



## D7.3 First Version of the Truly Anonymous Data Collection Prototype

Document Identification			
Status	Final	Due Date	31/10/2019
Version	1.0	Submission Date	31/10/2019

Related WP	WP7	Document Reference	D7.3
Related Deliverable(s)	D7.1, D7.2, D7.5, D7.7	Dissemination Level(*)	PU
Lead Participant	WALLIX	Lead Author	Norman Scaife (WALLIX)
Contributors	WALLIX	Reviewers	Miguel Angel Mateo Montero (ATOS) Azam Soleimanian (ENS)

Keywords:
Preliminary Version Demonstrator Report, technical specification, use-case, web analytics, 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.

This document and its content are the property of the FENTEC consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the FENTEC consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the FENTEC Partners.

Each FENTEC Partner may use this document in conformity with the FENTEC consortium Grant Agreement provisions.

(\*) 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
Norman Scaife	WALLIX

Document History			
Version	Date	Change editors	Changes
0.1	17/07/2019	WALLIX	First content
0.2	30/07/2019	WALLIX	Published as templates for D7.5 and D7.7
0.3	01/10/2019	WALLIX	First three sections completed
0.4	07/10/2019	WALLIX	Platform section added
0.5	10/10/2019	WALLIX	Software and Demonstrator sections added
0.6	11/10/2019	WALLIX	First complete draft
0.7	14/10/2019	WALLIX	Update after readthrough
1.0	28/10/2019	WALLIX	Update after internal review, submitted version

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable Leader	Norman Scaife (WALLIX)	31/10/2019
Technical Manager	Michel Abdalla (ENS)	31/10/2019
Quality Manager	Diego Esteban (ATOS)	31/10/2019
Project Coordinator	Francisco Gala (ATOS)	31/10/2019

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	1 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

# 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 of the Document . . . . .	6
2 Data Collection Prototype Demonstrator . . . . .	7
2.1 Introduction . . . . .	7
2.2 Cryptographic Protocol . . . . .	8
2.3 Polling Process . . . . .	8
2.4 Platform . . . . .	9
2.5 Software . . . . .	11
2.6 Demonstrator . . . . .	22
2.7 Performance and Security . . . . .	26
3 Conclusion . . . . .	27
4 Next steps . . . . .	28
References . . . . .	29

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	2 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

---

## List of Figures

---

1	Web Analytics software dependencies between components . . . . .	10
2	Architecture of the FENTEC Web Analytics Demo . . . . .	23

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	3 of 29				
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final

---

# List of Acronyms

---

Acronym	Description
API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command Line Interface
DMCFE	Decentralized Multi-Client Functional Encryption
FE	Functional Encryption
REST	REpresentational State Transfer

---

# Executive Summary

---

This deliverable describes the WALLIX Web Analytics use case prototype at the end of the second year of the project. The cryptographic protocol is described and the platform upon which the prototype is based is presented. Some notes about the software implementation are provided and a demonstration system for proving the viability of the technique is shown. Although this is a preliminary prototype we also give some brief indication of the performance of this early version.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	5 of 29				
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final

# 1 Introduction

Following the Requirements Analysis D3.1[3] and the Initial Technical Specification D7.1[5] we now present the initial work into the development of the Wallix truly anonymous web analytics use-case prototype.

The goals of this document are to describe, for the Wallix use-case:

- how the prerequisites for prototype development have been met,
- the current state of development of the use-case in terms of functionality, and
- a preliminary analysis of the security and performance of the prototype.

The Initial Technical Specification D7.1[5] defined the required functionality for the prototype and how to achieve that using existing tools plus cryptographic protocols defined by the academic partners and the FENTEC library implemented by Xlab. There are also some tentative performance targets but since these proved to be difficult to quantify they are only general guidelines.

The overall goal of this phase of the project (Task 7.2) is to produce a working prototype in order to assess the suitability and efficacy of the cryptographic protocol and its ability to meet the requirements of the application. To this end, in this document, we concentrate on describing how our work meets these requirements. This will then lead into the final phase of the project where we concentrate more upon the performance and security aspects of the use-case while maintaining the functional requirements.

This deliverable is produced concurrently with the development of the prototypes for the other two use-cases and leads into the reports on the final prototypes plus reports on performance and security:

- D7.4 Final version of the truly anonymous data collection prototype (M33)
- D7.5 First version of the privacy enhanced digital currency prototype (M22)
- D7.6 Final version of the privacy enhanced digital currency prototype (M33)
- D7.7 First version of the IoT key distribution prototype (M22)
- D7.8 Final version of the IoT key distribution prototype (M33)
- D7.9 First test report of the FENTEC prototypes (M23)
- D7.11 Performance report for FENTEC prototypes after first cycle (M24)

## 1.1 Structure of the Document

**section 2.2** describes the cryptographic protocol used for the prototype.

**section 2.3** specifies the physical phases of the polling process.

**section 2.4** gives a brief description of the platform which has been developed.

**section 2.5** outlines the structure of the software and the method of operation.

**section 2.6** summarises what the demonstrator version of the use case application attempts to prove.

**section 2.7** gives some brief notes on its current performance.

**section 3** concludes with a brief statement about conformance with project goals.

**section 4** gives some brief indications for the next steps in the project.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	6 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

## 2 Data Collection Prototype Demonstrator

### 2.1 Introduction

The WALLIX Web Analytics use case is intended to allow the gathering of statistical information in such a way that the participants' data remains anonymous to the entity conducting the analytics and also from each other.

This is the scenario in which Functional Encryption (FE) is ideal since it allows the application of simple functions to encrypted data. With the *proviso* that the participants in the statistical analysis (which we call a poll) are prepared to send their data in encrypted form for this type of analysis we are able to provide simple statistical analysis of the participants' data.

For the FENTEC use case the participants are WALLIX Awless clients. Awless is a command line tool for managing Amazon AWS accounts. Statistical analysis was originally implemented using cleartext transmissions to a central server hosted at WALLIX. However, the Awless user community refused to countenance the sending of any part of their Amazon configuration data to a third party, even a trusted one such as WALLIX. As a result, the statistical analysis was removed from public versions of the tool.

Further analysis of the Awless code combined with a review of the existing cryptographic protocols available, explained in FENTEC deliverables D4.1 [4] and D4.2 [6], led to the requirements analysis in D3.1 [3] and the technical specification for the use case prototype in D7.1 [5]. To summarize this work, we can state that the requirements of the system are mainly that all encryption is carried out on the clients which means a distributed key system. At the time of writing the specification, there existed one cryptographic protocol which suited these requirements, an inner product distributed multi-client functional encryption protocol (DMCFE [1]). Note that other protocols have become available subsequently, for example the DSum [2] protocol. For this first phase of the project, however, we have concentrated upon building a working prototype irrespective of the protocol used. In the latter stages of the FENTEC project we will focus more upon the performance and security and will consider improved protocols as part of that work. However, the requirement in this protocol for redistribution of encrypted data and the generation of shared keys plus the fact that we are fixed with an internet-based client/server model means that we cannot eliminate the need for a server altogether. Hence we propose a cryptographically neutral "Proxy" server whose only purpose is to enable the redistribution of data between clients. Subsequently, it has become host to other necessary elements in the prototype which we will discuss later.

Given this design we have constructed a prototype of a platform for conducting web analytics over a set of Awless clients. The additional requirements for management of the entire polling process have been worked out in a preliminary manner by using a mixture of automatic and manual control of the process. The proxy server has been implemented and is based upon the core of the WALLIX DataPeps end-to-end encryption service. This was chosen partly because it has all of the required support to run an API-based service on the internet and partly because the ultimate intention at WALLIX for this work is to incorporate the web analytics into DataPeps itself.

Demonstrating the prototype presents some problems in that the clients need a working version of Awless linked to an Amazon AWS account. This is difficult to arrange in practice but we have circumvented the problem by implementing a simple web server which allows actions to be performed upon a preset Awless installation. This allows participants in the poll to access AWS through the website and contribute to a predefined poll on a proxy server also built into the website host.

For the most part, the prototype demonstrator presented here conforms almost exactly to the specification presented in the technical specification document D7.1 [5]. The architecture is fixed for our application in any case and so has not changed at all since the specification was written. For the software components, we have made some simplifications for the purposes of demonstration and added some minor new components to enable live demonstration of the application in a timely manner. For clarity, we here assume familiarity

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	7 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final



with the specification and only point out differences as they occur.

## 2.2 Cryptographic Protocol

The DMCFE protocol [1] has four functions, which we summarize briefly here:

- **Setup:** Starting from a type 3 pairing  $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  provided with hash functions  $\mathcal{H}$  and  $\widetilde{\mathcal{H}}$  onto  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , respectively, each client randomly generates  $s_i$  and  $\widetilde{s}_i$ , also respectively. Then after redistribution of the  $\widetilde{s}_i$  values we compute  $\widetilde{dk}_1 = \sum_i \widetilde{s}_i$  which with the pairing definition and hash functions becomes the master public key  $mpk$ . The encryption key is  $ek_i = s_i$  and the secret key is  $sk_i = (s_i, \widetilde{s}_i)$ .
- **Encryption:**  $\text{Encrypt}(ek_i, x_i, l)$ , for encryption key  $ek_i (= s_i)$ , data value  $x_i$  and label  $l$ , returns encrypted data  $C_{l,i} = \mathcal{H}(l)^{s_i} \cdot g^{x_i}$
- **DKeyGen:**  $\text{DKeyGen}((sk_i)_i, \mathbf{y})$ , for secret keys  $(sk_i)_i$  and weighting vector  $\mathbf{y}$ , we compute distributed key  $\widetilde{C}_{\mathbf{y},i} = \widetilde{\mathcal{H}}(\mathbf{y})^{\widetilde{s}_i} \cdot \widetilde{g}^{s_i \cdot y_i}$ . We then have to redistribute  $\widetilde{C}_{\mathbf{y},i}$ .
- **Decryption:** For  $\text{Decrypt}(dk, l, C)$ , for distributed key  $dk$ , label  $l$  and encrypted data  $C$ , we first have to compute  $dk_{\mathbf{y}} = \prod_i \widetilde{C}_{\mathbf{y},i} \times \widetilde{\mathcal{H}}(\mathbf{y})^{-\widetilde{dk}_1}$ . From  $\mathbf{C} = (C_i)_i$  we can compute  $g_T^\alpha = e(\prod_i C_i^{y_i}, \widetilde{g}) / e(\mathcal{H}(l), dk)$  where  $e(\cdot, \cdot)$  is the pairing relation. Finally, we have to compute a discrete logarithm to access the result  $\alpha = \langle \mathbf{x}, \mathbf{y} \rangle$ .

This description omits some of the mathematical precision required to accurately describe the protocol but it contains the necessary steps and computations to explain how we have mapped this protocol to our architecture.

Any realistic implementation has to take into account the architecture of our platform but we can state a couple of simple principles here. Firstly, redistribution must occur in two phases, a collection phase where the clients send individual data to the proxy server and a distribution phase where the combined data are returned to the clients. Secondly, the data flow in the protocol is sufficiently flexible that we have some leeway to map individual computations to different phases of the poll to allow maximum efficiency. For example, we can combine the distribution phase of a previous operation with the collection phase of the following one. The mapping from protocol computations to poll phases is not necessarily one cryptographic function per poll phase.

## 2.3 Polling Process

The necessary, physical phases of the type of internet poll we wish to conduct are as follows:

**Configuration** Required to tell the Awless clients that they are participating in a poll. The parameters include a description of which statistical data to collect, the cryptographic parameters and the method of controlling the poll, for example the timing parameters such as the start and end times for the phases. We have proposed a configuration server for the poll in order to keep the proxy server as clean as possible (without any cryptographic data on it), but for the moment we simply combine this function with the proxy server.

**Statistics gathering** During the statistics collection phase the clients operate independently of any interference with the poll server other than to periodically check whether the gathering phase has ended. This is to minimize the impact of the poll upon Awless users. Since the values we are collecting are very simple there should be almost no effect upon the performance of Awless.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	8 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>		1.0
		<b>Status:</b>	Final

**Selection** Depending upon circumstances, we may wish to perform sub-sampling upon our collected data. Within the restrictions of anonymity which is the principal selling point of our technology, it is difficult to perform realistic outlier detection using FE. Initially, we restrict ourselves to simple missing data removal (ie. deselecting clients which have no data) by cleartext negotiation over the network. We make the assumption that it is acceptable to our clients to provide a simple flag in cleartext which indicates whether data is present or not. This may help in the first place with missing data due to, for example computer crashes or the desire by the client to withdraw from the poll but does nothing for the situation where a malicious client provides false data. For the remainder of the project we may consider more complex schemes implemented using FE.

**Data aggregation** Once we have decided upon which clients will contribute to the final analysis we have to perform the encrypted data and shared key redistribution. In fact, if we are performing sampling we also have to defer the Setup computation for the cryptography to this point since all multi-client FE protocols require a fixed number of participants. Once this phase is complete we then have the ability to decrypt the result.

**Decryption and dissemination of results** The decryption can occur anywhere that the combined data and keys are stored which initially will be the proxy server. It is then up to that server to decide who has access to the results although since the clients have all participated in the Setup phase they only need the combined values to do the decryption themselves. Due to the potential sensitivity of the results (particularly in the case of degenerate result sets, for example all of the clients return zero values except one which means that the results are actually the data provided by that client) we need to be careful how we distribute access to the results. This may mean additional symmetric encryption.

## 2.4 Platform

As already mentioned, our architecture is simply the proxy server to control the analytics process and the Awless clients as the system we wish to analyze. In order to implement the full functionality of the web analytics, however, there are several components built into both the proxy server and the clients. These were explained in D7.1 and are reproduced here in Figure 1. Since this design has not been altered for the purposes of the demonstrator we can use it as the basis for our description in this deliverable (with the exception that the proxy server (WP2) has been implemented in the same component as the configuration host (W1a)).

The FENTEC library components (W4a, W4b, W4d and W4e) have already been reported in FENTEC deliverable D6.2 [7] (Section 6.1). They correspond exactly to the functions in the DMCFE protocol we use with an additional component (W4c) to act as a placeholder for the data redistribution function. Note that the library functions provided in the FENTEC library have been refactored slightly from the original definition of the protocol for efficiency reasons but still correspond to the functions we have described in this document.

### 2.4.1 Proxy server

The proxy server (W2) hosts an API originally intended to be one method for controlling the DataPeps platform. The API endpoints have been rewritten to manage the web analytics platform instead. These endpoints are divided into two types; those to create and delete the poll data itself and those needed to enable the Awless clients to perform data redistribution. There are also some auxiliary endpoints which implement monitoring of the poll progress.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	9 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

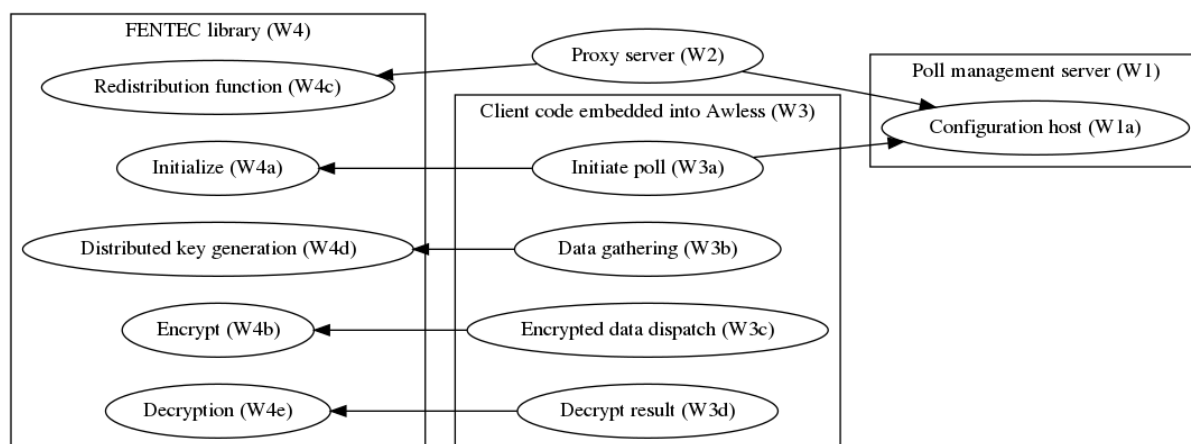


Figure 1: Web Analytics software dependencies between components

The configuration server (W1a) is currently implemented in the same code base as the proxy server and is a simple two-endpoint section of the API, one to create a poll and another to delete a poll.

Note that the proxy server depends upon an SQL server for persistent data. We do not describe the SQL setup here but for the size of data plus the number of accesses per second the SQL server is required to handle it can be implemented with a small single-instance server and is provided as standard on most cloud-based services, for example Amazon RDS. Future deployments of more large-scale web analytics systems may require a larger capacity and speed of persistent data.

## 2.4.2 Awless clients

The Awless platform has a complex structure but our required functionality, in terms of statistics gathering, is quite simple. The main problem here is integrating the functions in the cryptographic protocol into the phases of the polling already described. In architectural terms the entire client code is a single unit called from the main dispatch routine for executing Awless commands. This has been done to keep the FENTEC code as distinct as possible from the main Awless code base. In order to distribute a version of Awless we need to provide a custom version of the tool and by keeping the web analytics code separate it makes it easier to track updates to the main branch of the Awless repository.

For persistent data we use the existing database based on BoltDB<sup>1</sup> which was already used to store the internal configuration for Awless. Note that we have not yet implemented secure access to the web analytics poll management server so poll initiation (W3a) consists simply of the Awless user issuing a command to download the poll configuration data and setting up the web analytics database entries.

The data gathering phase (W3b) is similarly hooked into the command loop and has been designed to be as efficient as possible. We use a very simple indexing system based upon the command line interface which is very quick to execute but somewhat limited in scope. For the purposes of demonstration we restrict the statistics gathering to counting accesses to Amazon AWS endpoints although almost for free we can also count errors generated by these accesses. Configuring error counting and also the variance calculation is managed by appending a suffix (`_err` or `_var`) to the label name associated with each command. This allowed us to avoid propagating database access throughout the code.

The encrypted data dispatch phase (W3c) actually requires multiple accesses to the proxy server and so are spread across multiple command line requests and we lose the one-to-one relationship between Awless

<sup>1</sup><https://github.com/boltdb/bolt>

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	10 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

functionality and web analytics calls. This phase of the process is not particularly optimized at the present time but could possibly be improved upon in future developments.

Finally, the decryption phase (W3d) is implemented in a completely open manner and involves a single access to the proxy server which is linked into an Awless command. This will be made more secure in a future development of the code.

One potential problem is that the clients could be a very heterogeneous collection of computing platforms with widely varying performance. This may mean having a flexible method of managing the polls as they proceed. For the demonstrator, however, the time frame for running a poll is extremely short and so we use a simple, manual method for advancing the poll between the phases.

## 2.5 Software

Both platforms (the DataPeps core of the proxy server and the Awless clients) are written in Golang which at least gives some uniformity to the software.

PHASE	Awless	Request	Response	Endpoint	Action
<ANY>	create <pollConfig>	$\overrightarrow{\text{pollConfig}}$	←	/define	Create poll definition in DB and set poll status to CREATED.
CREATED	<none>			<none>	No poll actions are possible, system must wait until status is switched to CONTRIBUTION either manually or by timer.
CONTRIBUTION	contribute	→	$\overleftarrow{\text{pollConfig}}$	/contribute	Add clientNo to list of contributors in DB, return poll configuration.
STATISTICS	status	→	$\overleftarrow{\text{pollStatus}}$	/status	Return the current status for the poll, phase and numbers of contributors, selected clients and participants
SELECTION	select	$\overrightarrow{\text{hasData}}$	←	/select	Client sends a flag indicating whether data is present or not.
COMPLETED	activate	→	$\overleftarrow{\text{selectNo, T}}$	/activate	Server returns selectNo which is the index into the final selected clients and $dk_1 (= T$ which is in matrix form).
COMPLETED	send	$\overrightarrow{C_{l,i}, C_{y,i}}$	←	/keygen	Client executes Encrypt and DKeyGen and sends results to server which stores them in the DB.
COMPLETED	decrypt	→	$\overleftarrow{\text{pollResults}}$	/decrypt	Server checks cache, if results present then they are returned. Otherwise the shared data are extracted from the DB and the Decrypt function is executed, results are then stored in cache and returned.

Table 1: Poll phases and actions

Table 1 shows the sequence of events occurring during a normal poll (that is, one which runs normally without errors). Note that the cryptography is not actually deployed until after selection despite the fact that the Setup function can be run by all clients (even those not selected) to improve the efficiency. For clients which are not selected the key data is ignored and those clients are returned dummy information during the cryptographic operations. This is to prevent clients from deducing that they have not been selected.

The poll is managed from the Awless command line so a poll is currently created by issuing the command:

```
awless tawa create <url> <name> <poll params>
```

We use the acronym tawa to designate the set of sub-commands which relate to the FENTEC polling system in Awless. This stands for “truly anonymous web analytics”. In fact, we use this acronym throughout all the FENTEC web analytics code. Note that there is a large list of parameters for this command which is

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	11 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

a bit unwieldy for a command line interface. In future, we intend to develop a web-based platform for initiating and controlling polls.

Awless polling commands are divided into two types, one which requires the address of the proxy server as a command line argument, as in the `create` command here, the others only use the poll name and the address of the server is looked up in the the internal database. The Awless commands typically relate to one endpoint on the proxy server, for example the `create` command is simply a direct call to the `/api/v0/tawa/define` endpoint on the server. The poll name is a mandatory argument to all of these commands but have been omitted from Table 1. Likewise, there are other universal values such as the response status which are left out. Some phases, such as `COMPLETED` require a more complicated structure and multiple calls to different proxy server endpoints are required.

The `CREATED` status is really just a pending state allowing the poll to be defined before contacting potential participants. The polling actually starts when the phase is switched to `CONTRIBUTION`. Currently, the whole process is open in that any client connecting to the server will be added to the poll until the preset number of contributors is reached. In future we wish to implement a system whereby the Awless users are sent an access key in order to participate in a poll. Switching to the next phase at a specific time can be arranged in the poll parameters, there are deadlines for each active phase of a poll. It is also possible to do this manually with another Awless command:

```
awless tawa force $url $name contribution
```

This forces the phase to move to the next poll and there is logic at the server end to complete housekeeping tasks, such as computing any cryptographic functions, before the switch is made and to prepare properly for the new phase.

The status command returns information needed to monitor the progress of the poll:

```
awless tawa status $url $name
```

This returns the current phase name plus the counts of clients which have replied to the calls for contributions, selection and the final upload of encrypted data plus shared keys. This allows the Awless user to tell when the required numbers have been reached and to switch to the next phase enabling complete manual control of the poll.

Apart from these auxiliary request/response functions only the `select` command does not relate directly to the cryptography. This is a phase inserted between the statistics gathering phase and the completion phase and consists of the clients sending their “data present” flag `hasData` which is used to allocate the clients to their index in the actual cryptography itself. The actual selection is carried out when the poll switches to `COMPLETED` mode.

The remaining commands are:

- `activate` which runs the `Setup` function for the cryptographic protocol and returns the selection index plus the `mpk`,
- `send` which runs the `Encrypt` and `DKeyGen` functions on the client and sends the values for storage on the server and
- `decrypt` which combines the shared data and runs the `Decrypt` function to generate the results.

### 2.5.1 Server API

The full API for the proxy server is as follows:

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	12 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

```

func initTawaHandlers(c *config.Config, router *restful.Router, h *handlers.Sessions) {
    // Create, delete and monitor polls
    router.MustHandleChain("/api/v0/tawa/define", "POST", h.ChainLogTx, h.TawaDefine)
    router.MustHandleChain("/api/v0/tawa/delete", "POST", h.ChainLogTx, h.TawaDelete)
    router.MustHandleChain("/api/v0/tawa/status", "POST", h.ChainLogTx, h.TawaStatus)
    // Main poll and crypto phases
    router.MustHandleChain("/api/v0/tawa/contribute", "POST", h.ChainLogTx, h.TawaContribute)
    router.MustHandleChain("/api/v0/tawa/select", "POST", h.ChainLogTx, h.TawaSelect)
    router.MustHandleChain("/api/v0/tawa/activate", "POST", h.ChainLogTx, h.TawaActivate)
    router.MustHandleChain("/api/v0/tawa/keygen", "POST", h.ChainLogTx, h.TawaKeygen)
    router.MustHandleChain("/api/v0/tawa/decrypt", "POST", h.ChainLogTx, h.TawaDecrypt)
    // Auxiliary endpoints for debug
    router.MustHandleChain("/api/v0/tawa/posterror", "POST", h.ChainLogTx, h.TawaPostError)
    router.MustHandleChain("/api/v0/tawa/force", "POST", h.ChainLogTx, h.TawaForce)
}

```

The individual endpoints have the following functionality:

**2.5.1.1 Create poll** The type below is the create poll call parameter structure in protobuf format. There are quite a lot of parameters required to define a poll, these are explained in Table 2. Note that for create the only response is an OK flag which indicates success plus an error message upon failure. The action taken at the server upon this call is to simply create a database entry with the given poll parameters and set the phase for that poll to the initial poll state. Some verification is applied to the parameters.

```

1  enum TawaCryptoType {
2      DMCFE_BN256 = 0;
3      DMCFE_WR_E1 = 1;
4      DMCFE_WR_E2 = 2;
5  }
6
7  enum TawaStatus {
8      NULL = 0;
9      CREATED = 1;
10     CONTRIBUTION = 2;
11     STATISTICS = 3;
12     SELECTION = 4;
13     COMPLETED = 5;
14     FAILED = 6;
15 }
16
17 enum TawaSelectionType {
18     MOSTSIGNIFICANT = 0;
19     RANDOMSAMPLE = 1;
20 }
21
22 message TawaDefineRequest {
23     string Name = 1;
24     int32 Status = 2;
25     repeated string Commands = 3;
26     string Start = 4;
27     string Contribution = 5;
28     string Statistics = 6;
29     string Selection = 7;
30     uint32 MaxClients = 8;
31     uint32 NumClients = 9;
32     uint32 Bound = 10;
33     bool AllowNullData = 11;
34     bool Manual = 12;
35     TawaSelectionType SelectionType = 13;
36     TawaCryptoType CryptoType = 14;

```

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	13 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
<b>Version:</b>	1.0	<b>Status:</b>	Final

```

37 }
38
39 message TawaDefineResponse {
40     bool OK = 1;
41 }

```

Parameter	Description
<b>Name</b>	The name of the poll to create.
<b>Status</b>	The starting phase for the poll (usually CREATED or CONTRIBUTION. The NULL phase simply means that the poll has not been defined and is only used for status query responses.
<b>Commands</b>	List of AWS endpoints for which to gather statistical information using the standard naming scheme.
<b>Start</b>	For automatic poll timing, this is the date and time at which to end the CREATED phase and switch to CONTRIBUTION.
<b>Contribution</b>	Length of time to remain in the CONTRIBUTION state.
<b>Statistics</b>	Length of time to remain in the STATISTICS state.
<b>Selection</b>	Length of time to remain in the SELECTION state.
<b>MaxClients</b>	Total number of clients allowed to contribute to the poll, ie. before selection.
<b>NumClients</b>	Number of clients to select for the poll.
<b>Bound</b>	Crypto parameter, bound for the integers for the DMCFE_BN256 protocol.
<b>AllowNullData</b>	Boolean flag, when set allows clients with no data to contribute to the poll.
<b>Manual</b>	Boolean flag, override the timing parameters above and assume that the force command will be used to switch between polling phases.
<b>SelectionType</b>	Indicates the method of selection, either according to statistical significance or a purely random sample.
<b>CryptoType</b>	Protocol to use for the functional encryption. Several protocols are possible but we only use the DMCFE_BN256 protocol for now.

Table 2: Create request parameters

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	14 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final



2.5.1.2 Delete poll For the delete poll endpoint the only request parameter is the name of the poll and the only result is the OK status. The poll database entry is deleted along with *any other associated data*. This includes the results of the poll.

```

1 message TawaDeleteRequest {
2     string Name = 1;
3 }
4
5 message TawaDeleteResponse {
6     bool OK = 1;
7 }

```

2.5.1.3 Status of poll The status call takes the name of a poll and returns the current poll phase plus the numbers of responses from each of the data collection phases, these include contributions, selections, participants (in the crypto itself) and Ciphers which is the number of clients which have returned their encrypted data.

```

1 message TawaStatusRequest {
2     string Name = 1;
3 }
4
5 message TawaStatusResponse {
6     int32 Status = 1;
7     uint32 Contributors = 2;
8     uint32 Selected = 3;
9     uint32 Participants = 4;
10    uint32 Ciphers = 5;
11 }

```

2.5.1.4 Contribute The request for signaling intention to contribute is simply the poll name. The server responds, however, with a sufficient number of parameters for the polling to proceed on the client including the cryptographic parameters. For the most part these values correspond to the poll creation parameters listed in Table 2 except that the `contribute` response also includes the current poll phase plus a client number to be used for subsequent communications. The server functionality for this call is to increment the current count for contributors.

```

1 message TawaContributeRequest {
2     string Name = 1;
3 }
4
5 message TawaContributeResponse {
6     string Name = 1;
7     int32 Status = 2;
8     repeated string Commands = 3;
9     uint32 ClientNumber = 4;
10    uint32 MaxClients = 5;
11    uint32 NumClients = 6;
12    int32 CryptoType = 7;
13    uint32 Bound = 8;
14 }

```

2.5.1.5 Selection For selection, the client checks if data is present for each endpoint count and sets `HasData` to be `true` if any of them are non-zero. The significance value is computed from the endpoint

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	15 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final



counts in order to penalize zero values  $((total * total * (nocmds - zeros))/nocmds)$ , where `total` is the total count for all endpoints, `nocmds` is the number of endpoints that have been monitored and `zeros` is the number of endpoint counts which equal zero). These values are sent in cleartext so we need to confirm with the clients that it acceptable to provide this information unencrypted.

At this point, the server only records the number of clients selected plus their statistical summary. The selection itself is actually performed when the poll phase is switched from SELECTION to COMPLETED.

```

1 message TawaSelectRequest {
2     string Name = 1;
3     uint32 ClientNumber = 2;
4     bool HasData = 3;
5     int32 Significance = 4;
6 }
7
8 message TawaSelectResponse {
9     bool OK = 1;
10 }

```

**2.5.1.6 Activation** The `activate` call is the first true cryptographic operation for the FE. Activation consists of the clients again signaling willingness to continue with the poll and in return receive the necessary parameters to perform encryption. Note that clients which have not been selected receive dummy information. Ultimately, this is so that non-selected clients do not actually know that they have not been selected, however, for the moment we return the selection number as part of the response. Normally, this operation would work simply by using the conversion table between the original client number and the selected number at the server end but unfortunately for the DMCFE protocol we require the select number as a parameter to the `Encrypt` function. This may in fact never need to be implemented properly if the protocol becomes obsolete. Apart from this, the `activate` call returns the  $dk_1$  value in the form of a  $2 \times 2$  matrix as elements (`t00` etc.) The `Status` and `Commands` values are included for convenience. The functionality of the `activate` call is to compute the master public key and keep track of which clients have responded.

```

1 message TawaActivateRequest {
2     string Name = 1;
3     uint32 ClientNumber = 2;
4 }
5
6 message TawaActivateResponse {
7     string Name = 1;
8     int32 Status = 2;
9     repeated string Commands = 3;
10    uint32 ClientNumber = 4;
11    uint32 SelectNumber = 5;
12    uint32 NumClients = 6;
13    bytes t00 = 7;
14    bytes t01 = 8;
15    bytes t10 = 9;
16    bytes t11 = 10;
17 }

```

**2.5.1.7 Key generation** The `keygen` call allows the clients to upload their encrypted counts (`EncryptedCount`) and generated distributed keys (`KeyShare`) to the server. The encrypted squared counts are wrapped together with the original values for efficiency. The server does not perform any computation at this point, the shared values are merely stored in the database.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	16 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

```

1 message TawaKeygenRequest {
2     string Name = 1;
3     uint32 ClientNumber = 2;
4     string Label = 3;
5     bytes EncryptedCount = 4;
6     repeated bytes KeyShare = 5;
7     bytes EncryptedCountSq = 6;
8     repeated bytes KeyShareSq = 7;
9 }
10
11 message TawaKeygenResponse {
12     bool OK = 1;
13 }

```

2.5.1.8 Decryption The `decrypt` call is open to all clients. The first client to request decryption triggers the `Decrypt` function the results of which are then stored into the database for subsequent calls. The response is a list of poll results, one for each endpoint monitored during the statistics gathering phase. A result consists of the command name, the date and time that the poll was completed, the sum of access counts plus the sum of the squares of these values and also the type of cryptography used to perform the FE. The caching is essential because this call is computationally heavier than the rest. All of the shared keys and encrypted data have to be collected from the database and combined into the requisite values for the `Decrypt` call. If a poll has failed, because of missing or erroneous values then this is the call where the error shows up. Currently, the error analysis is very primitive but future versions of this call may be able to extract useful information by requesting further data from the clients. The average values can be computed from:

$$\text{SumAccesses}/\text{NumClients}$$

and the variance can be computed from:

$$(\text{SumSqAccesses} - (\text{SumAccesses} * \text{SumAccesses})/\text{NumClients})/(\text{NumClients} - 1.0)$$

```

1 message TawaDecryptRequest {
2     string Name = 1;
3 }
4
5 message PollResult {
6     string Command = 1;
7     string PollDate = 2;
8     string SumAccesses = 3;
9     string SumSqAccesses = 4;
10    int32 CryptoType = 5;
11 }
12
13 message TawaDecryptResponse {
14     bool OK = 1;
15     repeated PollResult Results = 2;
16 }

```

2.5.1.9 Post error into error log The `posterror` call allows clients to log errors into the servers' error logs. This feature has limited use since many potential errors results in failure of the client/server connection. It has been included for the rare circumstance in which an error on a client during a poll needs

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	17 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
<b>Version:</b>	1.0	<b>Status:</b>	Final

to be recorded on the server for subsequent analysis. It will be more useful for actual live polls than for the demonstrations given so far.

```

1 enum TawaSeverity {
2     Debug = 0;
3     Info = 1;
4     Notice = 2;
5     Warning = 3;
6     Error = 4;
7     Critical = 5;
8 }
9
10 enum TawaErrorKind {
11     PollInternalError = 0;
12     InsufficientParticipants = 1;
13     PollInformation = 2;
14     PollManagerError = 3;
15     PollProtocolError = 4;
16     PollComment = 5;
17     PollClientError = 6;
18 }
19
20 message TawaPostErrorRequest {
21     string stamp = 1;
22     string name = 2;
23     uint32 source = 3;
24     TawaSeverity severity = 4;
25     TawaErrorKind kind = 5;
26     string errortext = 6;
27 }
28
29 message TawaPostErrorResponse {
30     bool OK = 1;
31 }

```

**2.5.1.10 Force poll status** The force poll call is not intended to be part of the normal operation of the polling process. It is present to enable manual control of the process by allowing the phases to be switched externally. For this to work, the poll must be put in manual mode (see Table 2). When the phases are switched the normal polling operations associated with that change are called although there is currently no logic to prevent illegal phase changes. The parameters for this call include the desired poll Status value and, if successful, the response will include the actual new state.

```

1 message TawaForceRequest {
2     string Name = 1;
3     int32 Status = 2;
4 }
5
6 message TawaForceResponse {
7     int32 Status = 2;
8 }

```

## 2.5.2 Awless code

**2.5.2.1 Internal database** Awless has an internal database for configuration data but we keep the web analytics data separate. The internal DB structure is almost exactly the pollConfig data provided by the server but with a few exceptions:

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	18 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final

```

type PollDefinition struct {
    // Poll data
    Name          string
    Status        api.TawaStatus
    ServerUrl     string
    Commands     []string
    Start        time.Time
    Contribution  time.Duration
    Statistics    time.Duration
    Selection     time.Duration
    RemoteLogging bool
    AllowNullData bool
    Manual       bool
    SelectionType api.TawaSelectionType

    // Crypto data
    ClientNumber int
    SelectNumber int
    MaxClients   int
    NumClients   int
    CryptoType   api.TawaCryptoType

    // Needed by BN256
    Bound int
    T      [2][2][]byte
}

```

We store the address of the server along with the poll configuration data. General commands (such as `create`) always take the server URL from the command line but any polling-related commands use the address stored in the DB. The `RemoteLogging` flag allows an Awless user to disable sending error messages to the server. Finally, we also store the `ClientNumber` and `SelectNumber` values as well. These are updated during polling rather than being provided during configuration.

**2.5.2.2 Statistics gathering** Currently, we implement gathering of mean and variance values for counts of accesses to endpoints on the Amazon AWS service. The recording process is simple and is integrated into the main function dispatch routine in Awless:

```

/** RecordCommand
 * Increment the access count for a given command.
 */
func RecordCommand(db *database.DB, cmd []string) error {
    isStatistics, err := CheckStatusRequiredCPD(db, api.TawaStatus_STATISTICS)
    if err != nil {
        return err
    }
    if isStatistics {
        var index string
        var cnt int
        if index, err = indexFromCommand(cmd); err != nil {
            return err
        }
        err = db.SetStringValue(RecordCommandIndexKey, index)
        if err != nil {
            fmt.Printf("RecordCommand: can't store index %s\n", index)
        }
        if !indexIsMonitored(index) {
            return nil
        }
        if cnt, err = db.GetIntValue(strconv.Itoa(clientIndex) + index); err != nil {
            cnt = 0
        }
        cnt = cnt + 1
        return db.SetIntValue(strconv.Itoa(clientIndex)+index, cnt)
    }
    return nil
}

```

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	19 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

```
}

```

The counts are zeroed when Awless connects to the proxy server and are incremented each time a command is issued (cmd in the above code) which has an index which matches one of the Command values in the poll configuration. Note that we prefix the index with the client number in order to partition the counts for multiple clients in the same Awless instance. This is a debug feature which is also used to enable the demonstrator.

At the point of performing the functional encryption Encrypt function, we have the total number of accesses to each monitored endpoint. We can then encrypt these values according to the shared key provided by the proxy server:

```
// Currently we simply set the y values to 1 (average only).
y := data.NewConstantVector(cPD.NumClients, big.NewInt(1))

// Encrypt the values and turn into byte arrays
encCntBytes, keyShareBytes, err = common.EncryptValue(cPD.SelectNumber, cnt, idx, T, y)
if err != nil {
    return err
}
if isVar {
    encCntSqBytes, keyShareSqBytes, err = common.EncryptValue(cPD.SelectNumber, cnt*cnt, idx, T, y)
    if err != nil {
        return err
    }
} else {
    encCntSqBytes = []byte{}
    keyShareSqBytes = [][]byte{}
}

```

The isVar flag is computed from the index and indicates that we wish to encode the square of the count at the same time as the count. Computing the mean and variance from the counts and squared counts is very simple:

```
mean := sum / n
var variance float64
if n == 1 {
    variance = 0.0
} else {
    variance = (sumsq - (sum*sum)/n) / (n - 1.0)
}

```

where n is the number of clients, sum is the total count for one endpoint and sumsq is the sum of squares for the same endpoint. In terms of the statistics we perform for FENTEC this simple result is adequate and could be extended to include other values, such as counts of errors or even system parameters such as maximum memory occupancy. Higher statistical moments should also be possible but are not particularly useful except under exceptional circumstances.

2.5.2.3 Awless commands The relationship between the user-visible commands and the functioning of the poll is as follows:

- The create command has the following arguments:

```
awless tawa create URL NAME MAXCLIENTS NUMCLIENTS COMMANDS START_TIME CONTRIBUTION_DURATION \
    STATISTICS_DURATION SELECTION_DURATION ALLOW_NULL_DATA \
    MANUAL SELECTION_TYPE CRYPTO_TYPE BOUND
```

These parameters relate exactly to the parameters described in Table 2. The only action awless takes upon this command is to call the create endpoint at the server at the given URL. The client creating the poll is not added to the poll, it must perform the normal contribute operation.

- The delete command is very simple:

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	20 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b>
			Final

```
awless tawa delete URL NAME
```

The `delete` endpoint is called at the given URL and the poll with the given name is deleted. Note that *all* of the poll-related data is deleted, including the poll result, if present.

- The `status` command displays the current poll status.

```
awless tawa status [--phaseCount int] URL NAME
```

If the `-phaseCount` flag is missing then just the phase name is printed, otherwise the output is an integer value for one of the count values. This is used to implement scripts for running a poll under manual mode.

- The `force` command forces the status of the named poll to the supplied value and is intended for manual control of polling.

```
awless tawa force URL NAME STATUS
```

Again, this command is not treated as poll-specific so the URL address is required. Any poll phase can be given since this command is really for debug purposes but it only makes sense for the normal sequence of phases.

- The `contribute` command tells the server that the client wishes to contribute to a poll and returns the poll configuration data. This is the first point of entry into the poll so the URL address is required.

```
awless tawa contribute URL NAME
```

This command is only valid if the poll is currently in the `CONTRIBUTE` phase. The Awless client therefore issues a `status` command to update the local copy of the phase status. Apart from updating the configuration data the counts for the statistical gathering code are all reset.

- The `select` command is only valid during `SELECTION` mode.

```
awless tawa select URL NAME
```

Yet again the action carried out by Awless is to compute the necessary statistical summary and call the `select` endpoint on the server.

- The `send` command is more complicated for the Awless clients. Since the results of the redistributed `Setup` data are required for encryption the `send` command performs the `activate` and `keygen` calls in a single operation.

```
awless tawa send
```

The URL provided by the `contribute` command is used. The `activate` call returns the combined encryption key ( $\overline{dk}_1$ ) which is then used to encrypt the statistical data. The distributed key ( $\overline{C}_{y,i}$ ) is then generated and the two are combined in a single `keygen` call which stores these values on the server.

- The `decrypt` command triggers the `Decrypt` function on the server which is assumed to have gathered the necessary data to perform it. This command only works in `COMPLETED` mode and after all the encrypted data and shared keys have been uploaded.

```
awless tawa decrypt URL NAME
```

The server `decrypt` call only returns the counts for each endpoint monitored, the Awless client converts these into average and variance and generates a suitable display.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	21 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

- The `comment` command allows the user to insert a message into the server's logs.

```
awless tawa comment URL NAME COMMENT
```

- The `set` command allows control over the local behaviour of the Awless client's web analytics functions. Currently only the `RemoteLogging` facility can be switched on and off.

```
awless tawa set PARAMETER VALUE
```

In general, the functionality in the client code has been kept as simple as possible. However, the process could be made more automatic. The current design was mainly to enable experimentation with the various options for deploying the cryptographic protocol. Future versions of the client code will mostly just involve enabling the polling process and then letting it run automatically in the background.

## 2.6 Demonstrator

To present the techniques proposed by the WALLIX web analytics use case in short period of time presents some difficulties. Firstly, the time scales for significant statistics gathering are far too long for demonstration purposes. Secondly, our application requires access to both Amazon AWS services plus a configured and operational version of the WALLIX Awless tool. Finally, very little is visible during the polling process apart from the presentation of the final result.

To overcome these problems and allow some visible presentation of how the web analytics is supposed to work, we have devised a simple demonstration framework in which a small-scale poll is conducted locally. The proxy server has been deployed in an open manner to allow connection via a locally-installed Awless client. The Awless client is then front-ended by a simple web page allowing AWS operations to be conducted by proxy through the single Awless installation. The individual clients are identified by the web server and the AWS name-spaces are partitioned by prefixing the name-space with a value unique to each connected participant.

### 2.6.1 Demonstration architecture

The demonstration core is simply a web server implemented alongside the proxy server. This serves a simple form-driven page with no client-side code and implements a simple protocol for translating requests to create, delete and list users into actual Awless commands on the server. This is illustrated in Figure 2. To conduct a demonstration, a small number of volunteers are asked to log into the web page and execute random commands using the demonstration web page. Four commands are possible:

- Create a user on the AWS account.
- Delete a user from the AWS account.
- List the users currently defined.
- Display the results of the FE decryption.

The last option is only available when all of the participating clients have returned their encrypted data. The poll is run according to the following procedure:

- A poll is created on the proxy server.
- The poll is manually switched from `CREATED` to `CONTRIBUTION`.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	22 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

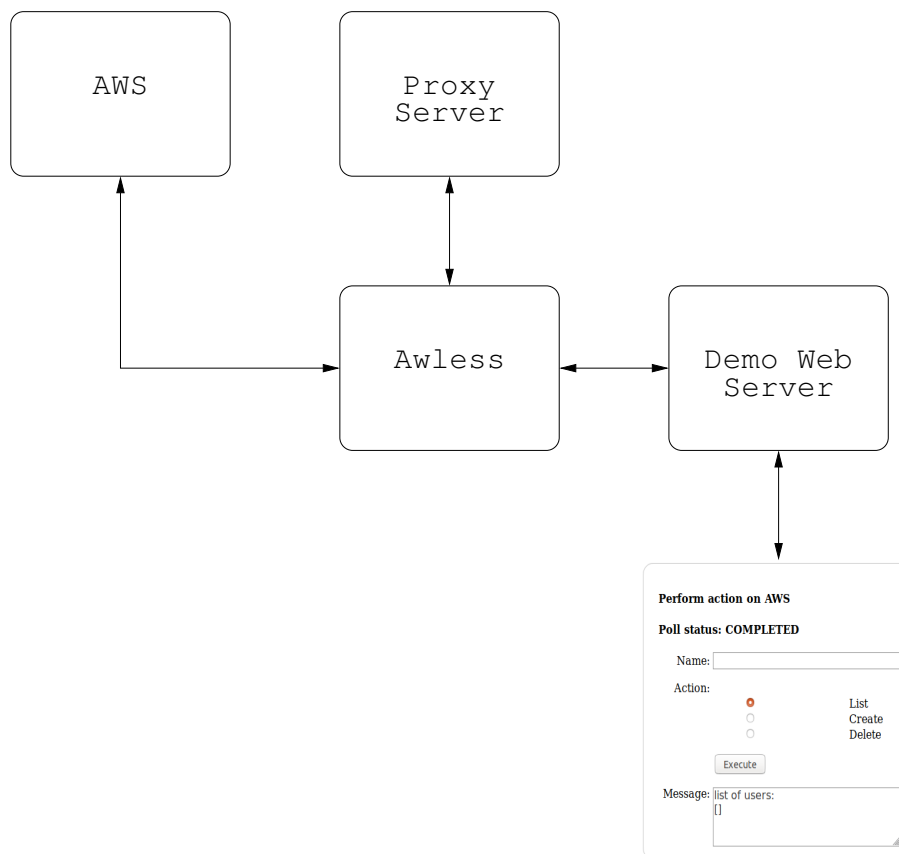


Figure 2: Architecture of the FENTEC Web Analytics Demo

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	23 of 29	
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	
	<b>Version:</b>	1.0	<b>Status:</b>	Final



- Each client performs any command and the `contribute` Awless call is performed automatically.
- Once all of the designated clients have called `contribute` the proxy server automatically switches to `STATISTICS`.
- For a short period of time each of the users runs a few random commands to generate some statistical data.
- The poll is manually switched from `STATISTICS` to `SELECTION`.
- Each user then causes the client to execute another random command and triggers the `select` function from Awless.
- Once all of the clients have executed `select` the proxy server automatically switches the poll to `COMPLETED`.
- Each user then executes one more command which causes the Awless client to execute the `send` command to trigger sending of the encrypted data.
- The `Decrypt` command becomes available on the demonstration web page and can be used to trigger the `decrypt` function and display the results.

## 2.6.2 Demonstration security

For legal reasons we are not allowed to use real data for this demonstration. What we are using here, however, is an expired free-tier AWS account which does not allow any paying actions on the AWS servers. For example, it does not allow the user to deploy a server online which, depending upon the type and size of server deployment can be a very expensive operation. In fact, the free account allows the users to create and delete user names but they are completely blank. They have no policies associated with them, cannot be used to log into AWS services and are deleted after the demonstration ends.

Note that we cannot prevent, for example, a user from accidentally typing their bank account data in as a user name and creating a user with this name on AWS servers but even this situation is relatively harmless since the only action which can be carried out is to display the name at the original user's web page. No other clients will see this data.

Similarly, the SQL server only stores the poll data (for example, the cryptographic parameters) plus the temporary data needed to run the FE. None of this data is user identifiable and in any case is deleted as soon as the poll is deleted after the demonstration has ended.

We thus contend that this demonstration meets the legal restrictions placed upon the FENTEC project by its contract with the European Union.

## 2.6.3 What does the demonstration prove?

In this demonstration we have shown that Functional Encryption can be used to perform statistical analysis in a useful manner. We have investigated the infrastructure necessary to deploy the cryptographic protocol and manage the intermediate data in a secure manner. Our initial design proved to be quite accurate in this respect since the client/server model imposed upon us matches the proxy server/Awless client combination. We have proved that for Awless we can compute the mean and variance of integer values gathered on a set of distinct clients. We were able to perform some initial optimization of the process by merging request/response pairs and also reduced the communications burden by colocating multiple values in the same messages, for example the sum values and the squared sum values can be transmitted together.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	24 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

A more difficult problem is the sub-sampling that we have implemented. This is very primitive but we were at least able to eliminate non-significant data provided we can get consent from our clients to transmit non-meaningful statistical summaries in cleartext. It may be possible to deploy the FE in such a way as to perform the sub-sampling in a secure manner but that remains an open problem.

In summary, our demonstration meets the key security requirements from WALLIX's point of view. The users' data are not visible to the poll manager (in this case, the proxy server) and the users are not able to access each others' data unless by deliberate collusion.

#### 2.6.4 What does the demonstration not prove?

Although the cryptographic protocol is sound and backed up by a proof of semantic security we still have to verify the security of the implementation. Since our implementation is not quite how we would design it for use in a real-life poll with potentially sensitive data, this validation would more usefully be carried out on a version more intended for a large-scale poll on the internet. We can, however, perform a partial audit of the security of the system on what we have since many parts of the system will be common to such a version.

We can compute very accurately the communications sizes for a real deployment but the actual transmission times may vary widely depending upon the amount of resources used to implement the servers. We are also hampered somewhat by the fact that our clients will be from a widely-varying set of communications performances. Without a full-scale deployment we cannot make accurate estimates of communications times.

Similarly, we can measure the performance of our prototype and perform extrapolation onto a potential large-scale poll but there are errors inherent in this process. In this case, our main problem is that we cannot easily scale our prototype into any realistic size. Each client is supposed to be an Awless instance linked to a real AWS account performing a wide variety of tasks in order to spot potential improvements which is our fundamental motivation for using FE technology.

#### 2.6.5 Difference between demonstrator and actual working poll system

In order to build a full-size working polling system we would have to modify our demonstrator accordingly:

- The poll management technique could be improved. Specifically, the client code should be automated as much as possible. Setting hard deadlines for each phase has turned out to be quite inflexible so we require more intelligent decisions as to when the phases can be switched, for example we could delay deadlines until the required counts are arrived at. Ultimately, we will have to retain some element of manual override for the system in the event of unforeseen circumstances. It may be possible, for example, to regress the phase in order to redo a failed operation but then cryptographic integration becomes more difficult.
- The prototype does not implement full authentication for the proxy server. DataPeps has an authentication mechanism for registered users. This was not included in the prototype server code but could be used in a full deployment to secure access to the poll and its results. Unfortunately, the mechanism is rather complicated and integrated into DataPeps security.
- Currently, any client can access the poll results. It is possible to envisage circumstances in which the poll results themselves can be considered sensitive information. With proper authentication built into the poll it would be simple to implement a system whereby only selected clients were able to access the results of the poll. It is also possible to provide access to poll results for external entities. Due to the nature of the cryptographic protocol the proxy server will always have access to the results and so can decide who gains access to them.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	25 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

- Deployment of the proxy server has only been implemented locally. For a full-scale poll we should deploy this server on a cloud platform. There are no technical difficulties to doing this but if significant computational ability or communications bandwidth were required it could prove expensive.
- As already stated, we can only do sampling in cleartext mode at present. For a full-size poll our vulnerability to missing data could prove a problem so some method of performing the sampling using FE could be investigated. Unfortunately, it is fundamental to FE that there can be no missing data (if we could perform FE with one missing element, we could easily perform the operation with only one data point thus unmasking the original data). It may be possible to perform sampling using multiple FE operations during the same poll.

## 2.7 Performance and Security

A preliminary performance test, Table 3, shows that the performance of the prototype is adequate for WALLIX's purposes. For 200 clients the decryption function can be executed in 17 seconds on a laptop. Since Awless has about 150 registered users and we have at most 10 to 20 values we wish to measure, the computational requirements for the cryptography are easily met. The performance of the current prototype will be reported in FENTEC deliverable D7.11 [8].

No. Clients	Setup	Encrypt	DKeyGen	Decrypt
5	63.910 $\mu$ s	9.4718ms	49.242ms	1.1443s
10	113.49 $\mu$ s	18.060ms	93.771ms	2.0968s
20	226.09 $\mu$ s	38.433ms	201.16ms	4.6551s
50	776.92 $\mu$ s	98.445ms	502.45ms	8.0319s
100	1.2902ms	198.15ms	1.0150s	9.6316s
150	1.5321ms	301.36ms	1.5106s	11.989s
200	2.3611ms	407.45ms	2.0389s	17.154s

Table 3: Timings for cryptographic functions for the DMCFE protocol

The communications loads are likewise well within parameters. The messages range from just a few bytes to approximately 200 bytes for the largest message. Since the communications complexity is linear for this protocol the transmissions are well within even the slowest conceivable network's capability. As already explained, in a wider context with much larger computational and communications requirements the implementation may need to be hosted by more significant architectures than we intend for FENTEC. Performing a security audit of the current prototype would have limited value here but will be reported in greater detail in FENTEC deliverable D7.9 [9]. For reasons already stated we need to upgrade the internal security for the prototype before being able to make meaningful statements about the implementation security in a wider context.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	26 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

### 3 Conclusion

In this deliverable we have presented the first version of the Wallix web analytics use case prototype. For this initial version we concentrated upon functionality and solving some of the operational difficulties in deploying the protocol in a real-life application.

We have a demonstrator for the prototype which can be used to show how a poll would work in practice before a small audience. Some shortcuts were necessary to enable this demonstration but it does have all of the essential elements of a fully-implemented polling system. We are able to conduct a small poll by allowing proxy access to a predefined Awless installation and can demonstrate the results of distributed statistical analysis using functional encryption.

The functionality of the polling method is more-or-less complete but there are some weaknesses in the method. Firstly, the requirement for having *all* of the data to be able to decrypt has proven rather fragile. For this first version we can work around the problem but we may need to investigate secure methods for dealing with this problem, either by rerunning phases of the poll to secure the required data or look at secure sub-sampling methods to overcome the missing data.

Although we have not performed any systematic testing of the prototype we can say with confidence that it already meets the performance requirements for the specific application which we described in Section 2.1. In a wider context, however, we may have some computational or network issues to deal with and we need to quantify what the current technique is capable of.

In summary, the prototype is at the stage expected for this phase of the FENTEC project. We have a working prototype with which we can move forward and develop to be more robust and performant in the later stages of the project.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	27 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final

---

## 4 Next steps

---

Our priorities for the remainder of the FENTEC project are as follows:

- Perform detailed performance analysis on the prototype.
- Analyze the security of the prototype implementation.
- Integrate new cryptographic protocols as they are produced by the academic work.
- Deploy the proxy server on a cloud platform with a view to designing a full-scale version of the web analytics system.
- Investigate integration of the technique into our DataPeps platform.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	28 of 29				
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final

## References

- [1] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Steven Galbraith and Thomas Peyrin, editors, *Advances in Cryptology — ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 703–732. Springer, 2018.
- [2] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with repetition for inner product. *Cryptology ePrint Archive*, Report 2018/1021, 2018. <https://eprint.iacr.org/2018/1021>.
- [3] FENTEC. D3.1 technical requirement report analysis. Technical report, European Commission, 2018.
- [4] FENTEC. D4.1 annual report on functional encryption schemes for prototypes y1. Technical report, European Commission, 2018.
- [5] FENTEC. D7.1 preliminary specification of fentec prototypes. Technical report, European Commission, 2018.
- [6] FENTEC. D4.2 annual report on functional encryption schemes for prototypes y2. Technical report, European Commission, 2019.
- [7] FENTEC. D6.2 preliminary functional encryption toolset api. Technical report, European Commission, 2019.
- [8] FENTEC. D7.11 performance report for fentec prototypes after first cycle. Technical report, European Commission, 2019.
- [9] FENTEC. D7.9 first test report of the fentec prototypes. Technical report, European Commission, 2019.

<b>Document name:</b>	D7.3 First Version of the Truly Anonymous Data Collection Prototype	<b>Page:</b>	29 of 29
<b>Reference:</b>	D7.3	<b>Dissemination:</b>	PU
	<b>Version:</b>	1.0	<b>Status:</b> Final