development process so the final technical specifications to be published in D7.2 near the end of the project may differ in some aspects from this initial document.
The goals of this document are to specify, for each use-case:

- what the prototype is expected to achieve in functional terms,
- the environment in which the prototype is expected to work, and
- the performance goals to be achieved by the prototype.

The Requirements Analysis in D3.1[4] looked at a suitable and required environment for development of the use-cases in terms of library support, cryptographic protocols and hardware in terms of processing requirements or additional speedup hardware such as FPGAs. This deliverable specifies in more detail the functionality of the use-cases in terms of a decomposition into subcomponents and the relationship between them, although each use-case partner is free to describe this decomposition using their own methods. The context for the use-cases has been set by the requirements analysis but here we state explicitly what operating systems, languages and other support we need to proceed with implementation. Performance metrics have proven hard to quantify and will vary depending upon the use-case. Where possible we have stated industry standard metrics for some cases.
This document is the next step in the project following the requirements analysis. Here we concentrate on how we intend to meet the requirements indicated by that work and proceed to develop specific technical details for the implementation of each of the use-cases.
This will involve more concrete decisions concerning the exact cryptographic methods used in the implementations leading to more precise definitions of the actual functionality we can achieve with each use-case. It also concerns the support required to realise them while setting performance goals for the prototypes to be produced as part of the FENTEC project.
The three use-cases are essentially independent of one another in terms of their functionality. However, they all use FE and the FENTEC library for their cryptographic components. Thus we present the technical specifications separately for each use-case but bear in mind the common thread between them.
Since we are only required in this document to specify the initial work for each use-case our specifications may not necessarily be complete. The major areas of concern are the actual cryptographic schema to be deployed and the manner in which these schema are to provide the expected functionality and performance. Some detail will be required as to the operating environment; for example which library dependencies will be used and the computational environment such as placement of processing resources. Peripheral issues such as how this technological core is to be presented to the users of the system, for example user interfaces need not be specified in detail in this document. These will be elaborated upon in D7.2.
This deliverable leads into the reports on each use-case (all due in M22):

- D7.3 First version of the truly anonymous data collection prototype
- D7.5 First version of the privacy enhanced digital currency prototype
- D7.7 First version of the IoT key distribution prototype

1.1 Structure and Methodology

Since the three use-cases are from widely differing application domains and are essentially independent of one another (inasmuch as the functionality of each use-case is not related to the other cases) the three technical specifications are presented in individual chapters. Note that this is not to say that there is no interplay or communication between each partner since it is expected that each use-case will be based on the same library (albeit differing subsets) developed as part of WP6. This document is structured in 6 major chapters besides this introduction:

section 2 gives the initial technical specification for the ATOS enhanced digital currency prototype.

section 3 gives the initial technical specification for the WALLIX data collection prototype.

section 4 gives the initial technical specification for the KUD IoT prototype.

section 5 gives comments from XLAB on how the use-cases conform to the FE library implemented in WP6.

section 6 concludes with a brief statement about conformance with project goals.

section 7 gives some brief indications for the next steps in the project.

The methodologies used to construct the prototypes may also have to be separate for each use-case. Each individual use-case partner, being from very different application domains and commercial backgrounds will most likely already have well-tested and trusted methods for generating code and systems, both physical such as computational platforms and software such as libraries and user interfaces. It is probably unreasonable, therefore, to expect a single uniform methodology to be applicable for FENTEC. Each partner then, is free to use whatever methods apply in their case with the small proviso that any software methodology applied by a partner should be able to incorporate code from the common FENTEC library.

2 Enhanced Digital Currency Prototype Technical Specification

2.1 Introduction

As described in Deliverable D3.1[4], over a payment platform designed following the Chaum Principles about untraceable payments[1], we propose the use of FE to build a payment system with dedicated digital coins, which contains information about how, when and where they can be used, and if they have associated benefits or discounts. It is also desirable to add the required mechanism to perform audits to participants without revealing more data than is strictly required to fulfil their fiscal duties and rights, according to the data minimisation principle of GDPR[2].
The FENTEC toolkit will be added as an extra layer on top of an existing payment solution to leverage the functionality provided by the platform such as availability and integrity. This will require the implementation of interfaces between the payment platform and the FENTEC toolkit and new user interfaces needed to make use of the new functionality.
The platform chosen for the implementation is depicted in the architecture on the TALER[3] website, a cryptographic e-cash exchange system which aims to enable anonymous and unlinkable customers' payments (cash-like anonymity) while preserving the merchant transactions easily identified and thus taxable. To this end, TALER implements a mix of several privacy and data protection measures (such as avoiding submission of personal information during the payment process or using blind signatures to create the digital coins) and accurate income information to facilitate taxation while avoiding money laundering. One of the benefits of this platform is that it is open-source with a big community and good support for developers. Its main concept is the privacy of customers and provides interfaces to perform audits to the platform itself and to merchants. The introduction of Functional Encryption (FE) brings a powerful expressiveness enabling the use of different types of digital currency as privacy-preserving money tokens for different purposes (eg. buy specific goods, specific merchants or loyalty programs) and to add a new mechanism to audit customer invoices.

2.2 Overview

The pilot is divided into two scenarios; special purpose coins and customer audit, in which participants collaborate in different ways. For both scenarios the aim of FE is to let participants have access or learn part of the data depending on their attributes and the conditions they must fulfil. Therefore, the most adequate approach that fits these general requirements is Attribute Based Encryption.

Special Purpose Coins Scenario This scenario consists of adding to coins a set of attributes to describe their characteristics. By means of FE, the merchant can only accept these digital coins as payment if the conditions associated with the merchant are fulfilled by these coins. Moving this description to the Functional Encryption domain, conditions are policies encoded in the Secret Key that merchant uses to access the coins' data, and coins are cyphered with respect to their attributes. This description corresponds to a Key-Policy ABE scheme(KP-ABE). From now on, keys related to this scenario will be denoted by a KP sub-index (Master key: MK_{KP}, Public key: PP_{KP}, Secret key: SK_{KP}).

In this scenario the exchange uses PP_{KP} to cypher digital coins, customers will use SK_{KP} to access the data required to charge coins into his wallet and merchant will use SK_{KP} to learn the data

[1] http://sceweb.sce.uhcl.edu/yang/teaching/csci5234WebSecurityFall2011/Chaum-blind-signatures.PDF
[2] https://gdpr-info.eu/
[3] https://taler.net/en/index.html

required to accept such coins as payment. Also, as can be deduced from the description of the use-case, the aim is to provide support for different types of coins, therefore the system must provide a set of PP_{KP} and the corresponding keys SK_{KP} for each coin type.

Customer Audit Scenario This scenario consists of using FE to ensure that data from customer's invoices can be only accessed by those auditors who fulfil a set of conditions. It should be added that there can be several auditors, each of them interested in different invoice data. In this case, conditions are policies encoded into the cyphered invoices while the secret key of an auditor includes the attributes of the auditor. This description matches the characteristics of a Cyphertext-policy ABE scheme (CP-ABE). From now on, keys related to this scheme will be denoted by a CP sub-index (Master key: MK_{CP}, Public key: PP_{CP}, Secret key: SK_{CP}).

In this scenario a customer uses their PP_{CP} to cypher invoices and an auditors will use their SK_{CP} to learn the data they needs.

2.2.1 Objectives

Functionality The pilot of the digital currency platform focuses on the creation of a payment system with the aim of preserving customer privacy while keeping the auditability of the system. In recent years there have been many approaches. Most of them provide solutions for a general purpose payment method. Our aim is to build a payment system with digital coins of general purpose but with specialized usage. The system is based, in this first description, on ABE FE schemes, although performance will be achieved and the number of required keys determined by the introduction of other schemes during the life of the project.

Architecture The architecture of the pilot will maintain the Taler platform architecture, our aim is to integrate FE libraries as an extra layer and develop the interfaces required at a programmatic level. In the case of the Auditor it will be needed to develop a new application to play its role, that entails the development of a new user interface. The pilot will be deployed in a cloud environment following Taler as a guide. Regarding the wallet application, although it can be implemented in many ways; mobile app, web app, etc. For the first iteration of the pilot we will develop a web app as it can be used by means of a web explorer in a PC or in a mobile, with the FE library running on the server side. For a second iteration we will consider running this library in a mobile to test its performance on constrained devices.

Environment The TALER payment platform chosen to implement the pilot and its main modules are written in C++ to run in Linux environments, which, while not a hard constraint, constitutes the main point of consideration for choosing the programming language and environment to develop the FENTEC tool kit and interfaces.

Performance The overall pilot with FE covers some steps in a purchase procedure through the payment platform, since money is converted to e-cash until the audition of purchases. Some of the FE functions can run concurrently in a system while others must run sequentially as a customer can only perform one payment at a time. The overall process will be explained later. Regarding performance of this pilot, it is constrained to the fact that most of the FE operations are carried out on the fly. Adding FE to the platform does not allow a situation in which payments become rejected by time-out. It will be studied for each case when performance can be kept within the thresholds of the platform. The system will then either be kept as it is or the performance may be improved by improving the FE schemes used or more processing capacity could be added.

2.2.2 Participants in the Use Case

This section describes participants of the use-case, which also corresponds with functional components, some of them are provided by Taler and will be modified according to the needs of the use-case. There are also new functional components such as issuer of Trust which are added to implement specific functions for the use-case. For the original Taler component, a more detailed description can be found at the Taler site[4].

Customers' wallet this component is an e-wallet application used to perform payments in digital cash. In the present use-case, the payment platform has an Exchange service which receives funds from banks and exchange it to its own coins or digital cash. This digital cash is then transferred to the customer's wallet application. Although there is only one interface to transfer funds to the Exchange there are two conceptual ways; the first one is where the owner of the wallet, the customer, performs the transfer from her/his bank account, the second where a third party performs the transfer, for instance the company where the customer works (eg. a ticket restaurant) or the market where the customer does weekly shopping (loyalty programs).

Exchange this component is the core of the payment platform. It issues digital coins in exchange for currency which is charged to the customers' wallets, keeping track of created and used digital cash to avoid double spending. In order to keep privacy of customers it implements a blind-signature pattern to deliver digital cash. This enable the possibility of correlate where digital cash is spent without revealing the identity of the customer. Regarding the consistency of the payment system it provides interfaces to perform audits over its operation as exchange service, and also a hosting service for customer's invoices.

Merchant this is an electronic commerce web application where customers can perform payments with the e-wallet. It must check that the digital cash are valid for the purchase and make a request to the Exchange to check that there is no double spending. Once the payment is carried out it can request the conversion of received digital coins into currency and transfer to the merchant's bank account.

Auditor This component offers an interface to retrieve information required to audit the operation of the platform as exchange money service. In a traditional concept it provides interfaces to audit the payment platform itself and the merchant. We propose to add a new interface to retrieve invoices of purchases done by customers. Invoices are cyphered with FE techniques in order to set what to reveal and to whom it should be revealed. Beyond legal ways when a judge can request information from the platform, this new interface is intended to let customers reveal some data of their purchases. Possible applications of this interface are to fulfil tax obligations, request help or funding, etc.

Bank this component is added to show the connection between the payment platform and the traditional monetary system. Into the pilot it transfers and receives money to/from the Exchange.

Cash Owner This role, new in the use-case, represents a third party who can instruct the exchange to issue digital coins to be used under specific conditions. It covers a wide range of possibilities, from local governments to issue their own money to brands issuing money into loyalty programs. Conditions can be associated with due dates, localization of merchants, type of product or Merchant, etc.

Issuer of Trust Although the scenarios focus on different aspects of a commercial transaction and use different encryption schemes; KP-ABE for the scenario of special purpose coins and CP-ABE for the scenario of customer audits, they need an entity as a source of trust, who will be the authority

[4] https://docs.taler.net/

in charge of generating and delivering keys to each of the parties. Following the Taler concept, the payment system does not track how digital coins are used; at which merchant the customer buys and which products are bought. Hence, it cannot play this role and makes it necessary to introduce a new entity who performs the duties of an issuer of Trust, the Master Authority. In general Governments are the entities that fix and apply regulation to the creation and use of currency. Therefore, it seems logical to adopt the government as an issuer of trust, or a third party certified by the government, for instance its Mint. The mission of this component is to generate all keys related to FE and their delivery to each party in a secure manner.

2.2.3 Functional Encryption flows

In both scenarios, before any of the parties start working it is necessary to legally establish a commercial relationship by signing contracts which covers the terms of each relationship, with the exchange server as nexus at technical level. The act of signing a contract allows to launch the corresponding processes of the FE schemes involved; set-up and delivery of keys. The set of attributes and available policies for FE are covered by the contract between parties, therefore, signing of contracts is common to both scenarios as they fulfil the requirements to carry out each scenario. This establishment process is reflected in Figures 1 to 4 below.
Note: in the following flow diagrams, all references to keys are done for FE scheme keys. Other keys used by Taler platform are out of the scope and their use could change due to the integration of FE schemes.

2.2.3.1 Establishment of Exchange This consists of the signing of a contract between the Issuer of trust and the Exchange to cover all the legal constraints related to the Exchange operation and its services. The act of signing this contract entails the setup process of the KP-ABE scheme, in this case the generation of the Master Key MK_{KP} and the Public Key PP_{KP} that the Exchange will use to issue general purpose Digital coins. In this case the attributes and policies associated with the keys should allow the use of digital coins, encrypted with them, without restrictions.

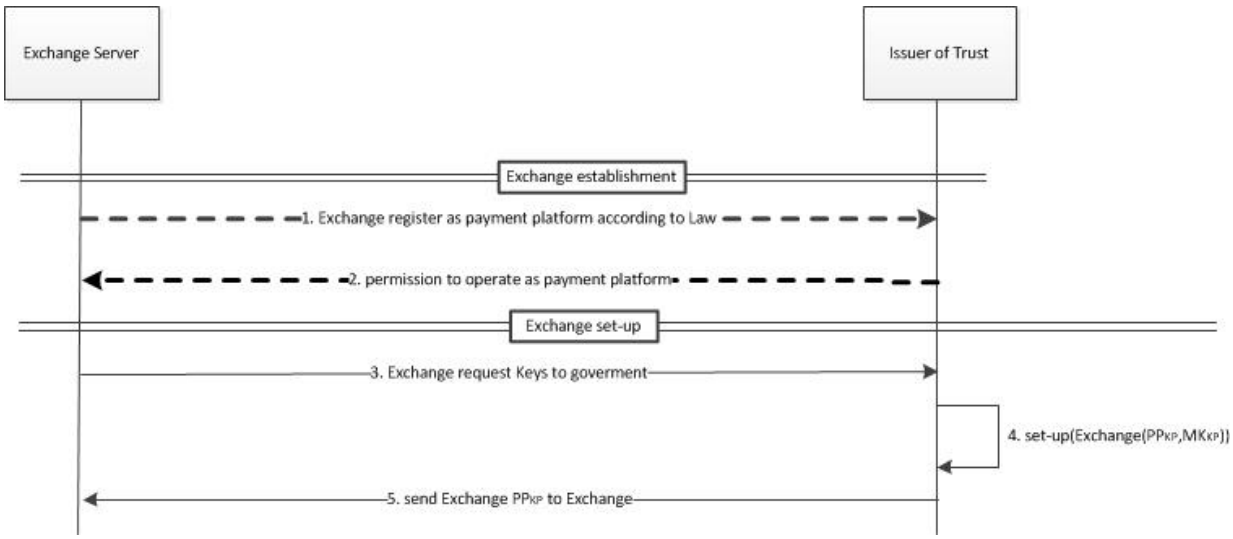

Figure 1: Establishment of exchange

1. Exchange signs a contract with the Issuer of Trust to operate as a Payment system.

2. Permission granted to operate as Exchange.

3. Exchange requests Keys from the Issuer of Trust to encrypt digital coins (according to the attributes and policies in the contract).
4. Issuer of Trust runs the set-up process to generate MK_{KP} and Public Key PP_{KP} for the Exchange.
5. Issuer of Trust delivers PP_{KP} to the Exchange in a secure manner.

2.2.3.2 Exchange-Cash Owner Relationship Establishment The cash owner is the entity which instructs the exchange to issue digital coins that can be used under specific conditions (policy). Additionally, it can also order the transfer of them to the wallet of a customer. The act of signing this contract entails the corresponding generation of a key PP_{KP} for the new type of coin and a SK_{KP} for each of the participants who are going to use these digital coins; customer to charge them into the wallet, and merchants to accept them as payment.

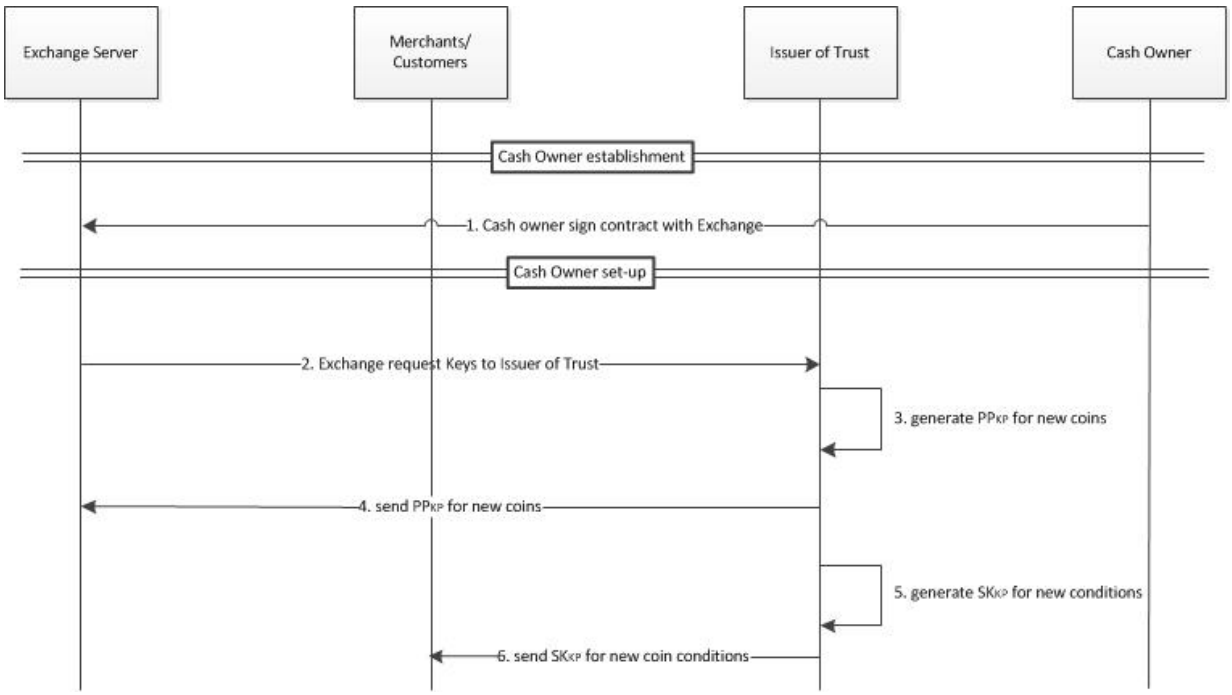

Figure 2: Cash owner setup

1. Cash Owner signs a contract with the exchange to create a special coin, this contract covers the conditions of use of this coin which result in FE policies.
2. Exchange requests PP_{KP} to cypher the new type of coin.
3. Issuer of Trust generates PP_{KP}.
4. Issuer of Trust delivers PP_{KP} key.
5. Optionally, the Issuer of Trust can automatically generate SK_{KP} for customers and merchants but this depends on the business model; if customers and merchants who can use these coins are known *a priori* or not.

2.2.3.3 Exchange-Merchant Relationship Establishment A contract between Merchant and the Exchange is prior to the Merchant being able to accept digital coins issued by the Exchange. Once the contract is signed the Merchant can request the keys to the Issuer of Trust Keys of the KP-ABE scheme (SK_{KP}) allowing Merchant to accept digital coins, it will receive a SK_{KP} for each type of coin it can use.

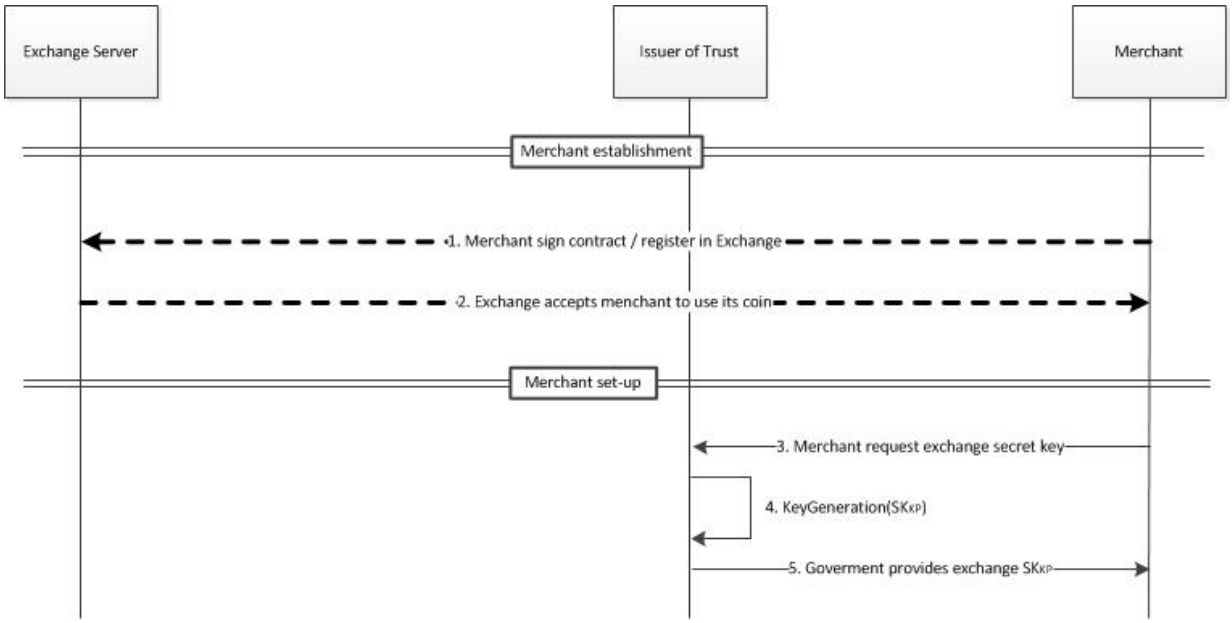

Figure 3: Merchant setup

1. Merchant signs contract with the Exchange to accept their digital coins as payment.
2. OK.
3. Merchant request from the Issuer of Trust the key to learn the data from each digital coin.
4. Issuer of Trust generate the required key.
5. Issuer of Trust delivers keys SK_{KP} to the merchant in a secure manner.

Note: As the Exchange can issue several types of special digital coin when it works with Cash Owners, the contract between Exchange and Merchant has to specify which kind of coins it can use and will receive a key SK_{KP} for each of the types of coin.

2.2.3.4 exchange-customer relationship Establishment A contract between customer and exchange is necessary before the customer can operate with digital coins issued by the exchange in his/her wallet. Once the contract is signed, the customer can request as many keys SK_{KP} needed to load digital coins into his wallet as it validated in the contract. This act also entails the generation of keys PP_{CP} the customer will use to cypher invoices according to the different audits.

1. Customer installs a wallet application or creates an account in a wallet cloud service.
2. Customer signs and accepts legal conditions to use digital coins issued by the Exchange.
3. Customer is granted.

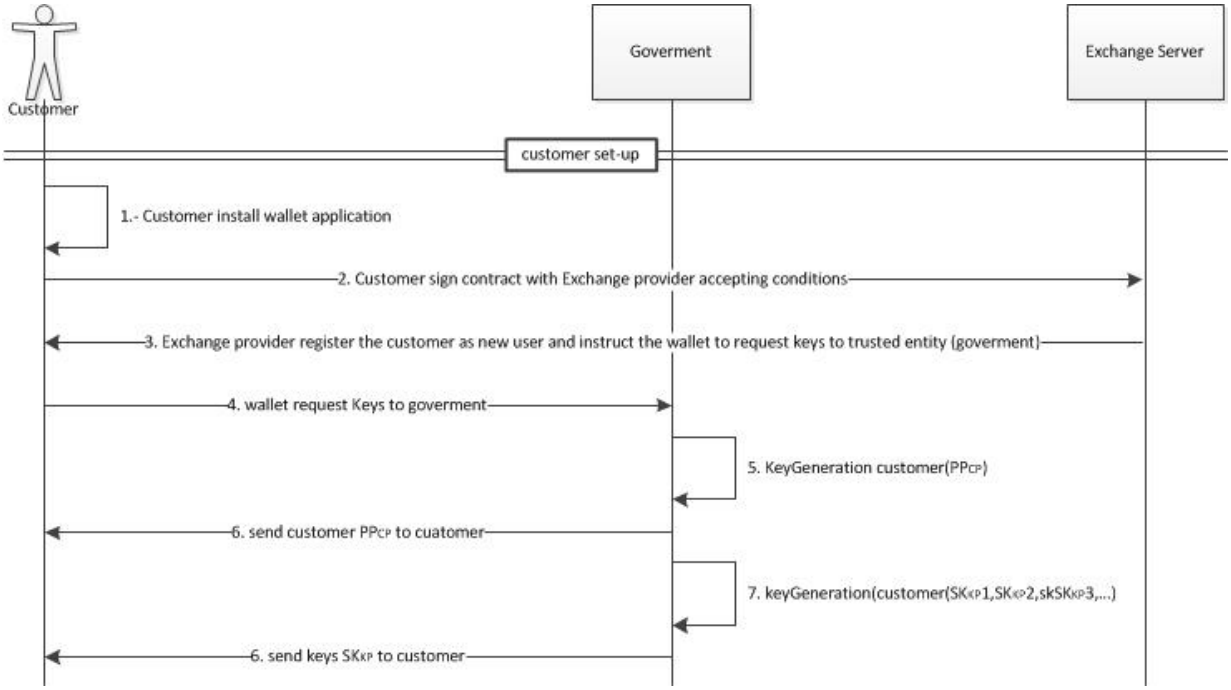

Figure 4: Customer setup

4. Customer's wallet requests keys from the Issuer of Trust.
5. Issuer of Trust generates PP_{CP}.
6. Issuer of Trust delivers PP_{CP}. This key is used to encrypt invoices.
7. Issuer of Trust generates SK_{KP} per each type of coin the customer can use.
8. Issuer of Trust delivers keys SK_{KP} to customer.

Once each participant is established the Customer can perform payments through the platform and request audits on invoices, the following three figures (Figures 5, 6 and 7) correspond to each of the steps involved in the use-case. From the point of view of the scenarios, the Special Purpose Coins scenario is depicted in the flows of Reload and Payment while the scenario of Audit is depicted in last step of payment flow and the Audit flow.

2.2.3.5 Reload customer wallet This flow is based on the Taler Withdrawal flow In this case we consider that there are two different players who can order the reload; it can be the customer owner of the wallet, in this case the digital coins will not have any condition about their use, or it can be a third party such as the company where the customer works or a local government or a market. In the second case the digital coins will be constrained to a set of conditions about their use (policy encoded in the secret key).

1. Customer or third party grants access to the bank portal.
2. Customer or third party requests that the Bank transfers money to the Exchange.
3. Bank transfers the money to the Exchange.

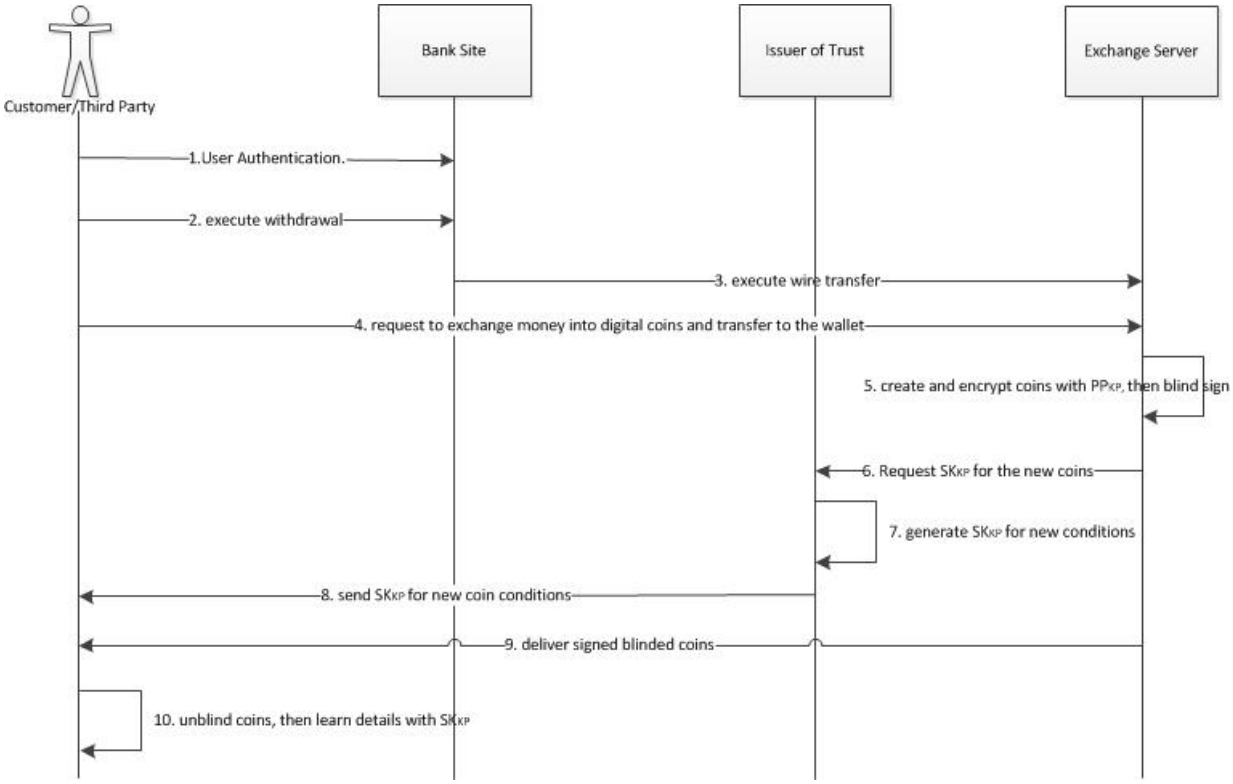

Figure 5: Taler withdrawal

4. Customer or third party makes a request to the Exchange to transfer the money to the wallet, this entails issuing digital coins and transferring to the wallet. In the case of the third party, it will indicate the conditions of use of these coins. This translates to choosing the policy in the SK_{KP} to decrypt them.

5. The Exchange creates the digital coins according to the denomination requested and encrypts them with PP_{KP} and blind signs them.

6-8. These steps are optional, they will be executed when the reload is instructed by a third party and the wallet does not have the appropriate SK_{KP}. The exchange requests the SK_{KP} for the customer to the Issuer of Trust, it generates the SK_{KP} and delivers it to the customer in a secure manner.

9. The Exchange sends the requested digital coins to the wallet which are then blind signed.

10. The wallet un-blinds the digital coins and uses SK_{KP} to get the required data from the digital coin.

2.2.3.6 Payment The following step is the Customer performing a purchase at a merchant site. Again, this flow is based on the Taler payment flow. This flow is based on the Taler Payment flow.

1. The Customer chooses the goods he wants to buy.

2. The Merchant site sends a proposal of purchase and payment.

3. The wallet sends the payment to the merchant's site which consists of cyphered digital coins.

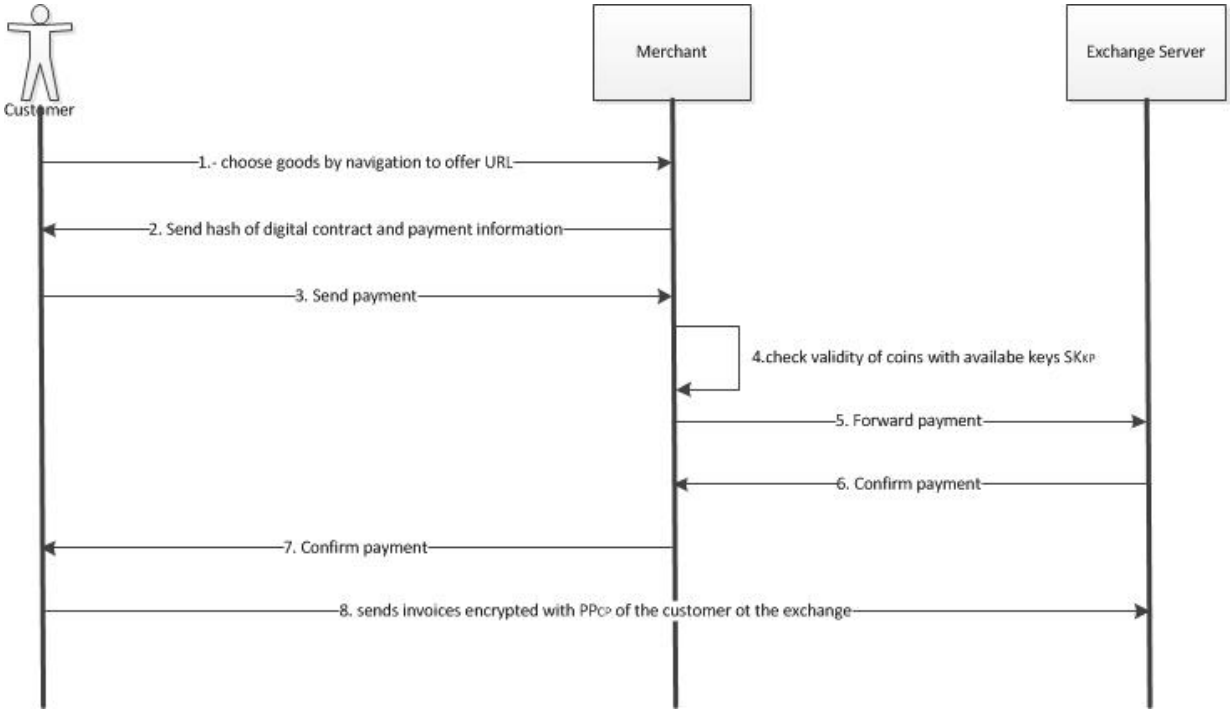

Figure 6: Taler payment

4. The merchant checks if the attributes of the received coins fulfil the policy encoded in any of the keys SK_{KP} he has, and therefore if the digital coins provided can be used for the transaction. This step is blocking and could result in rejecting the payment if it does not find a valid key.

5. If the Merchant has a valid SK_{KP} and therefore can learn the denomination of the coins then it requests that the Exchange checks for double spending and new digital coins in case it must give change.

6. The exchange answers the request by indicating there is no double expending and providing a signed refund permission if necessary.

7. Merchant site confirms with the wallet that the payment is valid and provides the signed refund permission to the customer's wallet. Later, the wallet can use this permission to obtain new coins as refund.

8. Once the payment has been carried out the customer encrypts the invoice with its public key PP_{CP} and sends it to the exchange to be hosted. This is the first step of the Audit Scenario.

2.2.3.7 Audit This flow, together with the last step of the payment flow constitutes the second scenario, Customer Audit, in which the auditor uses FE to learn the data they need to perform the audit.

1. The audit process begins with a request from the customer for instance to request a refund for a product that has been purchased with other ones and are present in the same invoice.

2. Auditor requests from the Exchange a copy of the encrypted invoices of the customer.

3. If the auditor is granted access to the Exchange, it will receive the encrypted invoices.

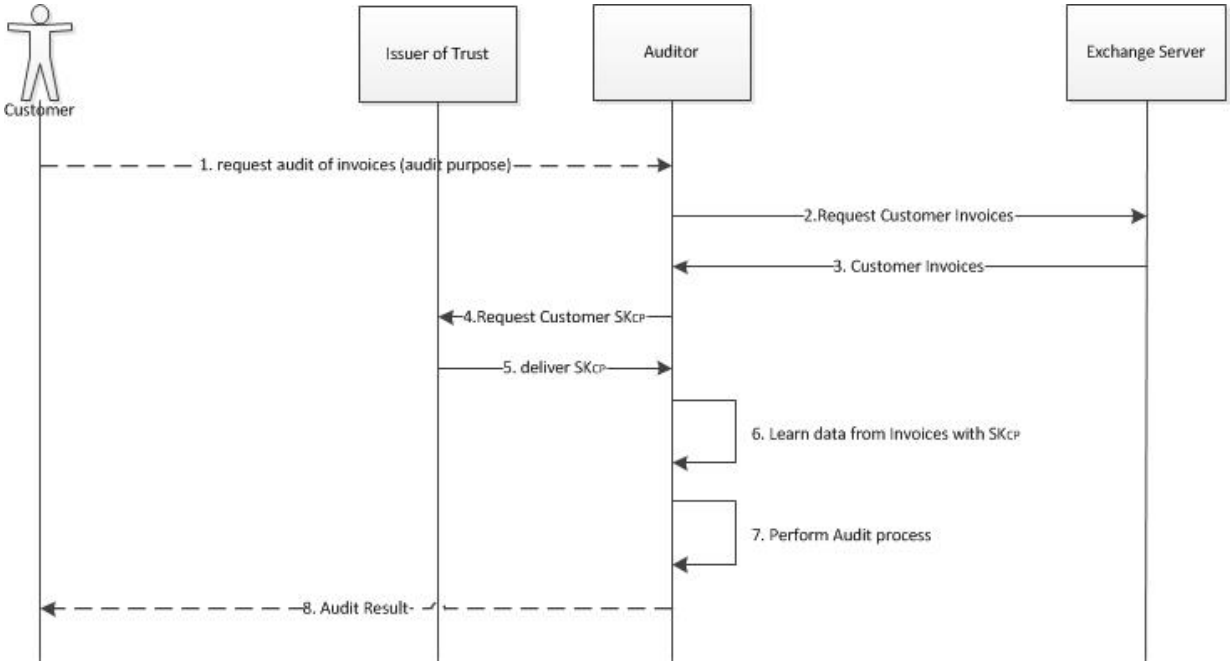

Figure 7: Audit

4. Auditor requests the SK_{CP} of the customer from the Issuer of Trust.
5. The Issuer of Trust delivers the SK_{CP} to the auditor if it has permission to access the data specified in the related policy.
6. Auditor runs the decryption process and will learn only the data it needs if it fulfils the policy e.g. the Auditor will not be able to see the exact time and location of purchases but rather the amount of spending in goods subject to tax relief.
7. Once the auditor has the data it performs the audit (calculates tax relief).
8. Finally, the auditor sends the result of the audit to the citizen.

2.2.3.8 Examples of datasets and policies

2.2.3.8.1 Example of Merchant policy for special purpose scenario In this case the Merchant will learn the data needed from the coins if the attributes associated with the cyphered text (digital coins) satisfy the policy encoded in SK_{KP}. This policy depends on the use of the coin, for instance in the case of coins issued into a loyalty program the policy can consist of checking that the due date has not expired and that the type of product and name of the loyalty program corresponds to the market one. Consider the following enumeration as the list of new attributes of a digital coin:

1. Owner: reference or id of the cash owner who instructs the exchange to create the coins.
2. Localization: geographical area where the coins can be used.
3. Merchant ref: this attribute indicates the characteristics of the merchant where coins can be spent. It is the name or reference of the Merchant brand where the coins can be spent, it can also contain a specific type of merchant in the case of coins for products (e.g., pharmacy).

4. Type of product: type of product that can be bought with the coins (e.g., baby care).
5. Loyalty program: name of a loyalty program to which coins are associated.
6. Due date: date after which the coin is no longer valid.
7. Coin ref: Name to identify the type of special coin.

while the Taler existing attributes for the coins are:

- Denomination
- Hash code

In this case the Merchant will learn data needed of coins if the attributes associated with the cyphered text (digital coins) satisfy the policy encoded in SK_{KP}. This policy represents the policy that coin has to fulfil to be used, for instance in the case of coins issued into a loyalty program the policy can be:

```
if due date has not expired, loyalty program corresponds to merchant...,
    the merchant will learn the following data from coins:
        Denomination (value of the coin), hash code...
```

2.2.3.8.2 Example of Customer policy In this case the Auditor will learn the data needed if his attributes, added to SK_{CP}, satisfy the policy encoded in the cypher-text, this is the invoice encrypted. Considering the following as an example of invoice:

- Invoice id
- Date
- Merchant ref
  - -Id
  - -fiscal ref
- Total amount
- Total tax
- Product List
  - -Product Name
  - -type
  - -Reference
  - -Quantity
  - -Price
  - -Tax rate (%)
  - -Total
  - -AuditorRef
  - -AuditPurpose

An example of policy can be:

```
if AuditorRef is ok, purpose of the audit is...,
    the auditor will learn the following data from the invoice:
      Date, MerchantRef and those products of type ....
```

2.3 Design

2.3.1 Methodology

The technical work will follow an Iterative and Incremental development approach, an alternative methodology. This decision comes from the particularities of and complexity of introducing FE into an existing payment platform that requires modifying different programmatic modules and interfaces. This methodology allows the division of the work into subtasks or functions of the whole functionality. The process consists of repeating a set of steps recursively for each of the subtasks: analysis, design, implementation, and testing. Once a subtask is finalized the result is integrated into the platform, which is built by adding functions progressively.
Source: Iterative and incremental development from Wikipedia[5].
For building our use-case, ATOS will follow almost the whole development process but the transition phase and any other tasks (within the construction phase) related to delivery will be omitted as we do not plan to have a production environment for the pilot.

2.3.2 Software Development Tools

ATOS will use GNU Taler as the base system for our developments and for the integration of FENTEC toolkit.
Programming languages involved in the current system:

1. C++.
2. Python.
3. Web languages (HTML, CSS and JavaScript).

The programming language used for the auditor's module will depend on the components that it will integrate with (we will use mainly C and/or Python) as we start from an already existing system. Also, we will integrate the version of the FENTEC toolkit that is written in C code.
Development and configuration tools:

1. Bash scripting.
2. Make.
3. IDEs and enhanced text editors.

As GNU Taler is an already existing system, it includes several testing tools and automated tests that are currently being used at deployment stage. For our use-case, ATOS will include new tests covering our modification and addition developments but keeping the already existing formats for the whole platform operation, while specific tests for FE functions will be added in a separate way within the involved modules.

2.3.3 Programming Standards

The ATOS use-case will be developed following an Object-oriented programming standard (OOP) as the core modules of GNU Taler are mainly developed following this paradigm. We aim to also include notions of Continuous Integration and Continuous Deployment (CI/CD) to facilitate and reduce integration efforts, but the successful application of this approach depends on the problematics related to the current implementation and configuration of GNU Taler.

[5] https://en.wikipedia.org/wiki/Iterative_and_incremental_development

2.3.4 Decomposition into Components

The following image shows participants required for the pilot and their dependencies, some of them will be simulated as they are not strictly needed for the aim of the pilot, but their role help to the understanding of the operation.

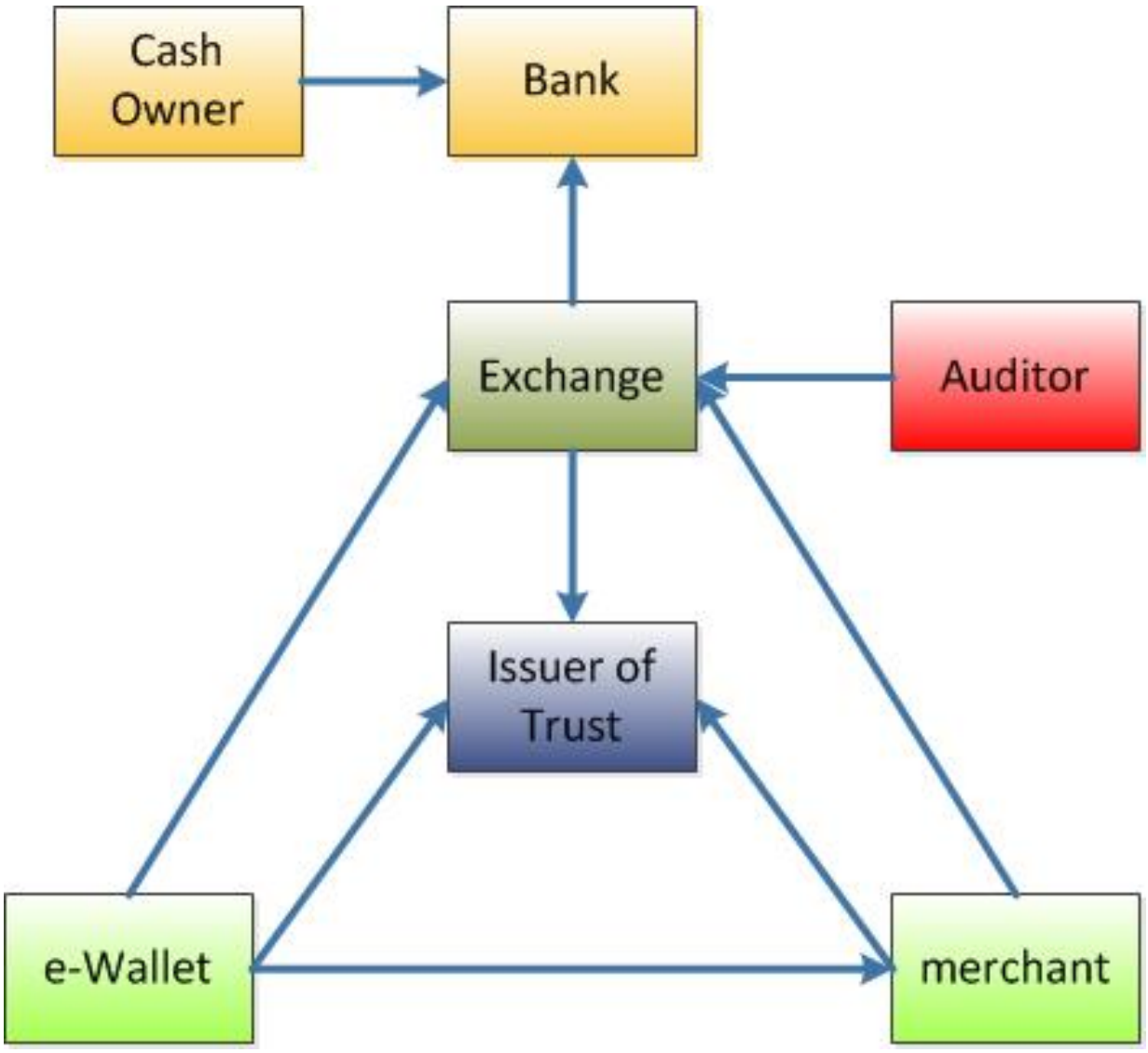

Figure 8: Functional dependencies

2.3.5 Detailed Description of each Component

1. **Bank module**

   - **Type** Java application.
   - **Purpose** False bank component only for demonstration. It will simulate transfer funds from/to bank accounts of the customer, merchants and third entities through the exchange system depending on the triggered operation.

- **Functionality** This component will allow the customer or a third entity to transfer real funds from its bank account to a e-wallet as digital cash. Once funds have been transferred to the Exchange it can be withdrawn to the e-wallet, allowing customer to use it in e-commerce. Also, once a purchase is done, the Exchange will transfer real funds to the merchant's bank account once the merchant performs the payment claim to the exchange server.
- **Dependencies** The bank module does not have dependencies.
- **Interfaces** It will implement a basic implementation required for the operation of the Exchange.
- **Computational requirements** -
- **Data requirements** -

2. **Exchange server**

   - **Type** Web server
   - **Purpose** Intermediary in charge of exchanging real currency into digital cash and vice versa. It issues the digital cash and cypher it according to a set of constraint that defines it use. It also works to check the validity of the digital cash received by merchants as payment.
   - **Functionality** This component is in charge of digital coins management between banks and e-wallets / merchants working as an exchange money service and monitoring validity of digital cash it issues.
   - **Dependencies** It directly depends on the Issuer of Trust role as it needs key generated by the issuer of trust to issue digital cash. This module also has dependencies with banks, as it must implement required interfaces to receive and perform funds transfers.
   - **Interfaces** The main interfaces will be based on a REST architecture and used for the communication with all other components. It also communicates with the Issuer of Trust to receive keys needed for issuing digital cash.
   - **Computational requirements** Although this component could receive a large amount of requests in a production environment, we plan a very low load for the prototype demonstration so, the exchange server will not need very high computation capabilities beyond a normal modern computer.
   - **Data requirements** Enough storage for all the information handled by this component that needs to be persisted like transactions details, coins specifications, etc. Furthermore, the exchange server raises several security and privacy issues that will need to be faced (authentication, crypto features, secure and privacy preserving storage of sensitive data, etc.).

3. **Customer e-wallet**

   - **Type** Browser add-on or web app.
   - **Purpose** Allows a customer withdraw digital cash from Exchange service (also reverse operation in case of refund) and buying items in the merchant store.
   - **Functionality** This module, firstly withdraw digital cash from Exchange, in this operation it will be able to accept the received cash if it fulfils a set of constraints cyphered in the cash document. Once the customer has digital cash within the e-wallet, he will be able to perform a purchase in any supported merchant's store. The e-wallet also supports refunds (undo deposit of the given coin, restoring its value) and emergency cash-back in case of abnormal situation like compromised signing keys or exchange server about to go out of business.

- **Dependencies** The e-wallet has dependencies with exchange server for management of digital cash, Issuer of Trust as it needs keys to be able to accept the digital cash withdrawn and with merchants e-store to perform purchases.
- **Interfaces** This component will use interfaces provided by the exchange server and merchant's store. It will also use interface with the Issuer of trust.
- **Computational requirements** The e-wallet will not need high computational capabilities as it will not perform too much operations or face any special performance issues.
- **Data requirements** Enough storage and protection measures for all data related to user account, coins, crypto keys, expenses information, etc. Also, all security and privacy aspects related to data processing in a browser (e.g. local storage, cookies, etc.).

4. **Merchant store**

- **Type** Web application
- **Purpose** Electronic commerce in which the customer will purchase products, these purchases will be paid using digital coins that later will be exchanged for real funds transferred into the merchant account.
- **Functionality** The merchant store will be a basic e-commerce application in which the customers will purchase products using digital coins from his electronic wallet. In order to be able to process the digital cash received as payment it will have to fulfil the set of constraints affecting to the merchant and the operation. Then the merchant will perform a payment claim to the exchange server to receive a wire transfer with real funds (equivalent to the payed amount of digital cash).
- **Dependencies** This component has dependencies with exchange server to check validity of digital cash and transfer it to bank, and with Issuer of Trust.
- **Interfaces** A basic graphical user interface to perform product purchase and a REST interface to perform payment. This component will also use the interface provided by exchange server as to need keys to process digital cash received as payment.
- **Computational requirements** Although this component could receive a large amount of requests in a production environment, the merchant store will not need high computational capabilities as we do not foresee heavy traffic or any other performance issue for the prototype demonstration.
- **Data requirements** This component will produce and store data related to user accounts, customer purchases, transactions, etc. so it will need enough storage capabilities and protection/privacy preserving measures for the processing of such sensitive information. Nevertheless, we expect a very basic and low amount of data for a simple demonstration prototype.

5. **Auditor module**

- **Type** Web application
- **Purpose** This module will retrieve customer's invoices from Exchange and perform will analyze them according to the restrictions set on its cyphering process.
- **Functionality** The auditors will use this component to obtain information about purchases, this information can be object of different aims: taxes, loyalty programs, etc.
- **Dependencies** The auditor module depends on the exchange server to obtain information about purchases from customer's invoices. It also has dependences with the Issuer of Trust.

- **Interfaces** Basic graphical user interface that will allow the auditors to obtain and verify information about customer purchases. This component will also have a REST API for the internal communication with the exchange server.
- **Computational requirements** The auditor module will not need high computational capabilities as it will not have heavy traffic or any special performance issues for the prototype demonstration.
- **Data requirements** No special needs beyond enough storage capabilities and effective protection measures for producing and storing sensitive data related to user accounts, information about auditors and customers, management of FE keys, etc. Nevertheless, we expect a very basic and low amount of data for a simple demonstration prototype.

6. **Issuer of trust**

- **Type** Web application
- **Purpose** it plays the role of the FE Master authority. It is in charge of generating all keys related to FE and their delivery to each party in a secure manner.
- **Functionality** This module manages the creation and secure delivering of keys needed by Exchange, e-wallet, merchant and auditor to cypher and learn documents of digital cash and customer's invoices.
- **Dependencies** This component has no dependences.
- **Interfaces** Secure interface for FE keys delivering.
- **Computational requirements** No special needs beyond performance issues that could raise from the FE crypto operations.
- **Data requirements** This component will mainly store information related to auditors and customers as well as the FE keys. For this purpose, enough storage and strong protection measures will be needed.

7. **Cash owner**

- **Type** none
- **Purpose** This role plays as a legal subject how want to create a new digital cash type.
- **Functionality** It defines the constraints affecting the use of the digital cash contract with exchange. It can also request the transfer of funds from a bank to the exchange.
- **Dependencies** This module has dependencies with bank to order transfers to Exchange. There is no technical dependency with Exchange because the operation of ordering to issue a new kink of digital cash need to be done through the sign of a new commercial contract.
- **Interfaces** -
- **Computational requirements** -
- **Data requirements** -

2.3.6 External Interfaces

This section specifies the external interfaces for the prototype:

User interfaces The user interfaces for the digital currency use-case are composed of three items:

i) a web application for the merchant store.

ii) web application or navigator extension for the e-wallet.

iii) a simple dashboard for the auditors to perform the auditing analysis, the latter will be exclusively oriented to testing and demo purposes.

Hardware interfaces There will be no hardware interfaces for the use-case prototype.

Software interfaces Referring machine-to-machine communication between the several components that are involved in the system, these interfaces will mainly use a REST architecture. We also include software interfaces needed for the integration of internal libraries as FENTEC toolkit or any kind of database or storage module.

2.3.7 Documentation

This section specifies the technical documentation related to development, deployment and usage. As GNU Taler is the base system for the ATOS digital currency prototype and it is an already existing system, we will start from the already included documentation adding explanations related to our addition/modification developments.

User guide It will contain usage instructions for the customer e-wallet and the auditing system, from a user operational perspective. As the use-case is only a prototype with no production environment, this documentation will be developed mainly for demonstration purposes.

Programmer reference This documentation will be developed within the version control repository, writing README files with technical explanations about how to develop and deploy the components plus any complex procedure or aspect that may need a deeper description.

2.4 Operating Environment

As we will use GNU Taler, starting from an already existing solution, as base system for the prototype; the specifications of the operating environment will strongly depend on the requirements and current implementation of this system.

2.4.1 Operating System

We will use a Unix based operating system like Debian distribution for the prototype implementation as GNU Taler makes use of Unix oriented libraries for the configuration and deployment of the components. Apart from that, the web applications used by human actors will not require any specific operating system, from an operational point of view, as they will run in a web browser.

2.4.2 Library Support

Our prototype involves a lot of heterogeneous components that have been developed using several programming languages and architectures. Having this in mind, a huge amount of support libraries will be integrated in the system. These libraries will be used to support features of the programming languages or configuration and deployment of the components. Nevertheless, the most relevant library is the FENTEC toolkit that will be in charge of Functional Encryption operations. As we do not plan to deploy the prototype in a production environment, issues related to licensing, quality or performance will not be considered during libraries integration.

2.4.3 External Network

Our use-case system is intended to be deployed through internet using web layers. This requires implementing some measures to ensure security and privacy like secure transport protocols as HTTPS/TLS, input data formats, strong authentication features, etc. We consider these aspects as very important issue in production environments but for the prototype demonstrator these requirements are not mandatory.

2.4.4 User Interfaces

The user interfaces for our prototype will be very simple and demonstration oriented, implementing basic features of the components. Most of them are already existing in GNU Taler but they will need to be modified with FE specific implementations. In addition, some new components of our use-case will need a GUI (e.g. dashboards) that will be implemented from scratch.

2.5 Evaluation Metrics

2.5.1 Functionality

From technological point of view, the functionality of the prototype can be validated and measured by designing, implementing and running operational tests with code coverage and average of fails as key indicators. Also, interfaces analysis is an important point to verify functionality as we can measure the functional behaviour (expected vs. real) of every component either through a graphical user interface or internal execution flows.

2.5.2 Performance

Performance metrics can rely on times average for cryptographic computation (e.g. encryption/decryption) and general execution of the system (e.g. response times). Ideally, the performance should not be affected by including functional encryption operations but if so, the impact should be minimal.

2.5.3 Security

The evaluation of the pilot regarding security will have as goal to assess the security of those parts of the payment platform affected by the integration of FENTEC toolkit. These parts should be evaluated on different aspects regarding their functionality, therefore the strategy of the evaluation will follow OWASP Application Security Verification Standard 3.1

2.5.4 Feasibility

FENETC integrates into a payment platform in two scenarios, one of them strongly linked to the platform, therefore the feasibility of FENTEC in this pilot depends on the payment platform used as base and its own feasibility which could derive in unsuccessful results. In order to assess the feasibility of FENEC we will independently evaluate each scenario to introduce FENTEC in commercial products.

2.5.5 Reliability

The reliability of the implementation can be validated in long-term testing of the first scenario; purchases with special purpose coins, as set-up of the audit scenario where the goal is to audit big amount of invoices.

2.5.6 Legal aspects

The evaluation of Legal aspects regarding functional encryption is constrain to the fulfilment of GDPR, mainly in audit scenario in which private data is processed by FE techniques, although simulated users will be used in the testing.

2.5.7 Ethical aspects

Any ethical issues which arise as part of the project should be investigated by partners with appropriate experience.

2.6 Meeting Requirements

This section explains how the forgoing specification will allow the design to meet the requirements listed in D3.1[4].

2.6.1 Non-functional Requirements

DC.01 Unforgeability This requirement is inherent to the blind signature scheme of the payment platform chosen for the integration of the pilot. The introduction of FE in the process of issuing coins, in the special purpose coins scenario should become the system stronger in this aspect.

DC.02 Double spending prevention This requirement is provided by the payment platform and will be revised as the coin structure is changed with the introduction of FE.

DC.03 Collusion resistance The Functional Encryption schemes to introduce in the payment platform must ensure this requirement. It will be evaluated when schemes will be integrated to the platform.

DC.04 Untraceability This requirement is provided by the blind signature scheme of the payment platform.

DC.05 Unlinkability: This requirement is provided by the scheme of the payment platform. It will be verified that the way in which FE schemes are introduced do not affect to it.

DC.06 Auditability: The amount of data and its type; private or no private data, is agreed between parties in the establishment step and reflected in a signed contract. ABE schemes are introduced in the Audit scenario to ensure that access to data is done according to the terms of these contracts.

DC.07 Auditability agreement: These agreements are covered by the contract signed in the establishment. The terms in these agreements must be easily translatable into policies of ABE schemes.

DC.08 System availability: The architecture and deployment of the platform must ensure the system availability to provide services and avoid errors in on-line blocking operations. This will be verified during development.

DC.09 Money ledger: This requirement is provided by the scheme of the payment platform.

DC.10 Data storage: The design of the architecture should comply with this requirement. The Data storage, that will be allocated in the Exchange server will contain only cyphered documents to fulfil also requirement DC.05 and DC.06.

DC.11 Data availability: The design of the architecture should comply with this requirement. The Data storage will provide and API to, once the access is granted, to retrieve only the documents allowed.

2.6.2 Functional Requirements

DC.11 Secure CP and KP ABE: this requirement refers to ciphertext indistinguishability, namely two transactions should be indistinguishable to the adversary. To measure it, one may first quantify the acceptable advantage of an attacker in the system. Typically, ABE schemes consider IND-CPA with adaptive security. This property is satisfied if no group of colluding participants can find the difference among two encrypted messages, namely m_0 and m_1 under a chosen access structure by the attackers, so that an individual participant alone is unable to decrypt. We take into account that an attacker with access to the decryption oracle can replay queries to such oracle even after having received the challenge ciphertext in the game among attacker and its counterpart, the challenger. Due to this, we envision adaptive security model for the ABE schemes in our use-case and more generally in ABE schemes that require collusion resistance. Therefore, to measure DC.11, we quantify the acceptable advantage of an attacker in the system, which should be not higher than random according to the typical security definition of IND-CPA with adaptive security.

DC.12 Portability: To meet this requirement the payment platform provides a set of interfaces and APIs of the Web Based technologies such as REST FULL APIs (Representational State Transfer - Application Protocol Interface)

2.7 Conclusions

This specification aims to serve as starting point to begin with the implementation of the use-case. It provides the description of the schemes and processes to add and modify over the payment platform selected to serve as base of the pilot. The introduction of cryptographic schemes meets those requirements which are not fulfilled by the platform itself. Taler payment platform is provided in a basic implementation and some resources to emulate external parties such as banks, and count with a community of programmers who provide support to everyone interested in it.

3 Data Collection Prototype Technical Specification

3.1 Introduction

The WALLIX use-case will use FE technology to enable the gathering of statistical data over access to a web service (web analytics) such that the end-user's data privacy is maintained. The idea is that web service designers, implementers and maintainers can use this data to judge the efficacy of the provided service and make recommendations for improvements to the service.
The problem with this kind of data gathering is that it can sometimes encroach upon user's privacy. Thus we are proposing the use of functional encryption to provide users with additional guarantees as to the security of the data they contribute to such analysis.
Note that in this section we interchangeably use the terms "data gathering" and "polling" to refer to the data gathering phase of the process.
However, using the FENTEC toolkit to implement the demonstration prototype is only part of the work in this use-case since we also need to ensure the security of the implementation outside of the FENTEC core. We also need to develop a suitable means of presenting the technology to; the hosts of the service being analysed, the end-users of that service and the consumers of the analytics themselves. It is also important to maintain security between these parties as well as from outside influences.

3.2 Overview

3.2.1 Objectives

The objectives of this specification are to provide sufficient information for the construction of the prototype to proceed. We do this for three different aspects of the use-case: functionality, environment and performance.

3.2.1.1 Functionality The data collection prototype is required to gather simple data such as web-site access counts and perform statistical analysis upon them such that the data themselves are never available in unencrypted form to any party other than the originators of the data. Although this sounds like a simple statement of Functional Encryption there are actually several possible ways in which this functionality could be achieved. In the requirements analysis report D3.1[4] we indicated that the most likely method would involve distributed key generation of some kind. This assumption allows us to proceed with the technical specification while also allowing some leeway in the cryptography which may evolve during the lifetime of the project.

3.2.1.2 Environment Our context is the Awless CLI for controlling Amazon Web Services accounts. In this environment, we are constrained to use Golang as the development. In addition, Awless is open source and licensed under the Apache 2.0 license which means that if the FENTEC library is to be imported into Awless then only a component of that library which is also licensed under a compatible license will be valid. The actual programming environment occurs in multiple parts. Firstly, there is the user's code which is required to run the setup algorithm, participate in the redistribution phase and then perform the encryption. This will be integrated into the Awless CLI tool which is standalone compiled Golang code. Secondly, in the requirements analysis we identified the need for a proxy server to facilitate the redistribution phase. This component will be very simple and completely unrelated to any other part of the data collection prototype and therefore could exist in several possible forms. Finally, there is the decryption phase which will allow access to the computed statistical functions over the gathered data. This will also probably exist within the Awless sources but will need control over which parties can access it.

3.2.1.3 Performance The computational performance of this prototype is difficult to specify accurately partly because we do not know in advance factors such as the numbers of participants in a poll or the mathematical accuracy required in the results, and partly because there are questions about the computational characteristics of the cryptographic schema which will be used. In the requirements analysis we gave a basic, abstract bottom-line performance level, ie. about 100 integers valued in the thousands. For time performance we can state that we need about one day to complete a poll. In the requirements analysis, we actually stated about one week but we have subsequently reduced this in order to enable detection of statistical outliers by running successive multiple 24 hour polls over our intended period. Within this time frame, we allow time for the poll results to be gathered and then some additional time to compute the actual results. Obviously, we have some flexibility here and we can cope with minor performance problems by allocation of resources (such as more processors for decryption). However, we will have to be aware of issues such as maintaining security and costing of the resources used if they are, for example, cloud-based.

3.2.2 Characteristics

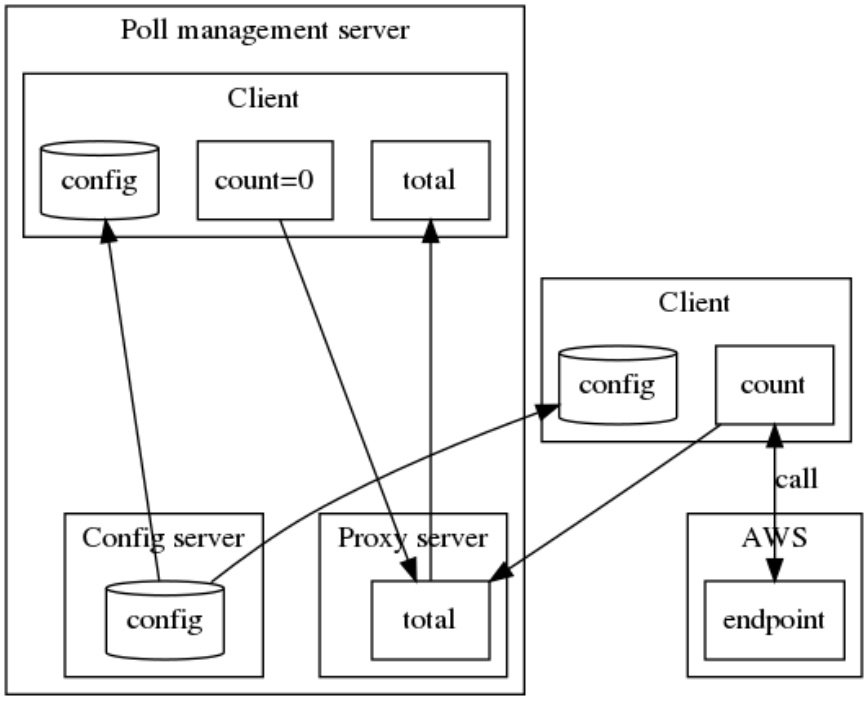

Figure 9: Web Analytics relationship between modules

An abstract diagram of the relationship between the principal modules in the design is depicted in Figure 9. To specify the characteristics of the use-case we envisage the following run-through of a successful poll. To initiate the poll, a poll configuration is created by a party with the authority to do so. The exact nature and location of this is left open. An endpoint will be required to link Awless into the poll using the configuration data.
The full configuration data will only become apparent once the actual polling system is written but it will contain values such as the number of participants, the length of time the poll is active, deadlines for responding to the different phases of the poll plus additional security information such as which participants will have access to the results. Additionally, we may require flags such as one for indicating the expected

behaviour if a poll should fail, for example if one of the participants does not return the required values in time. Note that the poll cannot actually commence until the total number of participants is known so there may be an additional time limit leading up to commencement of the poll to allow all participants to register.
Once created, the participants wait until the total number of participants is known which allows the cryptographic protocol to begin (when the Awless clients download any necessary cryptographic data). Each participant then moves into the actual data gathering phase.
The gathering of this information is already present in Awless although not currently activated so this code only needs to be reactivated and updated for the current use-case. Note, however, that the FE schema we use will only be able to encrypt and apply its function to a single value, eg. for one particular data point. However, most users of this code will need to monitor more than one such value. We will thus need to store and process multiple values of this type in parallel, say up to 20 data points as an initial guess. Handling these multiple values can most easily done by wrapping them together in communications but processing them separately at each site.
Which data points to monitor and statistically analyse will be stored in the configuration data. In other words the data points selected by the client will be named in a multi-valued field in the configuration file and passed to the Awless client in order to trigger monitoring of those data points. The configuration data should also include the basic encryption data for the poll for example the parameters needed to generate any public and/or private keys necessary for each participant to compute and transmit the relevant values.
Once the poll has started, ie. all participants have registered and the cryptographic routines have been initialized, the period of data collection begins and each participant starts collecting the required data (the link labelled *call* in Figure 9).
After the poll period is complete then the next phase is the redistribution of this data to allow combination of the encrypted data. Each participant sends their encrypted data to the proxy server (*Proxy server* in the figure), the address of which will probably be in the configuration file for the poll. It is likely that the proxy server will require access to the poll configuration data in order to enforce the privacy of the computed results to designated participants. Note that the configuration data is required in *all* of the depicted modules in Figure 9 (apart from AWS) and that this has been omitted from most of the modules for clarity.
Once all of the gathered encrypted results are in, the proxy server can then compute the required encrypted statistical functions. Using the methods currently available, this step is not computationally intensive. The second phase of the redistribution can then commence whereby the proxy puts the combined values (the *total* value in the *Client* module hosted by the *Poll management server*) online and they can be accessed by participants who are authorized to view the results. These results will probably only be online for a preset period of time and access will have to be secured by a mechanism which has yet to be decided.
If the poll should fail, for example one or more of the participants fails to upload their computed values to the proxy server then a recovery phase should be started. Either the poll is repeated and new values gathered which should hopefully include the missing values or the poll will have to be abandoned and the participants informed of the failure. It will then become the responsibility of the poll manager (the party which initiated the poll to begin with) to take appropriate action.
One other possibility for failure recovery is to ask a failed client to submit an encrypted zero count for their value. The FE algorithm could then be completed as normal and when the final result is computed the number of participants can simply be reduced by the number of zero values returned. In fact, each client could submit a zero value as part of their participation initialization.
The characteristics of this proposed system can be discussed in various contexts as follows:

3.2.2.1 Real-time constraints/targets The final implementation of the data collection prototype may require a certain amount of “tuning” to get right. Depending upon the number and type of participants, the number and nature of the values to be analysed and functions over them and the characteristics of the

cryptography and the communications, the deadlines may have to be adjusted to be able to complete a poll in the minimum time.
On the one hand, if the delays are too long then there will be unnecessary wasted time in the overall polling schedule. On the other hand, if the delays are too short, there is a higher risk of a participant not returning data on time. The balance will be difficult to determine but we may be able to mitigate this, for example at polling registration, the users could be asked what their likely or preferred polling times would be. This could be used to vary the poll delays to match each individual poll.
We reduce the values mentioned in the requirements analysis. We expect that a poll should be completed in one day, including gathering of data, redistribution of data and final analysis and distribution of results. Realistically, we can not put any more accurate times on the prototype at this stage.

3.2.2.2 Nature of the user interface From a user's point of view they will only see the initiation process as part of Awless. The address of the configuration data needs to be added to Awless in order to start participating in the data gathering. Management and control of the process is most likely to be via an API at the data gathering site.

3.2.2.3 Intended number of users Within a single poll there will only be one poll originator, the clients who participate in the poll and the managers of the hardware running the poll computation. As indicated above, we initially assume about 100 clients participating. In terms of the hardware required, the clients will be running on their own local machines but the site managing the poll will have to scale their hardware according to the size of the poll.
On the scale of managing polls, however, if we wish for example to run multiple polls from one site we will be obliged to scale the hardware accordingly. The software should be written with this scaling in mind.

3.2.2.4 Fault tolerance Although failed polls are a very real possibility for this system there are actually few options for recovery. The most obvious one is simply calling a re-poll upon error. However, it may be possible to detect such a failure early in the poll progression. For example, we may wish to implement a "*no response*" reply to the client's poll return values. This meaning that the client no longer wishes to participate in the poll despite registering initially. In this case we could trigger premature failure termination of the poll and re-issue the poll excluding the client that is no longer participating.
Another possible mechanism for dealing with such failures may be to enable a notification system which can prompt the failed client to respond. There could then be a warning, for example, some time before the poll closes issued to clients who have not yet returned their data.
Finally, as already stated above in Section 3.2.2, we can require each participant to provide default data at the start of data gathering in order to be able to disregard that user during the poll, if necessary. One potential problem with this method is that we may end up with statistically insignificant results (with too few final participants providing actual data).

3.2.2.5 Security features Security for the web analytics code can be divided up into two aspects. Firstly, there is the cryptographic security of the schema used to implement the functional encryption and secondly, there are other security aspects associated with the management of the polling data and controlling access to the results of the functional encryption.
The first aspect should be guaranteed by adequate proofs of cryptographic security provided by the academic research conducted as part of the FENTEC project. The details of the techniques ultimately deployed are currently unknown so for the purposes of this specification we define the cryptographic protocols in terms of abstract interfaces to the proposed implementations.
The second aspect is more varied but probably less likely to significantly change since they are well known aspects of securing data on websites and in normal executable code. Principally, generated keys for the

encryption have to be secured, unencrypted data must not be accessible outside their system of origin and the results of the decryption should only be accessible to pre-specified lists of clients.
How these different security aspects are managed will be developed during the implementation phase of the project. Rigorous testing of the code can eliminate some potential pitfalls and internal reviewing of code may also help.

3.2.2.6 Scalability and maintainability We have already mentioned some aspects of the scalability. For the web analytics prototype we are fortunate in that the overall structure of the prototype is quite simple which means that it should not be too difficult to arrange scaling, especially considering that the whole prototype is planned to be cloud-based. All cloud platforms have significant support for automatic scaling and load balancing.
Being a prototype the maintainability of the code is not a high priority, rather we wish this implementation to prove the viability of the new cryptographic protocols and to provide a model for more production-level code in future.

3.2.3 Architecture

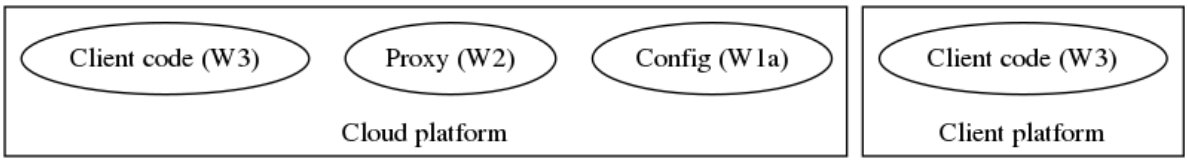

Figure 10: Web Analytics basic architecture schema

The architecture of the system is quite simple (Figure 10). We have a site, most likely cloud-based, which is responsible for managing the web analytics polling process. This system will provide connections for the Awless clients in order to initiate the poll within the Awless clients. Once the clients have downloaded the configuration data for the poll and then possibly asked for confirmation from the client's user the Awless instance then gathers the relevant data unassisted. Note that we have left the status of the proxy server as a separate system. This may actually just be implemented within the cloud framework but, alternatively, may be kept separate for security reasons.
In architectural terms, we simply need some interface code plus access to the FENTEC library in order to define an Awless instance as a client system within the overall polling architecture. Also in architectural terms we do not actually need any more hardware than that described already unless we get into difficulties with the performance of the decryption phase. In this case we may need to specify additional hardware, perhaps even dedicated FPGA support, as part of the architecture of the polling management system.

3.2.3.1 Physical placement of servers, number of processors The prototype will include a single cloud-based system which will provide a central management site for the web analytics process. Note that this site will never contain any sensitive data from clients apart from data which concerns the poll itself, mostly the configuration data. This site will, however, also be responsible for securing access to the results of the polls and limiting access to selected clients.
As such, it is unlikely that this server will require a large number of processors. For the purposes of the prototype, we envisage a single processor implemented on AWS.

3.2.3.2 Nature and placement of client systems Since the client systems are effectively a large distribution of heterogeneous systems which comprise the Awless client community, we do not actually have any effective direct control over the architecture of this system. However, we are in control of both the poll management site and of the software running on the client systems which opens up possibilities for controlling the client architecture, at least in a virtual sense. For example, we can put delays and checks into the client systems for uploading results to the proxy server in a manner which does not swamp the central proxy server. The crudest method being to simply cause each client to wait for a random delay before uploading the results.
We cannot predict exactly what measures will be needed throughout the development of the prototype. Some will be required as problems appear during development, other more predictable such measures may still require experimentation to arrive at a workable solution. The developers of the prototype are encouraged to adopt this type of method whereas this technical specification should take into account the speculative, experimental nature of the prototype we are trying to develop.

3.2.3.3 Nature and placement of analytics Initially, due to the nature of the cryptography we intend to use, all of the client systems will have the potential to compute the resulting analytics. There are two problems here, firstly, we need to control who actually has access to the results and, secondly, there may be significant amounts of computation required to compute them.
For the purposes of the prototype and to keep things simple we specify that the decryption is performed by any designated client system. Using the configuration file we indicate which clients will be sent the functionally encrypted results which should contain the Awless client controlled by the poll originator. It is then up to that particular client to provide the necessary resources to perform the decryption. For the future, the poll organiser may provide a centralized system to do the decryption but this creates extra problems of security in terms of distributing the results to the authorized clients.

3.3 Design

In this section we present a specification for developing a web analytics prototype according to the requirements in the Requirements Analysis deliverable D3.1[4] and according to the parameters already outlined in this document so far.

3.3.1 Methodology

For the WALLIX web analytics prototype we intend to use a test-driven methodology. This is partly because a large proportion of WALLIX's existing code is written in Golang and that Golang has support built into it for this type of test driven development (TDD). It is also partly because we are developing a prototype so the design itself may have to change to a small extent during development and TDD should allow us to maintain integrity and functionality in the code while other sections of code are written.
For TDD, note that it is more usual to start by writing tests and then produce the code to meet those tests. This is intended to be done incrementally, in fact it is accepted practice to write tests and code which minimally meet the requirements of the functionality at each point and to refactor the code as more tests and code are added. At each point in the development the most recent tests plus all previous tests have to be satisfied.
The ensemble of the tests itself constitutes a specification of the required functionality which the code has to meet and evolves as the code development progresses. This technique has several advantages; it ensures the quality of the code by focussing on the requirements, it encourages neatness in the code since it is by nature broken down into small steps, the tests constitute a form of documentation for the code, and at the

end of development the tests become a set of regression tests for the code and any subsequent alterations or bug fixes.

3.3.2 Software Development Tools

Here, we list the programming languages and development tools used for the WALLIX web analytics use-case.

The FENTEC Library contains all of the cryptographic primitives needed to implement the cryptographic protocol for the use-case. This will include the setup, encryption and decryption functions plus any auxiliary code needed by the proxy server to compute the redistribution phase of the algorithm. More specifically, this will be the Golang version of the FENTEC library which should implement the distributed protocol described in the requirements analysis.

Golang is the programming language for the use-case, for the main web analytics code and also, most likely, for any additional code such as the proxy server. Note however, that Golang is also supplied with the testing facilities required by the test-based methodology proposed for this use-case.

Web programming may be necessary for initiating web analysis polls. In this case, we may propose using code from our PEPS secure file product. While this is also written in Golang it contains the core of an advanced web server which we could adapt for the WALLIX FENTEC use-case. There may be copyright issues involved but we should be able to arrive at a licencing arrangement of some kind. In any case, the HTML and client inset code will be minimal and if there are problems using existing WALLIX code for this we can simply author the FENTEC code from scratch using similar methods.

3.3.3 Programming Standards

The WALLIX use-case will be developed using the Effective Go programming guidelines[5]. These are only general guidelines but have been found to be effective at WALLIX for allowing coordination and management of coding activities by multiple programmers. These methods will probably be unique to WALLIX unless one of the other partners takes up Golang as a development language.

3.3.4 Decomposition into Components

3.3.4.1 Functional specification of each component

Breaking our design down into components results in the following list:

Poll management server As already indicated, this server is required to manage the polling process as part of the web analytics gathering. This will host the configuration server, defining parameters such as; the maximum number of participants, the list of endpoints to be monitored, and the clients who are to be allowed access to the results.

Subcomponents are as follows:

- **Configuration host (W1a)** The configuration host should be posted online in order to allow the clients to download it into their Awless configurations. Although this is a simple endpoint on the server it needs to be secure and only respond to clients who are subscribed to the poll.
- **Result host (W1b)** This is optional and may be needed to allow authorized clients access to the results if significant computation is required to generate the results.

Proxy server (W2) This is a simple component whose only role is to collect the encrypted data from the clients, perform the combination phase of the cryptography and return the combined result to the selected clients. As such this component will be kept monolithic and as simple as possible.

Client code embedded into Awless There are several aspects to this code resulting in several subcomponents:

- **Initiate participation in poll (W3a)** This component takes the address of the endpoint on the configuration host and downloads the configuration file from that host. It then sets up the cryptography and starts gathering the information indicated in the configuration. A future enhancement may involve the Awless client polling the configuration server for newly activated polls but for now it is envisaged that the user manually copies the address of the configuration host into an Awless command.
- **Data gathering module (W3b)** This simply gathers the requested information in the configuration and keeps required statistical summaries. For security, these may need to be encrypted locally.
- **Encrypted data dispatch (W3c)** This component is required to monitor the progress of the data gathering operation and then perform the encryption phase of the cryptographic protocol. It then transmits the encrypted data to the proxy server for redistribution.

FENTEC library (W4) This is an important component but we are not currently in a position to specify the exact protocol to be used. To allow development to proceed, however, we define a simple, abstract interface to a notional distributed multi-client technique which is to be provided by other FENTEC partners.

- **Initialize (W4a)** Create any keys and other information required to initiate the cryptographic protocol.
- **Encrypt (W4b)** Encrypt the raw statistical data in preparation for the redistribution phase.
- **Redistribution function (W4c)** Not strictly part of the protocol but within the proxy server we require a function to be applied to the encrypted data to give the combined data. The communications which are part of the redistribution phase are separate from this function.
- **Distributed key generation (W4d)** A precursor function to the decryption routine. This also needs to be redistributed but can be merged with the redistribution function for efficiency.
- **Decryption (W4e)** Take the computed encrypted result and compute the final cleartext result.

3.3.4.2 Relationship between the components

Apart from error states, the computational path for a successful poll is essentially linear. The poll is started at the polling management server, the clients download the poll configuration and gather the data, this data is then encrypted and sent to the proxy server which then computes the combined data and returns it to the authorized clients for decryption and, potentially, redistribution of the results. The only branch in this scheme is when a poll fails and a decision is made whether to reissue the poll or terminate with failure.
However, the code structure is more complex since two of the components need access to different parts of the FENTEC library, the clients need access to the initialize, encryption, distributed key generation and decryption components at different phases of the operation. The proxy server also needs access to the FENTEC library. An additional complication is the configuration data which has to be accessed by all of the other components.

In terms of structuring the code the prototype should follow the above description of each component. Placement of code in the working prototype should also match the above description. There are still some issues to be resolved with securing the different types of data, including the generated keys and the final cleartext results but these can be resolved during prototype development and documented in the final specification document D7.2.

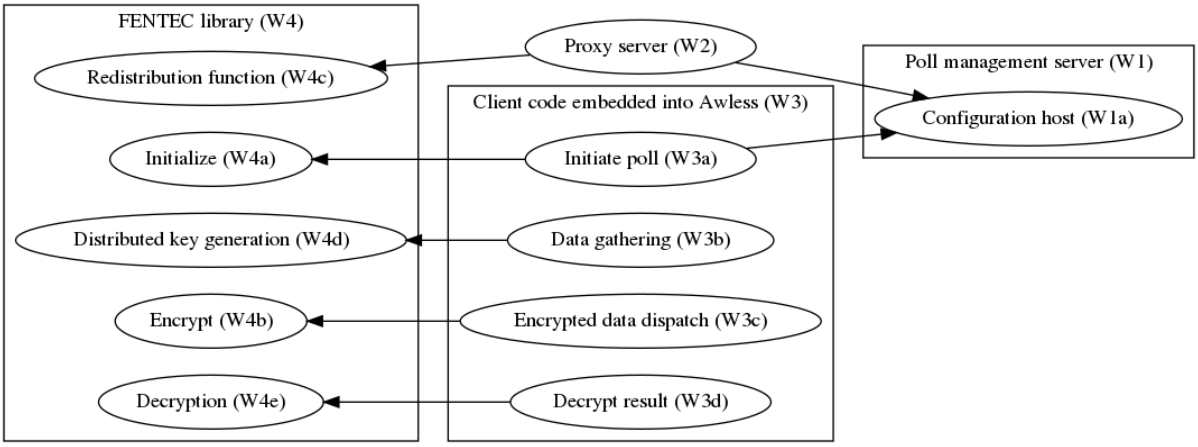

Figure 11: Web Analytics software dependencies between components

Figure 11 shows a very rough software dependency relationship between the components. Most of the modules require access to the configuration data and most of the client routines need access to some of the FENTEC library components, on addition being the *Proxy server* which also needs access to the redistribution function.

3.3.4.3 Definition of interface layers The interfaces between software components mostly involves specifying the common types used by each component. Since these types cannot be specified completely in advance we will develop the types and the interface layers during development and then report more completely in D7.2.

3.3.5 Detailed Description of each Component

The description of each component should provide sufficient information for a programmer to code the component and for a maintainer to debug or upgrade the component. We also follow the structure of the above list of components.

3.3.5.1 Configuration host (**W1a**)

3.3.5.1.1 Type Secure online data server.

3.3.5.1.2 Purpose This component will host an endpoint at which the Awless clients can download the configuration data for a poll. It is not simply an open-access server, however, since it should only respond to client systems indicated by the configuration file.

3.3.5.1.3 Functionality When an Awless client is given the address of this endpoint it connects and provides suitable authentication information. If this is valid the endpoint returns the current configuration data for the poll. It should then notify the poll management server that the associated client has joined the poll.

3.3.5.1.4 Dependencies The shared configuration data and the notifications system.

3.3.5.1.5 Interfaces This component provides the endpoint and can possibly connect to the poll management server for notification.

3.3.5.1.6 Computational requirements Minimal.

3.3.5.1.7 Data requirements Authentication data for each client, scaled according to the expected maximum number of active polls.

3.3.5.2 Result host (**W1b**)

3.3.5.2.1 Type Secured online data server.

3.3.5.2.2 Purpose Optional service for distributing the results of polls to authorized clients who do not have access to sufficient processing resources to perform the decryption.

3.3.5.2.3 Functionality This component acquires the cleartext results of polls from the decrypting system (expected to be the Awless client managed by the configuration server). Clients participating in the survey can subscribe to this service and receive the cleartext results of the polls if authorized by the poll originator.

3.3.5.2.4 Dependencies The Awless instance controlled by the poll management site and the shared configuration server which should contain the authentication information for valid clients who can receive this information.

3.3.5.2.5 Interfaces External web interface, interface to the Awless client with the results and an interface to the configuration server.

3.3.5.2.6 Computational requirements Basic client authentication for the expected maximum number of clients.

3.3.5.2.7 Data requirements Sufficient data storage for the number of active polls multiplied by the size of a poll result. There may be a fixed residency time for poll results, however, since for security reasons, once all valid clients have accessed their poll results for a given poll or a set maximum time has passed, the poll results should be forgotten.

3.3.5.3 Proxy server (**W2**)

3.3.5.3.1 Type Secure online aggregation service.

3.3.5.3.2 Purpose The purpose of this component is to provide an online server for the redistribution phase of the cryptographic protocol.

3.3.5.3.3 Functionality Each client is expected to make a connection to this server and upload the encrypted results of the statistical data. Once all of the clients participating in that poll have uploaded their results, this server computes the sum of the encrypted data and places it online for authorized clients to connect and access the computed sum. In the event of a client not submitting data on time, this server contacts the poll management server to trigger error processing for the poll.

3.3.5.3.4 Dependencies The configuration server, the poll management server, the FENTEC library.

3.3.5.3.5 Interfaces Public but secure connection for clients to connect to. The address of this port should be included in the poll configuration file. Connections to the configuration server plus possibly the poll management server may also be required.

3.3.5.3.6 Computational requirements The currently available method (DMCFE) only requires minimal computation to perform the combination phase of the algorithm but depending on the maximum number of connecting clients we may need significant bandwidth for the connection.

3.3.5.3.7 Data requirements Sufficient to store arguments and results for the maximum number of currently active polls.

3.3.5.4 Initiate participation in poll (**W3a**)

3.3.5.4.1 Type Executive module.

3.3.5.4.2 Purpose Accept the address of a poll configuration file from the command line, perform both cryptographic initiation and initialize the gathering of statistical information.

3.3.5.4.3 Functionality Connect to the configuration server and download a poll configuration file. This file should contain the information needed to perform the `Initialize` phase of the cryptographic protocol. Once this data is initialized, the gathering of statistical data can be started, most likely using flags or callbacks into the more general Awless code.

3.3.5.4.4 Dependencies The FENTEC library for the cryptographic routines and the online configuration host. This component will also depend upon the data gathering module **W3b**.

3.3.5.4.5 Interfaces Connection to the configuration file server.

3.3.5.4.6 Computational requirements An Awless client with sufficient computational resources to compute the `Initialize` phase of the cryptography.

3.3.5.4.7 Data requirements Storage requirements for any local private data needed to perform cryptographic functions.

3.3.5.5 Data gathering module (**W3b**)

3.3.5.5.1 Type Simple data storage and manipulation module.

3.3.5.5.2 Purpose Gather the statistical information requested in the poll configuration file.

3.3.5.5.3 Functionality During the data gathering phase of the poll this module intercepts calls to AWS endpoints, compares with the list of endpoints being monitored and maintains statistical information about that endpoint, initially just the count of times the endpoint is accessed but may require other values later on, for example the square of this value to compute the variance.

3.3.5.5.4 Dependencies Processing of statistical information will probably be triggered by some flags internal to Awless. The dependencies are therefore mostly all with Awless as well, ie. various internal modules within Awless.

3.3.5.5.5 Interfaces Interface to Awless' AWS client code.

3.3.5.5.6 Computational requirements We have to keep processing to a minimum in order to prevent degradation of Awless' performance. Maintaining the counts of endpoint accesses will have minimal impact here but we should be careful how we implement the checking of each endpoint. Several possible algorithms could be used to speed up this aspect of the component.

3.3.5.5.7 Data requirements Flags or callbacks to trigger data collection plus storage space for the selected statistical data. Expected to be minimal.

3.3.5.6 Encrypted data dispatch (**W3c**)

3.3.5.6.1 Type Client code.

3.3.5.6.2 Purpose Upload encrypted statistical data to the proxy server.

3.3.5.6.3 Functionality When the data gathering period ends, the raw cleartext data is encrypted using the `Encrypt` phase of the cryptographic protocol. This data plus any other required data such as decryption keys from the `Dkeygen` phase of the protocol is then sent to the proxy server indicated in the configuration file.

3.3.5.6.4 Dependencies The shared configuration file, the proxy server and the FENTEC library.

3.3.5.6.5 Interfaces External connection to the proxy server, must provide an internal connection to allow the Awless client to trigger data upload upon successful completion of the data gathering phase.

3.3.5.6.6 Computational requirements Sufficient to perform the encryption and distributed key generation which, for the DMCFE method are actually not significant. The data itself will not be large and uploading should require minimal computational resources.

3.3.5.6.7 Data requirements

3.3.5.7 FENTEC Library (**W4**) For this component we are not required to write the associated code so we depart from our previous format for specifying components and instead define a loose abstract interface for the library functions we wish to specify.

3.3.5.7.1 Initialize (**W4a**) Initialize the protocol according to the cryptographic parameters for the underlying method. Should return a structure with all necessary parameters for the remainder of the protocol to be completed.

3.3.5.7.2 Encrypt (**W4b**) On a *per client* basis, accept an integer (possibly scaled to within a limiting value), the pre-computed cryptographic parameters plus any indices and return a functionally encrypted value for the integer.

3.3.5.7.3 Redistribution function (**W4c**) We assume that all of the redistribution phases have been merged into a single phase. Given the encrypted values from all of the clients plus a set of distributed decryption keys also from each client, perform the combining operation by summing the values and aggregating the keys into a single array. The resulting combined value plus the set of keys are returned.

3.3.5.7.4 Distributed key generation (**W4d**) On a *per client* basis, given the `y` values from the inner product <`x,y`> plus the cryptographic parameters, return a distributed key value for the client.

3.3.5.7.5 Decryption (**W4e**) Inputs are the cryptographic parameters, the aggregated values and distributed keys plus the `y` values and the indices from the `Encrypt` phase, compute the inner product value. This should be in cleartext form and may involve significant computation.

3.3.6 External Interfaces

Here we specify the external interfaces for the prototype.

3.3.6.1 User interfaces The user interface for the web analytics code consists of the commands added to the Awless CLI to allow clients to join a poll. This interface is not complicated due to the relative simplicity of the use-case.

3.3.6.2 Hardware interfaces The only real hardware interface needed will be if FPGAs are required to implement the decryption phase for performance reasons. This should be easy to integrate into our code since the code will be implemented as an AWS service and FPGA facilities are integrated into these services.

3.3.6.3 Software interfaces There are a large number of software interfaces in the use-case. In this specification we restrict ourselves to the major interfaces between the components of the use-case plus the FENTEC library developed by other partners in the project. We may, however, need to avail ourselves of the shared memory or DB interfaces also provided as part of Amazon's AWS service.

3.3.6.4 Communication interfaces There are several communication interfaces as part of the use-case. The various data servers needed to compute the cryptographic protocol plus internal interfaces within systems which may be required to integrate the various components. Each of these needs to be secured, especially the ones visible on public networks. Internal interfaces also need to be checked for security to prevent sensitive data from leaking into other, potentially vulnerable, parts of the system.

3.3.7 Documentation

Given that we are developing a prototype and that we do not intend to produce a final sellable product at the end of the day we will tailor our documentation for the purpose.

User manual The user manual need only be sufficient to allow potential testers of the poll initiation and monitoring pages to understand how to setup and run a poll. A small amount of additional documentation will ne necessary to guide participants in polling tests to use the Awless side of the prototype.

Programmer reference This is intended to guide future programmers in developing commercialized versions of the code. We do not need maintainability for our code but we need to explain the problems overcome as part of the development and provide suggestions as to improvements or alterations leading to such commercial exploitation.

Technical specifications These should be complete, particularly for aspects of the system which are novel or otherwise contain hidden complexities. We do not need to justify our design choices precisely rather than to explain the system we ultimately envisage the prototype leading to.

3.4 Operating Environment

The operating environment for the use-case consists of the Awless client environment written in Golang plus the data servers specified above.

3.4.1 Operating System

For the prototype we will use Golang throughout. This provides an abstract interface over several operating systems and is very portable.

3.4.2 Library Support

The FENTEC library is the most important support library but we will make extensive use of many of the system libraries provided as part of Golang. In addition, Golang has an extensive online user-contributed set of libraries for specialized purposes. However, we should be aware during prototype development that some of the library support might not be of sufficient quality (in terms of support, performance or licensing arrangements) for production code.

3.4.3 External Network

Our prototype is based on the internet and requires adequate security throughout. As mentioned previously, we also need to ensure internal security as well.

3.4.4 User Interfaces

Our user interfaces will draw upon our experience developing web-based secure file servers. We can draw upon significant support for web design activities at WALLIX although this is not a major component of our intended prototype.

3.5 Evaluation Metrics

In this section we give some metrics for evaluating the success of the web analytics use-case.

3.5.1 Functionality

3.5.1.1 User interfaces These can be validated by subjective means, synthetic data can be entered into the required fields and the resultant poll behaviour verified either manually or by placing checkpoints in the computation path in order to verify the system state at that point. Note that since this prototype will be developed using test-based design we will already have significant error checking facilities available.

3.5.1.2 Statistical analysis The statistical analysis, while simple, should be verified against known data. Simple averages can be easily verified against synthetic data. If successful, we may also look at computing variances which can be verified in the same way.

3.5.2 Performance

3.5.2.1 Cryptographic computation We have given very rough guides for this performance. This can be validated in the prototype using profiling techniques which can be built into the code from the start and retained for future verification should the prototype change.

3.5.2.2 Polling characteristics This may have to be evaluated manually by running experimental polls. It would be better to setup real Awless instances for this purpose for verification accuracy but due to the restrictions on live data for the project this may ultimately have to be synthetic data.

3.5.3 Security

3.5.3.1 Cryptographic security The security of the prototype should be validated where possible. This may mean setting up artificial malicious entities in order to force leaks of data or even cryptanalysis.

3.5.3.2 Web analytics security We also need to verify that our implementation is secure from either leaks, bugs or external attacks. Different techniques may be required for this validation. In the past we have used formal analysis of our protocols to eliminate some potential pitfalls but that may be outside the scope of the FENTEC project.

3.5.4 Feasibility

The feasibility of the techniques used will be validated by building a successful prototype. To a certain extent this project is speculative and may lead to negative results in some cases. In any case, whether we can build the prototype or not, we need to assess the feasibility of the method for commercial exploitation.

3.5.5 Reliability

The reliability of our implementation can be validated by long-term testing. For example, we can use our automated testing facilities (such as CircleCI) to monitor long-term usage of the prototype.

3.5.6 Legal aspects

Evaluating the legal aspects of out methods is outside the scope of WALLIX's contribution to Functional Encryption but we should be prepared to participate in any such assessment, particularly with respect to GDPR.

3.5.7 Ethical aspects

Any ethical issues which arise as part of the project should be investigated by partners with appropriate experience.

3.6 Additional Requirements

3.6.1 Performance

The performance goals given for the prototype above are only for the immediate future. In terms of implementing the techniques used for a production level environment we may need to impose further performance goals.

- Computational requirements for FE components may change once further research has been completed on distributed methods. The requirement for a discrete log computation may be removed eventually but other components may then have a higher computational load.
- Memory usage size for FE components can be significant. Our estimates of polling size may be required to increase by several orders of magnitude once deployed in the real world and this may in turn affect the sizes of keys, for example.
- Communication loads are concomitant with key sizes and the architecture of our system may need to be adjusted, for example using multiple servers to achieve sufficient bandwidth.
- An assessment of the viability of our methods in the deployment of production software should be considered if we wish to take the method forward and use it in a commercial setting.

3.6.2 Safety

A full safety audit for the prototype is outside the scope of the FENTEC project. However, we should review our methods and implementations with a view to safety concerns, particularly with respect to the security of sensitive data. Although the data considered by our prototype is not critical (we have web site access counts which cannot compare with the potential for damage relating to, for example, bank account access codes) our methods may be applied in future to more sensitive data.

3.6.3 Security

The security of the methods used and their implementation, as least as regards to the prototype constructed during the FENTEC project, will be built in from the start and rigorously maintained throughout the development process. Testing of the final version of the prototype should be as comprehensive as possible. At the end of the project we wish to be in a position to provide concrete, verifiable guarantees as to the security provided by our techniques.

3.6.4 Quality

Since we are not generating production quality code and systems we are not really constrained by quality control issues, however we should at least make an effort to review our code internally in the manner of a quality control review if we have sufficient resources to do so.

3.7 Outstanding Issues

The web analytics use-case is quite small and simple, relatively speaking. This version of the specification is more or less complete but we may need to omit some of the non-essential components due to time or resource constraints, if necessary.
The final version of this specification to be published in D7.2 in M25 should contain all of the elements specified by this document and may also include elements not envisaged here but which become necessary during development.

3.8 Meeting Requirements

This section explains how the foregoing specification will allow the design to meet the requirements listed in D3.1[4].

3.8.1 Non-functional Requirements

WA.01 No central authority

In our abstract descripition of the cryptography we intend to use, we assumed that there is no central authority (the proxy server does not count since it does not have access to unencrypted data or even any keys). Any method we adopt will have to follow this criterion.

WA.02 Adequate implementation security

Our design has left some of the security decisions open but we have specified in the simplest terms the minimal requirements of the security provisions. Work during construction of the prototype will bear this aspect of the design in mind throughout. We have also specified that security tests be applied once the prototype is completed.

WA.03 Minimize interactions

This is implicit in our design incorporating the proxy server for redistribution of combined data.

WA.04 WALLIX use-case anonymous statistics conform to GDPR requirements

This is not within WALLIX's area of expertise but we intend to work with colleagues on the FENTEC project to test whether our implementation is compliant with GDPR or not. It should be noted that the relationship between partial encryption/decryption and GDPR is an area which has not been investigated.

WA.05 Client raw data should only be accessible by the data owner

Our specification should in theory comply with this requirement. Our data path starts with encryption by the clients and ends with decryption of FE results by authorized users. This means that due to the nature of FE itself, only the client which originates data (the web access counts) has the data in an unencrypted form.

WA.06 WALLIX use-case can be computed in reasonable time

This will be verified during development but preliminary work has indicated that at least the proposed DMCFE method will have performance adequate for the use-case. The ancillary components will also need to be checked for adequate performance, particularly the proxy server.

3.8.2 Functional Requirements

WA.07 The scheme implementation for the WALLIX use-case should be secure and with adequate performance

This will be guaranteed by the cryptographic work as part of FENTEC and also checked by the FENTEC library implementers.

WA.08 WALLIX implementation should be under acceptable license

We can ensure the WALLIX code conforms to a suitable licence and we are aware of at least one third-party base library which is also conformant.

WA.09 Usable and affordable hardware support for WALLIX use-case

Costing of the prototype probably will not actually reveal the true cost of a commercial or production deployment but we can certainly scale our own resource requirements according to current cloud architecture costs.

3.9 Conclusions

In summary, we believe this specification provides sufficient information to commence work on implementing the use-case. The prerequisites have all been met in terms of cryptographic protocols and other library support as well as hardware requirements which are currently minimal (although this may change once enough code has been written for performance measurements to be conducted). The base platform (Awless) already exists, as does the expertise in terms of methodology. There is also a sufficient set of metrics to judge the success of the use-case although these could not be quantified to any great extent.

4 IoT Prototype Technical Specification

4.1 Introduction

The Kudelski use-case will use Functional Encryption to enable privacy-preserving video processing. The Kudelski Group has a significant business in Digital TV security—being the market leader in this domain—and since recently has extended its market reach into the Internet of Things. Its video processing and security expertise is being leveraged for providing solutions to Video Surveillance operators. Kudelski has identified an opportunity with Functional Encryption in this domain and wants to further explore this in the FENTEC project.
In this section, we provide an initial technical specification of the KUD IoT prototype.

4.2 Overview

4.2.1 Objectives

The objective of this specification is to provide sufficient information on the use-case such that the development in this project allows integration into a suitable prototype. We do this for three different aspects of the use-case: functionality, environment and performance.

4.2.1.1 Functionality The business objective of Video Surveillance with respect to (1) secure access to the content, and (2) to protect the Video Cameras. The latter need has become clear with the Miria Botnet, where CCTV cameras have been hijacked and used for launching DDOS attacks. To secure access to content, Kudelski is proposing to Camera Manufacturers a key management and access management solution which permits to have video content encrypted and manage who can have access to what content. Operators (such as for example Video Surveillance firms) can as a result make sure that only authorized personnel can have access.
The problem with 'classic' video encryption systems however is the all-or-nothing property; similar to the problem statement of FENTEC. Properties of the video stream can only be obtained when having the decryption key available. With functional encryption solutions, we aim to change this paradigm and allow properties on video streams to be obtained without decrypting the video stream. FE schemes can enable the derivation of specific properties of video streams whilst preserving the privacy of PII data in the stream. For example in the Kudelski use-case, we want to develop a prototype of a Video Surveillance system, where an observer that receives encrypted video streams can determine whether or not there is (significant) motion in the video image or not. Such a prototype can be applicable as follows:

- Video Surveillance cameras are oriented on industrial equipment such as electrical power substations or large industrial facilities.
- During normal operation, there should be no (significant) activity around those facilities. No human activity is expected since those facilities are intended to work without local interaction—except for maintenance operations.
- The video stream is sent to a back-end system which stores the stream and/or displays it in a Video Surveillance Control Center.

To protect the video stream, the stream is encrypted by the camera (or cameras are retrofitted with a special-purpose component that does the encryption).
With FENTEC technology, we introduce an additional component on the network that is able to process the video stream - in a privacy preserving manner. In the prototype setup, we present a gateway to which

the cameras are connected. The gateway analyses the video stream with the goal to assess if there is motion detected around the facility or not. If there is motion (beyond a certain threshold), an alarm can be raised. The alarm could then trigger different sorts of behaviours. Typically, the gateway is load-balancing video streams coming from different cameras before sending variable quality streams to the backend. The alarm could trigger the transmission of the highest quality stream available from the camera where motion has been detected, instead of transmitting lower quality streams to the backend, or instead of dropping the stream completely to save bandwidth. Obviously, the alarm could also trigger actual alarms in control centers.

4.2.1.2 Environment

Our prototype is thought to work at three different levels:

- video surveillance cameras, in charge of generating the video content, and encrypting it in order to transmit it to the backend system.
- gateways, in charge of raising alarms if needed, for instance if more than one camera are detecting movement at the same time, or when a set threshold is reached by a camera, as well as forwarding the encrypted streams to the back-end system. Note that the gateways can also be in charge of performing QoS and might drop certain streams if these are deemed not relevant by the local decision making performed on the gateway using the information it can access about these streams thanks to the functional encryption scheme used to encrypt them.
- a back-end system, in charge of storing and/or displaying them in a Video Surveillance Control Center.

Cameras being very specific devices, we might have to use an external component such as a raspberry pi, or an FPGA device, in order to have the computing power required to demonstrate our use-case.
In a later iteration, the target platform for the gateway could be for example a raspberry pi but in its first stage, the gateway could simply be simulated using any regular computer, featuring for example a 3Ghz quad core processor.

4.2.1.3 Performance

Real-time processing of the encrypted video stream in order to raise alerts, and real-time decryption of the encrypted video stream in order to allow an operator to watch the live stream when needed. The processing needs also to be real-time in the sense that the video throughput should not be affected, even if the result of the processing is delayed.
More precise performance requirements are defined in section 4.6.1.

4.2.2 Characteristics

The prototype will be built in three phases. In the first phase, we will implement the different elements described below in section 4.2.3 such that they are working as intended, without any sort of encryption. We would be working with cleartext motion-vectors and non-encrypted frames, and the gateway would simulate the local decision making by computing the function that must later be extracted from the ciphertext using functional encryption.
In the second phase, we will implement the functional encryption scheme, allowing the local decision making to be done on the encrypted motion-vectors.
In the last phase, we will encrypt all the video streams using common encryption techniques, except for the motion-vectors that would still be using the functional encryption method implemented in the second phase.

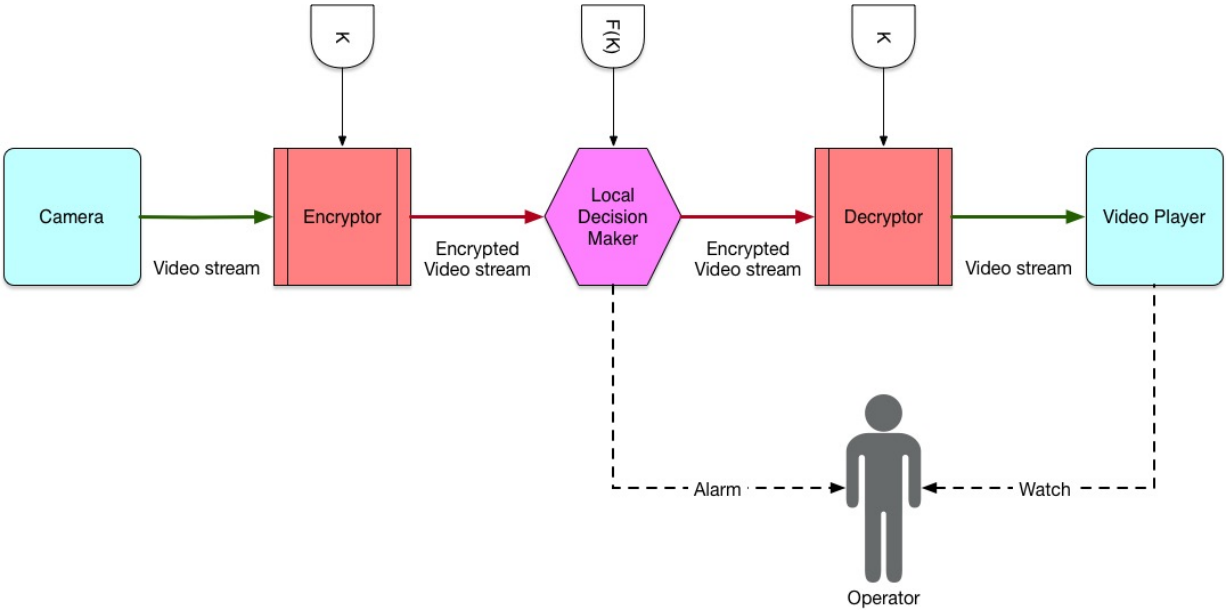

Figure 12: The functional architecture of the prototype

Once this third phase has been achieved, we shall have met all the requirements described in Requirements Analysis deliverable D3.1 [4] as well as in the current document, notably those further described in section 4.6.

4.2.3 Architecture

The functional architecture of the prototype is best described with the schema in Figure 12.
The functional blocks are:

- Camera. Records the scene and produces a video stream (H264 video).
- Encryptor. Encrypts the video stream using a Functional Encryption function configured with key K.
- Local Decision Maker. Extracts motion information from the encrypted stream using Functional Encryption function F(K), without decrypting the stream and without knowledge of the key K. Depending on the extracted data (e.g. motion detected), generates an alarm to alert the operator.
- Decryptor. Decrypts the video stream using the Functional Encryption function configured with key K.
- Operator. Is alerted by the Local Decision Maker and watches the video player when motion has been detected.

In this particular prototype, the synthetic information extracted from the encrypted video stream is the result of a mathematical function computed over the motion vector information found in the video stream. For instance, an average value of the motion vector for the image is computed and compared to a defined threshold. If the computed value is above the given threshold, it indicates that a movement has been detected in the encrypted video stream and adequate action may be taken, such as reviewing the video stream.
Note that the separation between the Camera and the Encryptor, or between the Decryptor and Video Player,

is purely pragmatic and is made to facilitate the setup of the prototype. In a production environment, the Encryptor function would be embedded in the secure Camera and the Decryptor function in the secure Video Player

4.3 Design

In this section we present a specification for developing a IoT-oriented video encryption prototype according to the requirements in the Requirements Analysis deliverable D3.1 [4] and according to the parameters already outlined in this document so far.

4.3.1 Methodology

The methodology used for this use-case is one of incremental development, where the functionality is developed in small controlled steps with frequent software releases that allow continuous testing and validation. This keeps the risk of regression or breaking changes to a minimum while providing flexibility and the ability to make changes depending on the evolution of the project without jeopardizing the end goal.

4.3.2 Software Development Tools

Here, we list the programming languages and development tools used for the IoT video encryption prototype use-case.

The FENTEC Library contains all of the cryptographic primitives needed to implement the cryptographic protocol for the use-case. More specifically, this will be the C version of the FENTEC library.

C++ and C are the programming languages chosen for the use-case.

4.3.3 Programming Standards

The use-case will be developed mostly using Object Oriented Programming (C++), as well as functional APIs (ANSI-C) when required for interoperation with external libraries.

4.3.4 Decomposition into Components

We describe here in more detail the actual setup of the prototype. Some elements are simulated or replaced by placeholders to ensure reproducibility of the proof of concept and make development more manageable. The encryption of the video stream is also slightly different at this point than the theory and makes use of two encryption schemes:

- The FE encryption scheme for the motion vectors, applied on the motion vector raw data before encoding and packaging into the video data.
- A more traditional symmetric encryption scheme for the rest of the media streams, applied after encoding and packaging of the data into the NAL.

Moreover, since the encrypted motion vectors must be accessible without decrypting the media stream, the encrypted motion vectors are carried in custom NALU packets that are not encrypted by the symmetric encryption scheme.

A future implementation will improve on this approach and bring back the bandwidth footprint closer to its unencrypted counterpart.
The physical and logical architecture of the prototype is described in Figure 13.
In this architecture, the camera is replaced with a static file to make the setup easier and ensure that the content of the video stream can be controlled accurately. A chain of stream processors is set up, each streaming the content further from the video file to the video player while applying a specific function on the stream. The stream processor is an FFMPEG (`https://www.ffmpeg.org`) process augmented and configured to execute the chosen function.

4.3.5 Detailed Description of each Component

4.3.5.1 Encryption

The encryption stream processor takes as input:

- an input stream (e.g. a file or URL)
- an FE encryption key
- a symmetric encryption key
- an output stream configuration (e.g. an URL)

The processor executes the following steps in order:

1. read the input stream
2. demux the stream
3. extract video data
4. extract motion vector data from the video data
5. run the motion vector data through the FE encryption function configured with the encryption key
6. encode the modified motion vector data back into the video sample
7. encrypt audio and video data streams at the NALU level using the symmetric encryption method, excluding the motion vectors
8. remux the stream
9. output the stream

4.3.5.2 Local Decision Making

The local decision making stream processor takes as input:

- an input stream (e.g. an URL)
- a FE data extraction key
- an output stream configuration (e.g. an URL)

The processor executes the following steps in order:

1. read the input stream
2. demux the stream

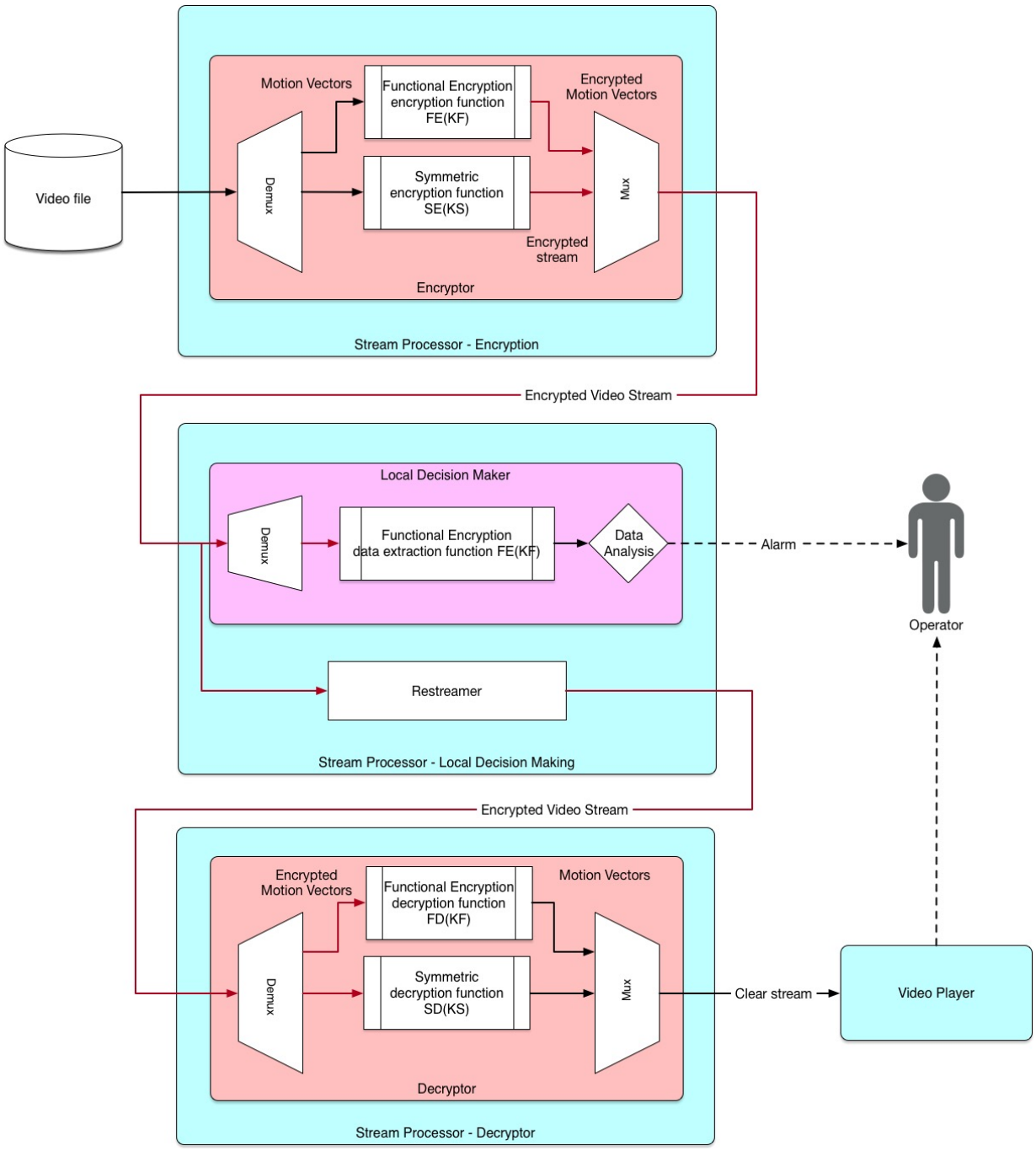

Figure 13: The physical and logical architecture of the prototype

3. extract motion vector data from the data stream
4. run the motion vector data through the FE data extraction function configured with the data extraction key
5. analyze the result of the computation and generate an alert for the operator if conditions are met

The processor also forwards the encrypted stream to the configured output.

4.3.5.3 Decryption The decryption stream processor takes as input:

- an input stream (e.g. an URL)
- an FE decryption key
- a symmetric encryption key
- an output stream configuration (e.g. an URL)

The processor executes the following steps in order:

1. read the input stream
2. demux the stream
3. extract motion vector data from the data stream
4. run the motion vector data through the FE decryption function configured with the decryption key
5. encode the corrected motion vector data back into the stream
6. decrypt audio and video data streams at the NALU level using the symmetric encryption method
7. remux the stream
8. output the stream

4.3.5.4 Video Player The video player is any standard player capable of playing the original video stream, typically the VideoLan (VLC) player or any comparable player.

4.3.6 External Interfaces

4.3.6.1 User interfaces There are two user interfaces in the prototype

- The Video Player interface. It allows playing back the video stream and is the unmodified UI of the chosen video player.
- The Local Decision Making interface. In its simplest form, this is the output of the stream processor's log, which can be seen in the terminal window where the process has been launched. Alerts are output in the log with a clear marking and possibly a sound to attract attention. In a future prototype, the process could output its alerts to a dedicated alert panel, or forward them to a control center.

4.3.6.2 Software interfaces The Functional Encryption functions are called by the stream processors using a C API.
Ideally the Functional Encryption library API should be similar to the following:

```
/*** Encryption ***/
/*
 * Functional Encryption encrypt context create function.
 * Creates an encryption context with a key and a data size.
 * @param key_data the key that will be used to encrypt the data. The format is
 *  private
 * @param key_data_length the size of key_data
 * @param data_length the length of the data arrays that can be encrypted with
 * the context
 * @return an encryption context that can be used in fe_encrypt calls. Memory
 *  management is the responsability of the library.
 *  See also fe_release_encrypt_context()
 */
void *fe_create_encrypt_context(void *key_data, uint16_t key_data_length,
                                uint16_t array_length);

/*
 * Functional Encryption encryption context release function.
 * Releases a context previously created with fe_create_encrypt_context().
 * After calling this function, the context is invalid and subsequent calls to
 *  fe_encrypt() with this context shall fail.
 */
void fe_release_encrypt_context(void *context);

/*
 * Functional Encryption encryption function.
 * Takes an array of clear 16 bits signed integers (plain text) and produces
 *  an array of encrypted 16 bits integers of equal size (cipher text).
 * @param context the encryption context returned by fe_create_encrypt_context()
 * @param data_in data to encrypt: an array of int16_t (16 bits signed integer).
 *  The size of the array is specified by the parameter 'array_length' in the
 *  call to fe_create_encrypt_context().
 * @param data_out Array of int16_t to hold the encrypted data. Memory is
 *  managed by the caller, the data_out array must be big enough to hold
 *  'array_length' int16_t values.
 * @return on success, the count of values encrypted (== array_length),
 *  on failure, -1.
 */
long fe_encrypt(void *context, int16_t *data_in, int16_t *data_out);


/*** Decryption ***/
/*
 * Functional Encryption decrypt context create function.
 * Creates a decryption context with a key and a data size.
 * @param key_data the key that will be used to decrypt the data. The format is
 *  private
 * @param key_data_length the size of key_data
 * @param data_length the length of the data arrays that can be decrypted with
 *  the context
 * @return a decryption context that can be used in fe_decrypt calls. Memory
 *  management is the responsability of the library.
 *  See also fe_release_decrypt_context()
 */
void *fe_create_decrypt_context(void *key_data, uint16_t key_data_length,
```

```
                                uint16_t array_length);

/*
 * Functional Encryption decryption context release function.
 * Releases a context previously created with fe_create_decrypt_context().
 *  After calling this function, the context is invalid and subsequent calls to
 *  fe_decrypt() with this context shall fail.
 */
void fe_release_decrypt_context(void *context);

/*
 * Functional Encryption decryption function.
 * Takes an array of encrypted 16 bits signed integers (cipher text) and
 *  produces an array of decrypted 16 bits integers of equal size (plain text).
 *
 * @param context the decryption context returned by fe_create_decrypt_context()
 * @param data_in data to decrypt: an array of int16_t (16 bits signed integer).
 *  The size of the array is specified by the parameter 'array_length' in the
 *  call to fe_create_decrypt_context().
 * @param data_out Array of int16_t to hold the encrypted data. Memory is
 *  managed by the caller, the data_out array must be big enough to hold
 *  'array_length' int16_t values.
 * @return on success, the count of values encrypted (== array_length),
 *  on failure, -1.
 */
long fe_decrypt(void *context, int16_t *data_in, int16_t *data_out);

/*** Data extraction ***/
/*
 * Functional Encryption data extraction context create function.
 * Creates a data extraction context with an operation name, a key and a data
 *  size.
 * @param operation null-terminated name of the data extraction operation
 *  function to be performed with this context.
 * Possible values for this parameter depend on the FE library version. Typical
 *  value could include 'average', 'sum', etc.
 * @param key_data the key that will be used to extract the data. The format is
 *  private. This is *not* the same key as the decryption key!
 * @param key_data_length the size of key_data
 * @param data_length the length of the data arrays that can be processed with
 *  the context
 * @return a data extraction context that can be used in fe_extract calls. Memory
 *  management is the responsability of the library.
 *  See also fe_release_extract_context()
 */
void *fe_create_extract_context(char *operation, void *key_data,
                                uint16_t key_data_length, uint16_t array_length);

/*
 * Functional Encryption data extraction context release function.
 * Releases a context previously created with fe_create_extract_context(). After
 *  calling this function, the context is invalid and subsequent calls to
 *  fe_extract() with this context shall fail.
 */
void fe_release_extract_context(void *context);

/*
 * Functional Encryption data extraction function.
 * Takes an array of encrypted 16 bits signed integers (cipher text) and
```

```
115  *  produces a single 16 bits signed integer values
116  * @param context the data extraction context returned
117  *  by fe_create_decrypt_context()
118  * @param data_in data to process: an array of int16_t (16 bits signed integer).
119  *  The size of the array is specified by the parameter 'array_length' in the
120  *  call to fe_create_extract_context().
121  * @return the computed value on data_in
122  */
123 int16_t fe_extract(void *context, int16_t *data_in);
```

4.3.7 Documentation

All necessary documentation is provided to build, install and operate the prototype, along with test data.

4.4 Operating Environment

4.4.1 Operating System

The prototype is built to run on Unix-like systems (e.g. Linux, macOS). For other systems, use of a Docker image is recommended.

4.4.2 Library Support

The prototype will mainly make use of the FFMPEG library, as well as the usual system libraries FFMPEG relies on.

4.4.3 External Network

The prototype shall operate entirely on a local network and will not require access to an external network except for initial setup.

4.4.4 User Interfaces

User interfaces are described above and are limited to command-line interfaces and the video player.

4.5 Evaluation Metrics

In this section we give some metrics for evaluating the success of the IoT video encryption use-case.

4.5.1 Functionality

The functionality of our IoT video encryption prototype and motion detection system should be experimentally verified in a controlled environment and may be demonstrated publicly if we consider our prototype sufficiently resilient and mature for such demonstration.
The functionalities described in paragraph 4.2.1.1 are to be implemented in such a way that it allows attaining the goal stated in the same paragraph, while leveraging the FE scheme developed in this project.

4.5.2 Performance

Typically, video performance is evaluated by comparing the maximal cumulated bandwidth we can attain with multiple devices streaming encrypted content to a gateway without disturbing its operations and while ensuring quality of service is retained.
This metric is expressed as a maximal cumulated bandwidth in MBps or, if we only concern ourselves with the performance of the cryptographic scheme in use, in term of "number of 16-bit integer values processed by time unit".

4.5.3 Security

The prototype has no security requirement beside the end-to-end encryption of the content using a cryptosystem developed by FENTEC to allow partial leakage at the gateway level of the data required to enable the proper operation of the gateway.
To evaluate the security of the end-to-end FE encryption scheme we use, we will rely on the security analysis done by our academic partners, depending on the security parameter λ we will fix for our usage. Different levels of security may be attained by using different values for λ, depending on their impact on the performance of the encryption and decryption processes, and on the size of the keys and ciphertexts.

4.5.4 Feasibility

The feasibility of the techniques used will be validated by building a successful prototype. To a certain extent this project is speculative and may lead to negative results in some cases. In any case, whether we can build the prototype or not, we need to assess the feasibility of the method compared to existing methods to achieve the same sort of privacy goals and security level.

4.5.5 Reliability

The reliability of our prototype can be evaluated by running a full-fledged demo in a controlled environment at modern video streaming speeds for an extensive period of time and monitoring its uptime and other metrics.

4.5.6 Legal aspects

Evaluating the legal aspects of our prototype is not planned, as we are going to process only test data in a controlled environment.

4.5.7 Ethical aspects

Per se, the ethical implications of a privacy-preserving motion detection method seem to be entirely dependent of its eventual usage and as such should not be an issue for our prototyping phase. Shall ethical issues arise, they would be studied and evaluated with due diligence by us and our partners with appropriate experience.

4.6 Additional Requirements

4.6.1 Performance

Since we expect to be able to process the motion vectors in near real-time, it means in the case of a full HD stream of size 1920×1080 pixels with 8×8 macro-blocks that we would need to be able to perform

the extraction (or "evaluation") and decryption operations on a array of 64800 integer elements (encoded as 16-bit integers) in 20ms or less. This means an extraction and decryption rate of about 50 MBps.
In case of a 4K stream, the rate linearly increases to 200 MBps.
Such speed might require a specialized hardware accelerator in order to be able to perform real-time operations.
Note that the gateway is expected to process multiple streams at the same time, which means that ideally the data extraction process should be as fast as possible. The figures above are representative of a single stream.

4.6.2 Security

A full Security audit for the prototype is outwith the scope of the FENTEC project. However, we should review our methods and implementations, particularly with respect to the security and privacy of the unencrypted video data. Although the data considered by our prototype is not critical (we are only processing test data in a fully controlled environment), we should consider that our methods may be applied in the future to more sensitive data.

4.6.3 Quality

Since we are not generating production quality code and systems, we are not really constrained by quality control issues, however assuring a good code quality might be good to ensure proper adoption and maintainability of the proposed system.

4.7 Outstanding Issues

The motion vectors extracted from the video streams are in a quantity dependent on the size of the frames and on the size of the macro-block.
The amount of motion vectors is therefore fixed for a given stream, and can be seen as a fixed size table of integers, encoded in 16 bits, depending on the size of the image and the size of the macro-blocks. For instance, in the case of a full HD stream, we have an image size of 1920×1080 pixels and would typically have 8×8 macro-blocks, which means a total of 32400 macro-blocks per frame, each having 1 motion-vector of size 2: a horizontal motion value and a vertical one, for a total of 64800 integer values per frame, split into 2 tables of each 32400 elements.
Since the motion-vectors are 15-bit **signed** integers, and since we cannot replace them with something larger than that, we need a means to provide a table of 16-bit elements, that would get functionally encrypted into a new table of 16-bit elements that we could substitute with the unencrypted value, without using a different number of elements than originally.
Adding extra information, such as an IV or a random r value might be done, but not for each frame, and such information should ideally be derived from the key in a deterministic fashion to avoid any extra data in the ciphertext.

4.8 Meeting Requirements

This section should explain how the foregoing specification will allow the design to meet the requirements listed in D3.1[4].

4.8.1 Non-functional Requirements

IoT.01 End-to-end encryption

The video data shall be encrypted in a way that does only allow the camera where it originates and the back-end system where it is sent to gain any information about the video content, besides the information that can be extracted at the gateway-level thanks to the use of functional encryption.

IoT.02 IND-CPA

An adversary shall not be able to decrypt anything or gain any information out of a chosen plaintext attack, since it might be possible to show known images to cameras, and since the cameras being edge-devices, they cannot be necessarily trusted at all time.

IoT.03 Adaptable security level

Depending on the available bandwidth or different factors, the frame size of the video stream might change and as such the amount of motion-vectors that are to be encrypted might change as well. It should be possible to instantiate the scheme using different parameters. Shall security be impacted, this should be clear and the choice of a given security level should be possible for any given quantity of motion-vectors, or shall this not be possible, the minimal quantity should be small enough to fit common resolution size.

IoT.04 Leakage resistance

No useful data shall be extracted from the encrypted video stream without the appropriate secret keys.

IoT.05 No privacy sensitive data during prototype

We shall only instantiate our prototype in controlled environments, and as such no sensitive data must be used during the prototyping, testing and demonstration phases of the project.

IoT.06 Fast decision time in gateway

The gateway needs to perform its local decision making in near real-time, as outlined in section 4.7.

Note that this is impacted by the extraction (or evaluation) of the data from the encrypted streams. Ideally switching delay between streams should be kept under 500ms.

IoT.07 High performance gateway

We might require specialized hardware in order to reach the performance requirement outlined in section 4.7, if we want to be able to process multiple streams at the same time at the gateway level.

IoT.08 Overall latency impact limit

In order to be able to perform all intended operations at the camera, gateway and back-end levels, we expect the extraction and decryption to be done in near real-time, as well as the encryption. This means that the latency introduced by the functional encryption scheme should not increase over time as more data are being processed.

4.8.2 Functional Requirements

IoT.09 Configurability

Since the gateways have to perform local decision making based on one or more streams, being able to tweak the threshold at which the evaluation of the function using functional encryption is

required. But this might also be done by simply giving a (normalized) amount corresponding to the estimated motion detected in the currently processed motion-vectors. This could potentially leak the amount of motion detected to the gateway but would enable a more fine-grained local decision making at the gateway level by allowing the gateway to perform statistical analysis over the amount of motion detected in consecutive frames, which directly translates into the motion detected over a given amount of time.

IoT.10 Secret threshold value

Encryption should not depend on the threshold value used by the gateway to trigger an alarm. In other words the edge-devices should not be encrypting the motion-vector differently, depending on the currently used threshold value at the gateway-level. The gateway should be able to silently change its motion-detection threshold value without having to signal it to the cameras. This implies that the FE scheme evaluation at the gateway-level should ideally be partly configurable, or should return an absolute value, letting the gateway act on its own based on the actual value returned after FE evaluation.

IoT.11 C API

As mentioned in section 4.3.3, we must be able to use C bindings to use the FENTEC library.

4.9 Conclusions

This technical specification of a video surveillance system is complete enough to warrant starting the implementation work. The functionality has been described in detail, the cryptographic needs have been addressed and some discussion about the required performance levels has been carried out. As with all of the use-cases, there still remain some questions to be answered, in this case particularly the hardware requirements but these can only be examined in detail during development. The criteria for success are clear enough and meeting the requirements analysis should also be feasible. This use-case may now proceed to the implementation phase.

5 Role of the FENTEC library in the technical specifications

This section gives some comments on the relationship between the proposed FENTEC library to be developed as part of WP6 and the technical specifications presented in this deliverable.
The FENTEC library will provide a broad set of functional encryption schemes in multiple programming languages. The implementation of the FENTEC library is the responsibility of WP6. Use-cases developed in WP7 will use the FENTEC library as the main building block to achieve privacy-enhanced applications. The requirements of the use-cases differ in terms of cryptographic schemes and programming languages. Furthermore, the WALLIX use-case has an additional requirement (WA.08) to provide the FENTEC library under an acceptable license due to Awless being open source and licensed under the Apache License Version 2.0.
WP6 will release functional encryption schemes in the following programming languages:

Golang Golang is required by the WALLIX use-case as the encryption operation will be integrated into the Awless client written in Golang. The first version of the library (named GoFE) has already been released and is available on FENTEC Github account: `https://github.com/fentec-project/gofe`. Furthermore, a demonstration of how to build a machine learning classifier on top of encrypted data is given in `https://github.com/fentec-project/fe-ml-example`.

The Golang library has been licensed under Apache License Version 2.0 to meet the requirement WA.08. Note also that the GoFE library is written entirely in native Golang and does not include any third party code not provided as part of the Golang language itself. This means that there are no additional licensing problems associated with such code.

C Functional encryption schemes in C are required for both the ATOS and Kudelski use-cases. Furthermore, C is still the most commonly used language in the IoT world and the Kudelski use-case could naturally be extended towards various IoT use-cases.

Rust A relatively young programming language designed in a way that allows compile-time detection of data races and guarantees memory safety. Moreover, Rust is a cross-platform language with a minimal runtime and it is completely interoperable with C due to its straightforward and efficient foreign function interface, allowing C code to be called from Rust and vice versa. This implies a great potential for the usage of the language in the world of IoT, as there are not many fast as well as safe alternatives in the field. So far Rust can run on the ARM Cortex-M One family of microcontrollers and there is great effort in the community towards making Rust a real competitor to C/C++ in the area of IoT and bare-metal embedded applications.

Python Python enables rapid prototyping and the first implementations of functional encryption schemes in FENTEC have been coded using Python. Later, this code was translated into Golang, C and Rust.

The number of actual code repositories (libraries) will in fact be more than four. The reason is that different functional encryption schemes have been designed for different purposes. While the WALLIX and Kudelski use-cases require inner-product functional encryption schemes, the ATOS use-case requires ABE schemes. Inner-product and ABE schemes differ heavily and do not have a lot of functionality in common. Obviously, the same cryptographic primitives are used in both types of schemes (such as bilinear pairings and hash functions) but these are exported from the FENTEC libraries (or built-in to the programming language) and are not implemented by the FENTEC consortium. If there is enough common functionality between inner-product and ABE schemes they will be evaluated later in the project and a

decision will be made whether to provide this functionality in a separate library which can be included in both inner-product and ABE libraries.
To improve the performance of the implemented schemes, various algorithms for computing discrete logarithms (baby-step giant-step, Pollard rho, Pollard kangaroo) have been provided in all four programming languages.
Currently, the following schemes are implemented:

- Decisional Diffie-Hellman (DDH) version of [1]
- Learning With Errors (LWE) version of [1]
- Ring-LWE version of [1]
- DDH version of [2]
- LWE version of [2]
- Ring-LWE version of [2]
- Scheme for quadratic multi-variate polynomials [6]

While no use-case required post-quantum safe schemes, the library provides support for LWE and Ring-LWE schemes (most notably Gaussian samplers and matrix operations) which are lattices based on cryptographic primitives providing post-quantum security.

6 Conclusion

In this document we have presented three technical specifications; the ATOS use-case which proposes a new digital currency with auditing capabilities, the WALLIX use-case which describes a new anonymous method for performing web analytics and one for the Kudelski use-case which proposes the use of FE to generate additional information streams from encrypted video channels.

The ATOS use-case specification gives a breakdown of the digital currency and its attendant auditing system into detailed schemas which indicate the complete set of exchanges needed to implement the system securely. It is also shown how the deployment of attribute-based encryption methods is integrated into this process enabling the auditability aspect of the design. In fact, two scenarios are presented each of which requires a different type of ABE and there is a significant effort required to initialize the process which requires a trusted entity. All of these components are detailed in the specification. The development methodology proposed is the Iterative/Incremental method well suited to this kind of design. Although there are some questions about performance and data requirements, particularly key sizes, this use-case is now ready to proceed to the implementation phase.

The WALLIX use-case specification details the breakdown into a small number of components on two platforms but there are some final decisions to be made on the placement of some of the components within the architecture. Timing constraints for a web analytics polls are actually quite relaxed, since it is estimated that about a day to completion should be sufficient for most cases of the type proposed for this project. Other factors may become relevant should this method prove successful and be used in a commercial context, for example the cost of the processing resources on cloud platforms may become significant. The main point for this document, however, is that the method should be capable of being implemented using this specification as a reference in such a way that we can make meaningful predictions as to the performance, cost and security of the FE technology in this context. Certainly, we have addressed the main obstacles to beginning the development work, we have identified the resources that will be necessary for successful completion, suggested a methodology for managing the work and have defined some basic citeria for success. We can state that the WALLIX use-case can now proceed to the implementation phase.

The Kudelski use-case describes the characteristics of the proposed privacy-preserving video stream processing system. The functionality is described in detail and a method of construction involving three phases is given. Since near real-time processing speeds are required by this use-case plus the fact that FE processing requirements can be quite heavy, the hardware requirements may require upgrading during development in order to achieve the necessary performance. The software environment for this case is relatively standard, being based upon C and C++ and the proposed methodology is also standard, being an incremental method. The component breakdown is quite simple despite the relative complexity of the architecture which may require hardware speedup at more than one component. Again, however, this specification is adequate for allowing the implementation work to proceed. All of the prerequisites in terms of library support, architectural decisions and other resource requirements have been met.

The FENTEC library has already been developed to the point where it can be used to support all three use-cases in their initial development. Further work may be required as the prototypes proceed but this will only become apparent when the need arises within each use-case. In particular WALLIX's licensing requirements have already been met.

7 Next steps

For each use-case, the development work can begin although this specification does not give any indications as to the planning or scheduling of the tasks required to realize each case. The following tasks; Task 7.2 for the anonymous data collection type (WALLIX), Task 7.3 for the privacy-enhanced digital currency (ATOS) and Task 7.4 for the IoT surveillance camera (KUD) all begin in FENTEC project M9 immediately after the publication of this deliverable.

One small point to bear in mind for all use-cases is that although Task 7.5 Prototype testing and Task 7.6 Prototype performance evaluation are not due to start until M23/M24 these tasks should be borne in mind throughout the prototype development by all parties. Some effort has already been made in suggesting ongoing testing and also some goals for performance targets are already in this specification but these evaluation tasks will be made easier if their objectives are taken into account from the start of development.

References

[1] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, pages 733–751, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[2] Shweta Agrawal, Benoit Libert, and Damien Stehle. Fully secure functional encryption for inner products, from standard assumptions. Cryptology ePrint Archive, Report 2015/608, 2015. `https://eprint.iacr.org/2015/608`.

[3] Jeffrey Burdges, Florian Dold, Christian Grothoff, and Marcello Stanisci. Enabling secure web payments with gnu taler. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 251–270. Springer, 2016.

[4] FENTEC. D3.1 technical requirement report analysis. Technical report, European Commission, 2018.

[5] golang.org. Effective go guidelines. `https://golang.org/doc/effective_go.html`. Accessed: 2018-07-05.

[6] Edouard Dufour Sans, Romain Gay, and David Pointcheval. Reading in the dark: Classifying encrypted digits with functional encryption. *IACR Cryptology ePrint Archive*, 2018:206, 2018.