D4.3 Final Perturbation Analysis of the Implementation

Jesus Luna (TUD), Neeraj Suri (TUD),
Giancarlo Pellegrino (TUD), Heng Zhang (TUD),
Michael Bladt Stausholm (ALX)

Editors:	*Jesus Luna, Neeraj Suri, Giancarlo Pellegrino (TUD)*
Reviewers:	*Norbert Goetze (NSN), Ioannis Stamatiou (CTI)*
Identifier:	*D4.3*
Type:	*Deliverable*
Version:	*1.0*
Date:	*30/07/2014*
Status:	*Final*
Class:	*Public*

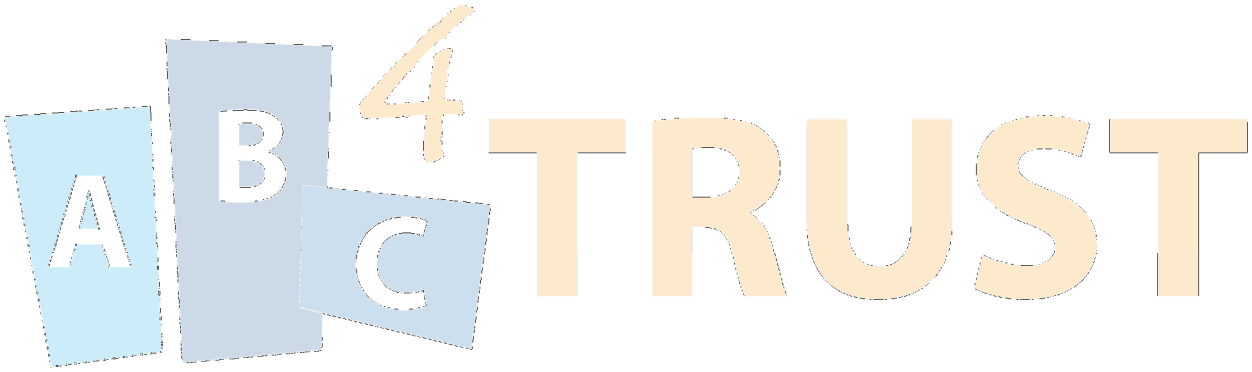
Abstract

The activity of Perturbation Analysis assesses the robustness of the ABC4Trust architecture as initially described in Deliverable D2.1 and Heartbeat H2.1. Following the basic test cases that verify the correctness of the implementation (Task 4.6), the perturbation analysis activity designs individual misuse cases, and plans campaigns to inject a range of perturbations (e.g., at the ABCE API-level) on the different operational blocks and also the architectural data flows to test the robustness of the implementation and to suggest any corrective actions as needed.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

Members of the ABC4TRUST consortium

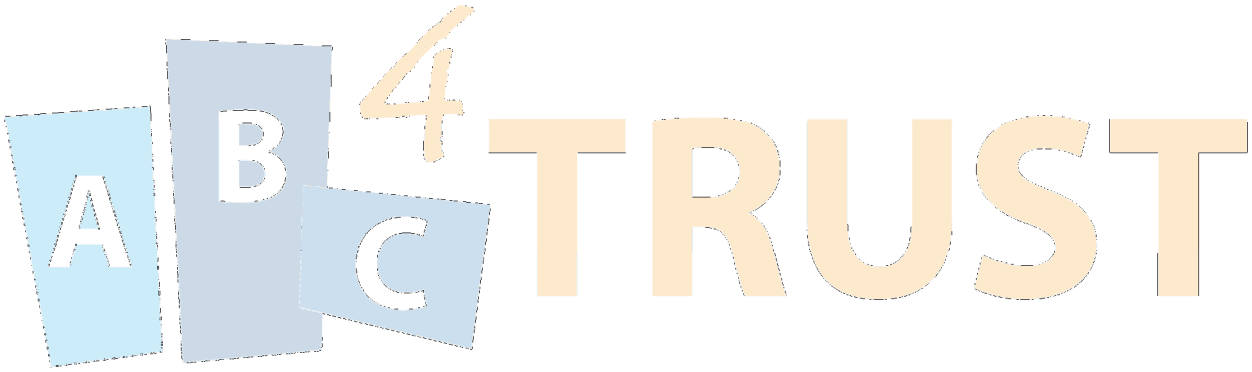
1.	Alexandra Institute AS	ALX	Denmark
2.	CryptoExperts SAS	CRX	France
3.	Eurodocs AB	EDOC	Sweden
4.	IBM Research – Zurich	IBM	Switzerland
5.	Johann Wolfgang Goethe – Universität Frankfurt	GUF	Germany
6.	Microsoft Research and Development	MS	France
7.	Miracle A/S	MCL	Denmark
8.	Nokia Solutions and Networks GmbH & Co. KG	NSN	Germany
9.	Research Academic Computer Technology Institute	CTI	Greece
10.	Söderhamn Kommun	SK	Sweden
11.	Technische Universität Darmstadt	TUD	Germany
12.	Unabhängiges Landeszentrum für Datenschutz	ULD	Germany



The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

List of Contributors

Chapter	Author(s)
Executive Summary	Jesus Luna, Neeraj Suri (TUD).
Introduction	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino (TUD).
Setup	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino, Heng Zhang (TUD) and Michael Bladt Stausholm (ALX).
Issuance	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino, Heng Zhang (TUD) and Michael Bladt Stausholm (ALX).
Presentation	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino, Heng Zhang (TUD) and Michael Bladt Stausholm (ALX).
Revocation	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino, Heng Zhang (TUD) and Michael Bladt Stausholm (ALX).
Inspection	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino, Heng Zhang (TUD) and Michael Bladt Stausholm (ALX).
Summary	Jesus Luna, Neeraj Suri, Giancarlo Pellegrino, Heng Zhang (TUD) and Michael Bladt Stausholm (ALX).



The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

Executive Summary

This deliverable investigates the robustness of the ABC4Trust reference implementation. The process followed is of perturbation analysis, which entails introducing deliberate perturbations to investigate the degree of deviance or tolerance by the ABC4Trust architecture towards them. The perturbation analysis was conducted on the ABCE (and related core components invoked by the ABCE) of the reference implementation (D2.1 and Heartbeat H2.2 – "old crypto" architecture). Then, those tests that resulted in non-compliant outcomes (i.e., do not follow the specification) were repeated on the "new crypto architecture" (D4.2 and Heartbeat H4.1) for comparison purposes. This process allowed developers of the reference implementation to design and assess the effect of the required corrective actions.

This document presents both the methodology of the Perturbation Analysis as well as the obtained results over the full life cycle of the privacy-ABC credentials. This covers both the basic tests to verify the implementation correctness (Task 4.6), the perturbation analysis activity designs individual misuse cases, and plans campaigns to inject a range of perturbations (e.g., at the ABCE API-level) on the different operational blocks and also the architectural data flows to test the robustness of the implementation. As an overview, the deliverable documents the analysis from the perturbations conducted at the data-flow and component-levels of the ABCE. In addition, while not part of the PA, some supplemental misuse observations as reported by the pilots are included in the Appendix.

Overall, the results from the conducted Perturbation Analysis highlight the uncovered robustness-related risks, and also provided a preliminary guidance for the corrective actions to take while developing the resultant ABC4Trust reference implementation (as reported in Heartbeats H2.2, H4.1 and finally in Deliverable D4.2).

This deliverable was co-authored by Jesus Luna (Executive Summary, Introduction, Sect. 1-7), Neeraj Suri (Executive Summary, Introduction, Sect. 1-7), Giancarlo Pellegrino (Introduction, Sect. 1-7), Heng Zhang (Sect. 1-7) and Michael Bladt Stausholm (Introduction, Sect. 1-7).

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257782 for the project Attribute-based Credentials for Trust (ABC4Trust) as part of the "ICT Trust and Security Research" theme.

Table of Contents

Index of Figures

Index of Tables

1 Introduction

The objective of the "perturbation analysis activity" is to experimentally assess the robustness of the ABC4Trust's reference implementation. *For the purposes of this document, robustness will be understood as "the implementation's correctness (in particular integrity) in the presence of failures".*

Following the implementation of functional test cases (i.e., unit testing) that verify the correctness of the reference implementation (as defined in Deliverables D2.1, and Heartbeat H2.1), the perturbation analysis (Task 4.7) conducts testing campaigns to inject outlier test cases (including stress cases) and also a range of perturbations (on the different operational blocks and also the architectural data flows). The overall goal is to ensure the robustness of the final implementation (as reported in Heartbeats H2.2, H4.1 and finally in Deliverable D4.2) under those specific sets of tests i.e., the perturbation analysis does not aim for completeness. Furthermore, this document contributes a methodological approach to perform a perturbation analysis targeting the robustness of the system.

This deliverable reports the design and results of the range of perturbation tests and analyses conducted on the reference implementation[1]. The final reference implementation provided in D4.2 will include requisite countermeasures to provide resilience against the potential risks identified after the finalization of Task 4.7. These design/implementation enhancements are motivated by the results obtained in this document.

The rest of this section will introduce the necessary background related with the performed perturbation analysis (including framework and methodology). Subsequently Sections 2-6 present the actual results of the analysis organized according to the various elements of the privacy-ABC's life cycle. A detailed organization of the document appears in Section 1.6.

1.1 Base Concepts

This subsection reviews the basic terminology (as conventionally used in the dependability/security testing communities) needed to provide the background for understanding the rest of this document. For further details, the interested readers are referred to [1].

Robustness

Robustness refers to the implementation's correctness (in particular availability and integrity) in the presence of failures.

Perturbation

A perturbation is an event (e.g., related with some misuse or abuse of the system), that appears to have the potential to alter the system's delivery of correct operations i.e., affects the system's robustness.

Perturbation analysis (PA)

The main objective of the PA is to investigate how a system, or parts of a system, behave under anomalous (i.e., perturbed) operational conditions. A perturbation analysis is capable of demonstrating what sort of outputs a system produces under anomalous circumstances. Often a perturbation analysis will simulate scenarios that represent deviations from the system specification (also called "misuse cases[2]"). The common assumption is that these misuse cases have not been considered at design-time, and as such a corresponding reaction might not have been specified. Contrary to traditional functional testing (correctness) and penetration testing (where usually a stable architecture and typically the

[1] As mentioned in the Description of Work (DoW), the Perturbation Analysis presented in this document was applied only to the reference implementation, not to the pilot deployments (cf., WP6 and WP7).
[2] Further details about misuse cases will be presented in Section 1.2

source code level implementation details are needed), the primary target of a perturbation analysis is assessing the system's robustness (c.f. Figure 1). It is important to highlight that a perturbation analysis is, by itself, not a means of determining correctness, but primarily to evaluate if the system's robustness/fault-tolerant mechanisms actually work in containing the encountered perturbations.

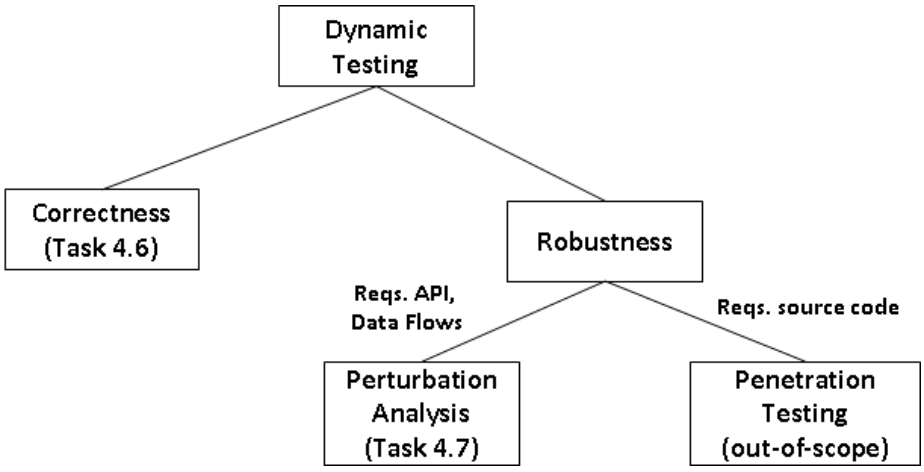


Figure 1: Dynamic Testing Classes

Perturbation Campaign

Perturbation analysis typically does not test individual misuse cases, but instead a set of them that are collectively regarded as a *perturbation campaign*. A campaign might target different elements of a system (cf. ET below).

Evaluation Target (ET)

ET refers to the specific element of the system (e.g., component, building block) that is being targeted by perturbation. The PA presented in this document will be applied to the architectural data flows (as designed by WP2) and interfaces (WP4), therefore allowing some degree of isolation from the continuous changes at the component's source code level. In general, it is expected for an ET exposed to a perturbation to observe a "fail-safe" behavior i.e., in the event that a failure is detected then the ET should respond in a way that it does not compromise the robustness of other modules. A typical "fail-safe" approach entails raising an exception and subsequently halting the operations at a pre-defined safe state to prevent the propagation of a failure to other modules. Halting of operations, as possible and specified in the system design, can entail literal stopping or switching to a pre-defined degraded form of operations. Alternate nuances of fail-safe involve raising flags that require user inputs to address the identified anomaly in order to proceed with execution.

1.2 Perturbation Analysis Framework

The perturbation analysis presented in this document is based on the framework shown in Figure 2 where an ET (from the reference implementation documented in Deliverable D2.1 and Heartbeat H2.1) is exposed to a perturbation based on the functional test cases developed in Task 4.6 (cf. Section 1.1) in order to assess the system's robustness. In the proposed framework, an ET is selected according to the following criteria:

⇒ First, by focusing on a particular stage of the privacy-ABC's life cycle (i.e. Setup, Issuance, Presentation, Revocation and Inspection as documented in Deliverable D2.1 [2] and Heartbeat H2.1 [7]).

⇒ Second, by selecting each one of the *(i)* flows taking place at the ABC4Trust architecture level (as defined by WP2), and *(ii)* components and interfaces of the ABCE.

The perturbations shown in this document will be based on a set of functional tests designed in Task 4.6, with the goal to assess the robustness of the reference implementation. While not part of the PA, some supplemental misuse observations as reported by the pilots are included in the Appendix A and Appendix B.

Utilizing this framework, it is possible to achieve a comprehensive approach that ensures that perturbations are being tested at all levels of the system: design, implementation and operational (including end-users) levels. It conceptually incorporates perturbations derived from system specifications on the different levels of abstraction during the construction of the system, as well as feedback from operational conditions anticipated from the pilot deployment. The results of the Perturbation Analysis are used by designers and developers in WP4 to improve the robustness of the final reference implementation (i.e., Deliverable D4.2 and Heartbeats H2.2 and H4.1).

This document adopts existing established perturbation frameworks (e.g., [1] and [2]) that target the ET's assessment of availability and integrity in the presence of failures (e.g., software or network-related). In the next section we present the methodological approach that implements the proposed PA framework.

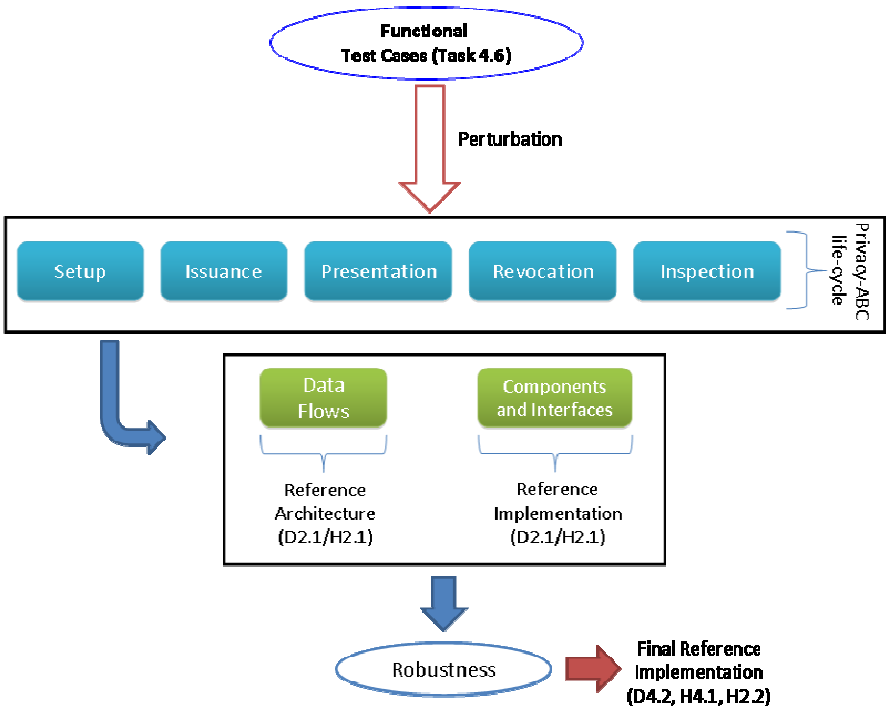


Figure 2: Perturbation Analysis Framework

1.3 Overview of the methodology

The methodology that implements the framework presented in the previous section consists of the steps illustrated in Figure 3. At a glance, the methodology starts by identifying the ET based on the framework (Figure 2), i.e., the privacy-ABC life-cycle and its associated flows/components and interfaces from Deliverable D2.1 and Heartbeat H2.1 (as discussed in Section 1.2). In Steps 2 – 4, the ET is classified so the corresponding data flow, component/interface or user-level perturbation campaign can be composed and applied. In Step 5 the results of the performed perturbation are analyzed to determine the needed corrective actions as well as the validation of any such correction. Take into account that designed corrective actions might involve changing the ET type (therefore, the arrow pointing from Step 6 to Step 2 in Figure 3).

Finally, in Step 6 the foreseen corrective actions are feedback to WP2 and WP4 for the design and development of the final reference implementation (Deliverable D4.2 and associated Heartbeats H2.2 and H4.1).

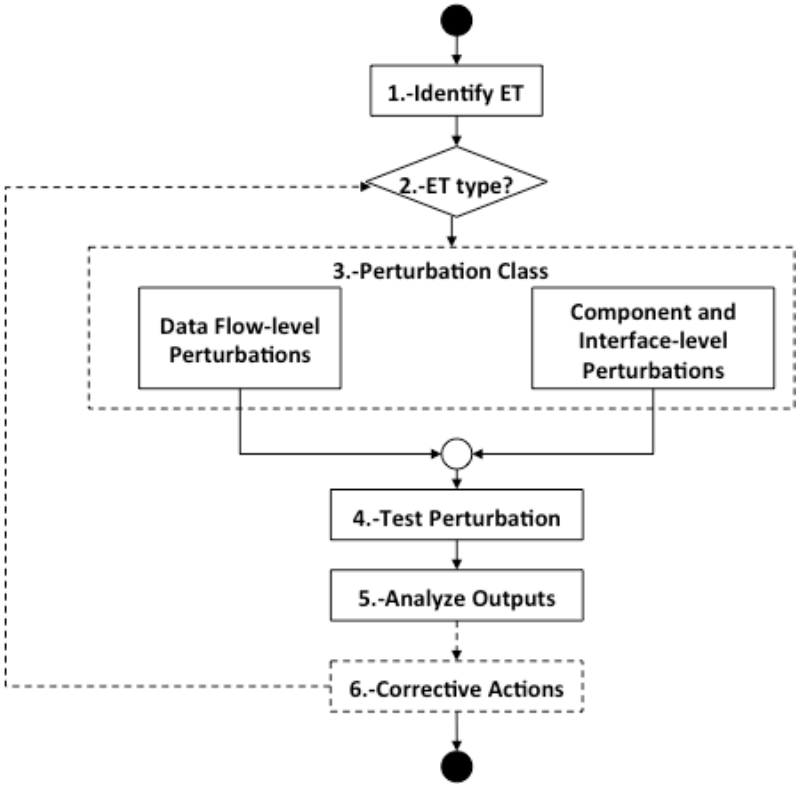


Figure 3: Perturbation Analysis Methodology

It is best-practice to document each perturbation as "misuse case scenarios", where design details are specified about the applied perturbations, observed results and even with respect to the mitigation/corrective actions that have been taken. At the state of the art there is no commonly agreed way to document a misuse case, therefore in Table 1 we propose a template to be used in the rest of this document. Such a template also gathers information to the test e.g., trying to reproduce the failure once a corrective action has been deployed.

Table 1: Misuse case scenario template

Scenario [no.]: Name of the misuse case scenario	
Summary	Short description of the scenario.
Evaluation Target (ET)	Describe the ET (e.g., interface name, specific data flow, …).
ET Type	Classify the ET as any of the following: 1 Architecture Data Flow (Arch). 2 Component/Interface (Comp).
Normal flow	Describe the normal (i.e., "correct") flow/usage of the ET.
Perturbation	Describe the perturbation to test in this misuse case (e.g., send to the Verifier a malformed token).
Perturbation Class	Classify the perturbation to test in any of the following (cf. Section 1.1): ⇒ Data flow-level: Outlier cases (DF-O), including stress cases (DF-S). ⇒ Component and Interface-level: Data-Type cases (C-DT) or Outlier cases (C-O).
Output old arch.	Document the output/result of the tested perturbation in the old crypto architecture (i.e., as documented in Deliverable D2.1 and Heartbeat H2.1). The result of a test can be any of: • *Compliant*: if its execution follows the documented specification i.e., detects the failure by triggering an exception, or, alternatively, if the observed behavior does not show any evidence of uncontrolled resource consumption. • *Non-compliant*: if its execution does not follow the specification i.e., the test does not detect a fail-safe and no exception is triggered. Or, alternatively, if the observed behavior shows evidence of uncontrolled resource consumption. • *Inconclusive:* if it cannot be determined if the specification was followed or not, possibly because the test's execution time exceeded a prefixed amount of time.
Output new arch.	Document the output/result of the tested perturbation in the new crypto architecture (i.e., as documented in Deliverable D4.2 and Heartbeat H4.1/H2.2). The result of a test can be any of: • *Compliant*: if its execution follows the documented specification i.e., detects the failure by triggering an exception, or, alternatively, if the observed behavior does not show any evidence of uncontrolled resource consumption. • *Non-compliant*: if its execution does not follow the specification i.e., the test does not detect a fail-safe and no exception is triggered. Or, alternatively, if the observed behavior shows evidence of uncontrolled resource consumption. • *Inconclusive:* if it cannot be determined if the specification was followed or not, possibly because the test's execution time exceeded a prefixed amount of time.

Mitigation/Corrective action	Document the action(s) taken to either mitigate or correct the observed fault (e.g., applied some specific patch to the Application Server). If the perturbation was correctly handled by the ET, then just document in this field the correctness of the implemented mechanism.

1.4 Detailed methodology

This section details the methodology depicted in Figure 3.

1.4.1 Step 1: Identify the ET

The PA starts by analyzing the whole system in order to identify the components that can compromise the overall robustness of the system. For example, applying a perturbation to an API call's URL parameter might result on a "URL not found" exception (which can be managed at run-time by the interpreter), however applying a perturbation to a security level parameter might result on malformed system parameters that can be used by other components and cause unexpected results.

1.4.2 Step 2: Classify the ET

After having identified the ET at Step 1, the analysis classifies the ETs into one of the available categories. The category will be used in the next step to select the adequate perturbation. The ET categories used by this document are:

1. *Arch*itecture flow e.g., issuing a credential from scratch.
2. *Comp*onent/Interface e.g., ABCE API's `initIssuanceProtocol()` method.

It is worth noting that PA at the *Comp* level will be focused on the ABCE API located underneath the context specific application, e.g., the User application or Issuer web application. This is a convenient point to inject faults before any cryptographic primitive is used (e.g., at the transport level). This approach (originally proposed by Nik [4], [5]) allows us to gain the control needed to apply the perturbations.

1.4.3 Step 3: Select a Perturbation Class

The classes of perturbation are selected from the list in Table 2. Each class defines a group of tests. Each test is designed starting from valid functional test cases, and then derived according to perturbation class. DF-S tests (a particular class of outlier cases DF-O) were derived by introducing sustained concurrent requests. DF-S tests will consider two parameters: the number of concurrent requests *k* and the time interval in second *t*. The test keeps *k* concurrent requests during a period of *t* seconds. C-DT and C-O tests are performed by selecting inputs over a set of invalid inputs. Invalid inputs are identified by combining the syntax and semantics of the API function parameters. The selection is done manually and by using a uniform distribution function.

Table 2: Perturbation Classes

ET Type	Perturbation Type	Comment/Example
Architecture data flow		
	Outlier Case (DF-O)	These are perturbations testing values that appear to deviate markedly from other members of the sample in which it occurs. DF-O includes stress cases

		Perturbations (DF-S) aimed towards taking a system to an extreme operation mode (close to its DoS border).
Component/Interface		
	Data Type (C-DT)	These perturbations test values that are valid for the type of parameter (e.g., -128 to 127 for Java's `byte` data type), but that are invalid for the specification. For example, typical DT perturbations [4] and [6] for an integer parameter include: param--, param++, 1, 0, -1, INT_MAX and INT_MIN. In Service Oriented Architectures, the use of DT perturbations is both useful and more efficient than other techniques (e.g., bit flipping) for testing fault tolerance mechanisms [4].
	Outlier (C-O)	As defined in DF-O, these are perturbations testing values that appear to deviate markedly from other members of the sample in which it occurs.

1.4.4 Step 4: Test Perturbation

After selecting the perturbation, the tests are executed against the ET. This may require access to the running system (physical/remote) and to the code (e.g., to increase log verbosity). In any case, the perturbation analysis should guarantee that the perturbation is repeatable under the conditions documented in the misuse case. In this document, the implementation of the designed perturbations is based on the test cases defined in Task 4.6.

Stress tests are executed within a finite amount of time. If the test does not produce an outcome within the time frame, then the test is considered *inconclusive*. The amount of time depends on the specification of the test (see, for example, Scenario 2.1.1). The results of a test execution are documented for each scenario using the template shown in Table 1.

1.4.5 Step 5: Analyze Outputs

During the test execution, the outputs are monitored and documented as part of the misuse case. This is a critical step, because corrective actions will be designed and deployed in the final version of the reference implementation (cf., Deliverable D4.2 and H4.1, H2.2) based on these observations.

Apart from the documented outputs of each individual use case, the results associated with the stress test cases are included in Appendix C.

1.4.6 Step 6: Take Corrective Actions

The final step consists of a set of actions to correct/mitigate the observed behaviors. This may require fixing the software and changing the system's specification. This step usually falls outside the scope of traditional perturbation analyses (cf., Voas [1] and Nik [4]).

1.5 Scope, limitations and testbed setup

Based on the Description of Work (DoW), the tests associated with the perturbation analysis documented were based on the first version of the reference implementation and architectural data flows presented in D2.1 [2] and Heartbeat H2.1 [7]. Such architecture (and its corresponding implementation) will be referenced as "old crypto architecture" in the rest of this document.

For the PA, both the data-flow level and component-level perturbations were designed focusing the ABCE (Attribute Based Credentials Engine) and those core components invoked by the API while performing the tests (including the Crypto Engines CE's). Where applicable, the designed tests took into consideration both of the Crypto Engines (i.e., Idemix and U-Prove). Given the DoW timeline for designing/conducting tests, those components without stable specifications based on their undergoing development for modification to the new crypto architecture (Deliverable D4.2 and Heartbeat H4.1/H2.2) were precluded from this testing. Nevertheless some testing was also conducted in the new architecture based on the architecture development schedule's match with the testing timeline[3].

Also as mentioned in the DoW, the obtained results and suggested corrective actions were considered as feedback for the development of the final reference implementation (cf., the new crypto architecture described in Deliverable D4.2 and Heartbeats H4.1 and H2.2). For the sake of completeness and comparability, all tests that resulted in NON-COMPLIANT results in the old crypto architecture were tested in the new crypto as well.

The tests reported in this deliverable were performed offline on local computers. The computer was a Pentium (R) E5700 3GHz with 4GB of RAM running Windows 7 Professional N SP1. Tests were performed on Java Virtual Machine 1.6.0, and Microsoft .NET 4.5.1.

Each PA test produced a log file and one or more Comma Separated Value (CSV) files. Log files and CSV files were processed in order to detect an expected fail-safe behavior (i.e. the generation of one or more Java exceptions) and to read the resource consumption measurements. CSV files associated with stress-test cases are included in the Appendix.

1.6 Organization of the document

The rest of this deliverable is organized following the stages of the privacy-ABC life-cycle and associated flows/components and interfaces. Section 2 shows the misuse cases and results considered during the setup phase of the privacy-ABC life-cycle. Section 3 presents the scenarios during the issuance phase where the user obtains privacy-ABC credentials from an issuer. Then, Section 4 shows the scenarios testing the ABCE during the presentation stage in which the user's access protected resources. Section 5 and 6 show the scenarios to test, respectively, the revocation and inspection phase. Finally, Section 7 summarizes the results of the analysis and concludes the deliverable.

While not part of the perturbation analysis, the functional bugs and the user-related failed installation instances reported by the WP6 and WP7 pilots are listed in Appendix A and B respectively. Appendix C shows the detailed CVS-formatted results of the stress test cases.

[3] Based on the DoW's schedule, the PA tests were designed from M25-M32 , whereas the initial specification of the new crypto was available at M36 (Heartbeats H2.2 and H4.1). The final version of the reference implementation will be available at M45 (Deliverable D4.2).

2 Setup

As described in Deliverable 2.1 [2], during the setup stage "the ABCE layer provides for each party a dedicated method to obtain its private and public (if any) cryptographic parameters. Private keys will be stored in the trusted storage of the corresponding party". Also according to these documents, the API calls involved in the setup stage are: `setupSystemParameters()`, `setupIssuerParameter()`, `setupRevocationAuthorityParameter()` and `setupInspectionPublicKey()`.

As mentioned in Section 1.5, the perturbation analysis was focused on the data flows and interfaces documented in Deliverable D2.1., i.e., the ABCE and both crypto engines (where applicable).

2.1 Data flow-level perturbations

The following section presents the data flow-level misuse cases considered for the Setup stage of the privacy-ABC life-cycle. The presented scenario was selected and designed to test the robustness of the data flows involved during the Setup stage. In particular this section focuses on testing the ABCE under stress conditions (targeting the consumption of available resources, possibly due to a denial of service), which is one of the most common cases found on the relevant literature [1].

Scenario 2.1.1: Stress perturbations on the ABCE component	
Summary	A stress perturbation is tested on the ABCE to assess its resilience against a denial of service (DoS).
Component Under Evaluation (ET)	ABCE – Setup (All Entities)
ET Type	Arch
Normal flow	The ABCE component supports concurrent setup requests, which are triggered by the respective `setup*` ABCE API calls [1].

Perturbation	Stress the ABCE component by keeping *k* concurrent setup requests during a period of *t* seconds. The goal is to monitor the resource consumption (i.e., heap memory consumption) and the availability of the service (i.e., time to process all the requests). *When applicable, this experiment should be repeated for all involved CEs.*
	The parameters *k* and *t* are the following:
	• k ∈ {500, 1000, 1500};
	• t ∈ {60, 120, 180}.
	The pseudo-algorithm is the following:
	`1 - Until time t is reached:`
	`2 - init = freeMemory();`
	`3 - Execute k concurrent requests;`
	`4 - mem = freeMemory() - init;`
	`5 - Log mem, t, and k`

	Where `freeMemory()` is the API call to obtain the amount of system memory available[4]. Document the outputs and assess the consumption of the resource for (a) Idemix and (b) U-Prove.
Perturbation Class	DF-S (subclass of DF-O)
Output old arch.	**RESULTS**: (a) Compliant, (b) Non-compliant. **DETAILS:** This test does not aim at detecting a fail-safe behavior; it rather monitors the implementation under a constant number of concurrent requests over different time intervals. In this section, we provide more data to detail the conclusions for (a) and (b) We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of memory when testing SetupSystemParameters(), SetupIssuerParameters(), SetupRevocationAuthorityParameters(), SetupInspectionPublicKey(). **(a) Idemix** Below, we provide the PPMCC indexes respectively for SetupSystemParameters(), SetupIssuerParameters(), SetupRevocationAuthorityParameters(), SetupInspectionPublicKey(): • The PPMCC between *t* and the memory usage are 0.12, 0.25, -0.37, and 0.19; • The PPMCC between *k* and the memory usage are -0.41, -0.37, -0.10, and 0.29; The results show that the memory usage is positively, yet poorly, correlated with *t* and that there is mainly a more substantial negative correlation between the number of sustained parallel requests and the memory usage. An example of memory consumption for SetupSystemParameters() is shown in Figure 4. The x-axis is the duration of the test *t* in seconds, while the y-axis is the amount of memory used by SetupSystemParameter() in MB. The three functions represent the memory consumed by SetupSystemParameter() Idemix when executing concurrently 500, 1000, or 1500 concurrent requests. The memory usage of Idemix 500 tends to decrease over the time while Idemix 1000 presents the opposite behavior. Finally, Idemix 1500 has a consumption peak of 250 at 120s, while at 60s and 180s the memory usage is in the range of 100-120 MB. The variation of correlation between the memory consumed, and the parameters t and k suggests that t and k are not significantly influencing the consumption of memory in Idemix. The variation of used memory can be explained by an efficient object dereference by the implementation and then by the execution of the Java garbage collector. **(b) U-Prove** We tested only SetupSystemParameters() and SetupIssuerParameters(). The other functions were not tested because UProve does not support revocation and inspection. The PPMCC indexes, respectively, for SetupSystemParameters() and SetupIssuerParameters() are: • The PPMCC between *t* and the memory usage is 0.13, and -0.39; • The PPMCC between *k* and the memory usage is 0.74, and -0.70; Figure 5 shows the memory consumption of SetupSystemParameters() over the time.

[4] The real implementation used the java.lang.Runtime API . See the following URL for more details: http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html

	The x-axis is the number of concurrent requests *k*, while the y-axis is the amount of memory consumed by SetupSystemParameters(). The three functions plot the memory used when performing concurrent requests for 60 seconds, 120 seconds, and 180 seconds. Figure 5 shows the positive correlation between the number of concurrent requests and the memory consumed, in which an increasing number of concurrent requests cause the system to use more memory. If an attacker can control this behavior, e.g., by sending forged requests, then U-Prove may exhaust the available resources. The results for SetupIssuerParameters() show a negative correlation between the memory usage and the parameters *t* and *k*.
Output new arch.	**RESULTS**: (a) N/A, (b) Non-compliant. **DETAILS:** This test does not aim at detecting a fail-safe behavior; it rather monitors the implementation under a constant number of concurrent requests over different time intervals. In this section, we provide more data to detail the conclusions for (a) and (b) We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of memory when testing SetupSystemParameters(), SetupIssuerParameters(), SetupRevocationAuthorityParameters(), SetupInspectionPublicKey(). **(a) Idemix** Test case (a) was not executed against the new architecture. **(b) U-Prove** We tested only SetupSystemParameters() and SetupIssuerParameters(). The other functions were not tested because U-Prove does not support revocation and inspection. • The PPMCC between *t* and the memory usage is 0.83, and 0.69; • The PPMCC between *k* and the memory usage is -0.38, and -0.13. The results suggest that increasing number of concurrent requests have a negative correlation with the amount of memory used. However, longer time interval in which the concurrent requests are performed influences the memory consumption.
Mitigation/Corrective action	None

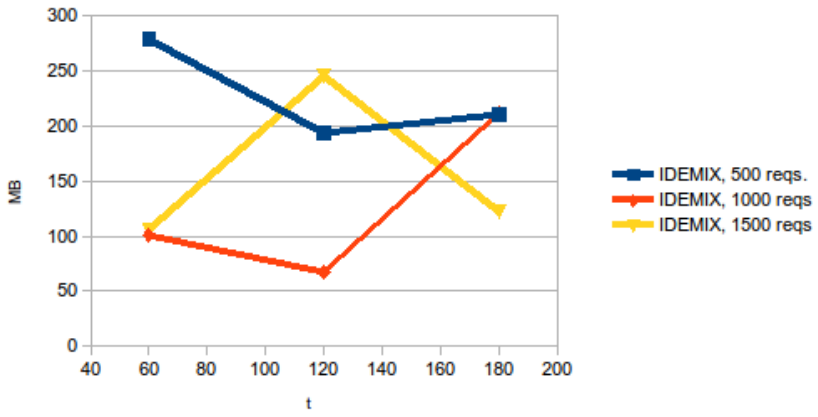


Figure 4: Memory Consumption of SetupSystemParameters() with Idemix.

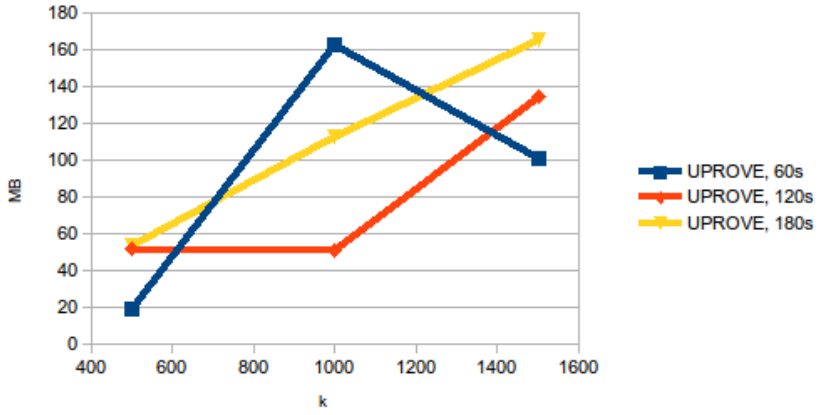


Figure 5: Memory Consumption of SetupSystemParameters() with U-Prove.

2.2 Component and Interface-level perturbations

In this section are presented the misuse case scenarios related with the ABCE API calls used during the Setup stage, namely `setupSystemParameters()`, `setupIssuerParameter()`, `setupRevocationAuthorityParameter()` and `setupInspectionPublicKey()`.

As mentioned in Section 1.2, the perturbation tests are mostly focused on those function parameters that can compromise the robustness of the overall system. For example, while we considered testing the system's resilience against a perturbation affecting the value of the `securityLevel` parameter of the `setupSystemParameters` call (which might propagate to the CE component and crash/corrupt it), we did not consider as critical a perturbation that changes the value of the `cryptoMechanism` parameter of the same call (it might only allow linking to an invalid URI, without any major security compromise).

Scenario 2.2.1: Data-type perturbations to the `securityLevel` parameter in the `setupSystemParameters()` call	
Summary	This perturbation aims to test the robustness of the API call that sets up the system parameters, by using values that fall outside the specification of the `securityLevel` parameter.
Component Under Evaluation (ET)	ABCE API Call - `setupSystemParameters()` – Issuer
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `SystemParameters setupSystemParameters(int securityLevel, URI cryptoMechanism)` Security levels 80 and 128 MUST be supported (i.e., this is part of the functional/correctness testing performed by Task 4.6), other values MAY also be supported [2].

Perturbation	This test considers the following two parameter perturbations: a) *Perturbation of* `securityLevel` *parameter values*: All the negative Java integer values fall outside of the `securityLevel` specification. The selection criteria is a uniform distributed random function that selects 500 negative Java integers and 500 supported cryptoMechanism. In order to reproduce the experiment, we store the specific input used in the logs. b) *Perturbation of* `cryptoMechanism` *parameter values*: The test generates 10 well-formed (i.e., syntactically correct) random URI objects, e.g., urn:qro3D:vykQviPwps:BAFBn3kJVD:A5klCQNuuQ Document the outputs and observe the correctness of the results/implemented exception catching mechanisms.
Perturbation Class	C-DT
Output old arch.	**RESULT:** (a) Compliant, (b) Compliant **DETAILS:** **(a) Perturbation of `securityLevel`** The method "testIssuer" in class "eu.abc4trust.abce.perturbationtests.section2.Test21" encountered an error to create "SystemParameters" with "constraint 3" on Idemix by assigning a invalid value. The same error recurred on U-Prove. **(b) Perturbation of `cryptoMechanism`** The method "runTest" in class "eu.abc4trust.abce.perturbationtests.section2.Test21" encountered an error to create "IssuerParameters" with "bit length constraint" and an error to create "SystemParameters" with "Unsupported security level" on Idemix by assigning a invalid value. The same error recurred on U-Prove by assigning an invalid value.
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 2.2.2: Perturbation of the `credspec` parameter in the `setupIssuerParameters()` call	
Summary	This misuse case aims to test the robustness of the API call that sets up the Issuer's parameters, by adding perturbations to its `credspec` parameter.
Component Under Evaluation (ET)	ABCE API Call - `setupIssuerParameters()` – Issuer
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `IssuerParameters setupIssuerParameters(CredentialSpecification credspec, SystemParameters syspars, URI uid, URI hash, URI revParsUid)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the `setupIssuerParameters` call, by implementing as individual tests the

	following perturbations to the `credspec` parameter: a) Set /`abc:CredentialSpecification/@Revocable` to `true`, but do not specify a revocation handle. b) Create a credential specification with a duplicated value in `/abc:CredentialSpecification/abc:SpecificationUID` c) Select randomly 655 values in {0, …, 65535} for /`abc:AttributeDescriptions/abc:AttributeDescription/@MaxLength`. This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**: (a) Compliant, (b) Compliant, (c) Non-compliant **DETAILS**: **(a) `Revocable` to `true`** The method "presentIDCard" in class "eu.abc4trust.abce.perturbationtests.section2.Test22" encountered an error to verify the presentation token with "not valid" by "eu.abc4trust.exceptions.TokenVerificationException". **(b) duplicate `SpecificationUID`** The method "issueIDCard" in class "eu.abc4trust.abce.pertubationtests.section2.Test22" encountered an error to issue credential with "null". **(c) random `MaxLength`** The test case (c) failed because upon providing a randomly chosen value, it did not observe a fail-safe behavior of the component. The sequence of events produced by the component are: 1. Setting CredentialSpecification AttributeDescriptions.MaxLength = *x* 2. Used IssuerParameters to issue a credential 3. Successfully created a presentation token 4. Successfully verified presentation token Where x is a randomly chosen value of the domain {0, …, 65535}. Note: The test expects and exception at step 1 when the actual call to `setupIssuerParameters` is performed. If the exception is not raised, then the test attempt to use the parameters for issuance/presentation and verify if that works.

Output new arch.	**RESULT**: (a) N/A, (b) N/A, (c) Compliant **DETAILS**: **(a) `Revocable` to `true`** This test case was not executed against the new architecture as the result in the old architecture was Compliant. **(b) duplicate `SpecificationUID`** This test case was not executed against the new architecture as the result in the old architecture was Compliant. **(c) random `MaxLength`** The new crypto arch produces the following exception: "com.ibm.zurich.idmx.exception.ProofException: Incorrectly re-computed NValue: sig:0:cs:credSpec:c14n"
Mitigation/Corrective action	None

Scenario 2.2.4: Perturbation of the `uid` parameter in the `setupIssuerParameters()` call	
Summary	This misuse case aims to test the robustness of the API call that sets up the Issuer's parameters, by adding perturbations to its `uid` parameter.
Component Under Evaluation (ET)	ABCE API Call - `setupIssuerParameters()` – Issuer
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `IssuerParameters setupIssuerParameters(CredentialSpecification credspec, SystemParameters syspars, URI uid, URI hash, URI revParsUid)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the `setupIssuerParameters` call, by generating parameters with **the same** `uid`. This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**:Compliant **DETAILS**: The implemented test encountered an error to issue credential using first set of IssuerParameters by "Incorrect issuer public key for "credentialToBeIssued".
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 2.2.5: Data type perturbations to the `securityLevel` parameter in the `setupRevocationAuthorityParameters()` call	
Summary	This perturbation aims to test the robustness of the API call that sets up the RA parameters, by applying a set of data type perturbations to the `securityLevel`.
Component Under Evaluation (ET)	ABCE API Call - `setupRevocationAuthorityParameters()` – Revocation Authority
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `RevocationAuthorityParameters setupRevocationAuthorityParameters(int securityLevel, URI cryptoMechanism, URI uid, RevocationInfoReference infoRef, NonRevocationEvidenceReference evidenceRef, RevocationUpdateReference updateRef)` Security levels 80 and 128 MUST be supported (i.e., this is part of the functional/correctness testing performed by Task 4.6), other values MAY also be supported [2].
Perturbation	Test the following values, which fail outside of the `securityLevel` specification: Select randomly 500 negative Java integers for the `securityLevel` and 500 supported `cryptoMechanism`. This feature in implemented only in Idemix.
Perturbation Class	C-DT
Output old arch.	**RESULTS**: Compliant. **DETAILS**: The method “issueIDCard” in class“eu.abc4trust.abce.pertubationtests.section2.Test25” encountered an error as “Failed to issue credential” with “null”.
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 2.2.6: Perturbation of the `uid` parameter in the `setupRevocationAuthorityParameters()` call	
Summary	This misuse case aims to test the robustness of the API call that sets up the Issuer’s parameters, by adding perturbations to its `uid` parameter.
Component Under Evaluation (ET)	ABCE API Call - `setupRevocationAuthorityParameters()` – Issuer
ET Type	Comp

Normal flow	As documented in H2.1 the specification of this API call is: `RevocationAuthorityParameters setupRevocationAuthorityParameters(int securityLevel, URI cryptoMechanism, URI uid, RevocationInfoReference infoRef, NonRevocationEvidenceReference evidenceRef, RevocationUpdateReference updateRef)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the setupRevocationAuthorityParameters call, by executing it two or more times in order to generate two or more parameters with **the same** `uid`. This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**: Compliant **DETAILS**: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section2.Test26" encountered an error of "Failed to issue credential using the second set of revocation parameters".
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 2.2.7: Use of NULL values in the URI parameters of the `setupRevocationAuthorityParameters()` call	
Summary	This misuse case aims to test the robustness of the API call that sets up the Issuer's parameters, by adding perturbations to the parameters of type URI.
Component Under Evaluation (ET)	ABCE API Call - `setupRevocationAuthorityParameters()` – Issuer
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `RevocationAuthorityParameters setupRevocationAuthorityParameters(int securityLevel, URI cryptoMechanism, URI uid, RevocationInfoReference infoRef, NonRevocationEvidenceReference evidenceRef, RevocationUpdateReference updateRef)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the `setupIssuerParameters` call, by setting (as individual tests) the following URI type parameters: a) `cryptoMechanism` set to null b) `uid` set to null c) Select randomly 10 invalid URI paramters for `cryptoMechanism`. This feature in implemented only in Idemix.
Perturbation Class	C-DT

Output old arch.	**RESULTS**:
	(a) Non-compliant, (b) Compliant, (c) Compliant
	DETAILS:
	(a) `cryptoMechanism` set to null
	The test case (a) failed because upon providing a null crypto mechanism object, it did not observe a fail-safe behavior of the component. The sequence of events produced by the component are:
	1. Successfully produced parameters, now trying to create IssuerParameters
	2. Managed to issue a credential
	3. Successfully created a presentation token
	4. Succesfully verified presentation token
	5. Used Revocation AuthorityParameters to create a valid presentation token
	Additional tests may be performed to verify whether an exception is raised in latter phases.
	(b) `uid` set to null
	The test case (b) observed a `NullPointerException`. This exception is not reported in the log files because, according to the test case developer, "*NullpointerExceptions dont have a message field, hence a null value is logged as the message*".
	(c) Select randomly 10 invalid URI paramters for `cryptoMechanism`
	The test case (c) observed the same exception
	`Failed to issue credential :`
	`java.lang.NullPointerException`
	This exception that is raised within the method `issueIDCard` of class "eu.abc4trust.abce.pertubationtests.section2.PA_II_2_2_7randomUID".

Output new arch.	**RESULTS**: (a) Compliant, (b) N/A, (c) N/A **DETAILS**: **(a) `cryptoMechanism` set to null** The new crypto arch produces: "com.ibm.zurich.idmx.exception.ConfigurationException: Idemx: Technology for creating issuer parameters is not supported. Exception" **(b) `uid` set to null** This test case was not executed against the new architecture as the result in the old architecture was Compliant. **(c) Select randomly 10 invalid URI paramters for `cryptoMechanism`** This test case was not executed against the new architecture as the result in the old architecture was Compliant.
Mitigation/Corrective action	Idemix was/is the only crypto engine supported by ABC4Trust that provides revocation, so the ABCE is hardcoded to use it. Therefore crypto mechanism was/is not used.

Scenario 2.2.8: Data-type perturbations to the `securityLevel` parameter in the `setupInspectorPublicKey()` call	
Summary	This perturbation aims to test the robustness of the API call that sets up the Inspector's public key, by applying a set of data type perturbations to the `securityLevel`.
Component Under Evaluation (ET)	ABCE API Call - `setupInspectorPublicKey()` – Inspector
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `InspectorPublicKey setupInspectorPublicKey(int securityLevel, URI mechanism, URI uid)` Security levels 80 and 128 MUST be supported (i.e., this is part of the functional/correctness testing performed by Task 4.6), other values MAY also be supported [2].
Perturbation	Test the following values, which fail outside of the `securityLevel` specification: Select randomly 1000 negative Java integers for the `securityLevel` and 1000 supported `cryptoMechanism`. This feature in implemented only in Idemix.
Perturbation Class	C-DT
Output old arch.	**RESULT**: Non-compliant **DETAILS**: The component does not generate an exception. The log file shows that all tests reproduced the following sequence of events: 1. Running test with security level *x* (x is the negative value) 2. Successfully produced inspector key, now trying to create IssuerParameters 3. Managed to issue a credential 4. Successfully created a presentation token 5. Successfully verified presentation token
Output new arch.	The `securityLevel` parameter is replaced by a `SystemParameters` object in the new crypto architecture, so this scenario is no longer relevant.
Mitigation/Corrective action	The `securityLevel` is actually read from the `SystemParameters`, located in the Inspectors KeyManager. The Parameters is therefore never actually used.

Scenario 2.2.9: Perturbation of the `uid` parameter in the `setupInspectorPublicKey()` call	
Summary	This misuse case aims to test the robustness of the API call that sets up the Inspector's public key, by adding perturbations to its `uid` parameter.
Component Under Evaluation (ET)	ABCE API Call - `setupInspectorPublicKey()` – Inspector

ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `InspectorPublicKey setupInspectorPublicKey(int securityLevel, URI mechanism, URI uid)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the `setupInspectorPublicKey` call, by generating parameters with **the same** `uid`. This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**: Compliant **DETAILS**: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section2.Test29" encountered an error of "Failed to inspect using the first key".
Output new arch.	N/A
Mitigation/Corrective action	None

3 Issuance

When issuing a credential, an interactive protocol is executed between the User and an Issuer, where at the end the User obtains a privacy-ABC credential (or an error message) [2]. Issuance can be "from scratch" (i.e., credentials are issued without relation to any existing credentials or pseudonyms already owned by the Users) or "advanced issuance" (i.e., the information embedded into the newly created credential can be invisibly "carried over" from existing credentials already owned by the User).

As mentioned in Section 1, the designed/tested perturbations focused on the core components documented in D2.2 i.e., the Issuer ABCE and the involved CEs.

Also, regardless of the issuance type (i.e., from scratch or advanced) the API calls involved during this stage are the Issuer's `initIssuanceProtocol()` and, both Issuer's and User's `issuanceProtocolStep()`. The reader is referenced to Deliverable 2.1 [2], for further details about the credential issuance stage.

3.1 Data flow-level perturbations

Misuse cases related with data flow-level perturbations are developed in the rest of this section. Notice that despite the fact that Issuance is a multi-legged protocol, the involved Issuance-related API calls are stateless despite a "unique context" is kept among different messages from the same issuance process to ensure that the Issuer is serving the right request.

Scenario 3.1.1: Stress perturbations on the ABCE component (Advanced Issuance with no carry over attributes)	
Summary	A stress perturbation is tested on the ABCE when issuing a credential with advanced features, to assess its resilience against a denial of service (DoS).
Component Under Evaluation (ET)	ABCE – Advanced Issuance - Issuer
ET Type	Arch
Normal flow	The ABCE component supports concurrent issuance with advanced features requests, which are triggered by the respective `issuanceProtocolStep()` ABCE API call [2].

Perturbation	Stress the ABCE component by keeping *k* concurrent issuance requests during a period of *t* seconds. The goal is to monitor the resource consumption (i.e., heap memory consumption). *When applicable, this experiment should be repeated for all involved CEs.*
	The parameters *k* and *t* are the following:
	• k ∈ {500, 1000, 1500};
	• t ∈ {60, 120, 180}.
	The pseudo-algorithm is the following:
	`1 – Until time t is reached:`
	`2 – init = freeMemory();`
	`3 – Execute k concurrent requests;`

	`4 – mem = freeMemory() – init;`
	`5 – Log mem, t, and k`
	Where `freeMemory()` is the API call to obtain the amount of system memory available[5].
	The test consists of two phases: a preprocessing phase and a test phase. The preprocessing phase starts by making *k* serial calls to the Issuers initIssuanceProtocol, giving *k* IssuanceMessage (IM1). These *k* IM1 are then passed to the user ABCE (again serially), yielding *k* new IssuanceMessages (IM2). The test phase uses the *k* IM2, which are each assigned to a separate thread that calls `issuanceProtocolStep()` on the issuer. These threads are run in parallel.
	Document the outputs and assess the consumption of the resource for (a) Idemix and (b) U-Prove.

Perturbation Class	DF-S (subclass of DF-O)
Output old arch.	**RESULT**: (a) Compliant, (b) Non-compliant **DETAILS**: We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of resources. **(a) Idemix** • The PPMCC between *t* and the memory usage is 0.5; • The PPMCC between *k* and the memory usage is 0.19; The results show a correlation between the memory usage and the time. However, the maximum memory usage peak is 60MB **(b) U-Prove** The test execution reached the timeout set for these tests. U-Prove becomes unresponsive after a set amount of requests have been sent. U-Prove has had some issues with hardcoded (artificial) limitations.
Output new arch.	**RESULT**: (a) N/A, (b) Compliant **DETAILS**: We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of resources. **(a) Idemix** This test case was not executed against the new architecture as the result in the old architecture was Compliant. **(b) U-Prove** • The PPMCC between *t* and the memory usage is 0.48; • The PPMCC between *k* and the memory usage is 0.60; The results show a correlation between the memory usage and the time. However, the maximum memory usage peak is 49MB

[5] The real implementation used the java.lang.Runtime API . See the following URL for more details: http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html

Mitigation/Corrective action	None

Scenario 3.1.2: Malformed Advanced Issuance's parameters	
Summary	This perturbation consists of an outlier case for Advanced Issuance, where a list of attributes to be carried over does not correspond to the respective Issuance Policy sent by the Issuer.
Component Under Evaluation (ET)	ABCE - Advanced Issuance – Issuer
ET Type	Arch
Normal flow	When issuing a credential with carried over attributes (i.e., Advanced Issuance [2]), an Issuance Policy containing a credential template specifying the existing user credential's attributes to "reuse" is sent by the Issuer.
Perturbation	The credential template used by the Issuer will contain an empty list of user attributes to reuse in the requested credential; however the respective Issuance Policy should be non-empty in order to trigger the Advanced Issuance. This feature in implemented only in Idemix.
Perturbation Class	DF-O
Output old arch.	**RESULT**: Compliant **DETAILS**: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section3.Test12" encountered an error of "Failed to issue credential: Proof does not verify".
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 3.1.3: Modifying the Issuance Message's Context attribute	
Summary	This perturbation consists of an outlier case for Advanced Issuance, where the message's context attribute changes to a non-existing session. The perturbation is done at the user-side of the issuance protocol.
Component Under Evaluation (ET)	ABCE - Advanced Issuance – Issuer
ET Type	Arch
Normal flow	To allow the linkage of different legs of an issuance protocol, each message includes a Context attribute, which must have the same value on all legs [2].
Perturbation	Change the value of the Context attribute with 10 non-existing issuance sessions. The selection of the 10 sessions is done randomly. This is only implemened in Idemix.

Perturbation Class	DF-O
Output old arch.	**RESULT**: Compliant **DETAILS**: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section3.Test13" encountered an error of "Failed to issue credential: java.lang.NullPointerException".
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 3.1.4: Modifying the Issuance Message's Context attribute	
Summary	This perturbation consists of an outlier case for Advanced Issuance where the message's context attribute changes to an existing, but different from the current one, issuance session.
Component Under Evaluation (ET)	ABCE - Advanced Issuance - Issuer
ET Type	Arch
Normal flow	To allow the linkage of different legs of a issuance protocol, each message includes a Context attribute, which must have the same value on all legs [2].
Perturbation	Change the value of the Context attribute to an existing issuance session, but which is different from the current one. That is, Client 1 initiates issuance with server (gets issuance policy with context=AA in return), but never responds. Then Client 2 initiates issuance with server (gets issuance policy with context=BB in return), but in this case client 2 computes a response to the received issuance policy using context=AA (from Client 1's). This feature in implemented only in Idemix.
Perturbation Class	DF-O
Output old arch.	**RESULT**: Compliant **DETAILS**: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section3.Test14" encountered an error of "Failed to issue credential : Proof does not verify".
Output new arch.	N/A
Mitigation/Corrective action	None

3.2 Component and Interface-level perturbations

In this section the misuse case scenarios related with the ABCE API calls used during the Issuance stage, namely `issuanceProtocolStep()` and `initIssuanceProtocol()` are presented. Where applicable, the calls are analyzed for both Users and Issuers. For the presented misuse cases, the same considerations apply as for those shown in Section 2.

Notice that perturbations to the `PresentationPolicy` embedded into the `IssuancePolicy`, will be presented in Section 4.

Scenario 3.2.1: Perturbing the initIssuanceProtocol()call (Issuers)	
Summary	This perturbation aims to test the robustness of API call that initiates the interactive issuance protocol (Issuers).
Component Under Evaluation (ET)	ABCE API Call - `initIssuanceProtocol()` – Issuer
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `(IssuanceMessage, boolean, URI) initIssuanceProtocol(IssuancePolicy ip, Attribute[] attributes)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms, by implementing as individual tests the following perturbations: a) Create an issuance policy with a non-existing value in `/abc:IssuancePolicy/abc:CredentialTemplate/abc:CredentialSpecUID` b) Create an issuance policy with a non-existing value in `/abc:IssuancePolicy/abc:CredentialTemplate/abc:IssuerParametersUID` c) Create an issuance policy with a set of issuer parameters that are not meant for the credspec specified in the template. That is, not matching (although valid) `/abc:IssuancePolicy/abc:CredentialTemplate/abc:CredentialSpecUID` and `/abc:IssuancePolicy/abc:CredentialTemplate/abc:IssuerParametersUID` d) Specify a set of non-existing attributes to be carried over from existing credentials (i.e., `…/abc:UnknownAttributes/abc:CarriedOverAttribute/abc:SourceCredentialInfo`). e) Create 10 randomly generated malformed attribute type-value pairs (i.e., the `attributes` parameter) This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**: (a-e) Compliant

	DETAILS: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section3.Test21" encountered the following exceptions: a) "Failed to issue credential : Could not find credential description with UID: "my:random:uri" b) "Failed to issue credential : java.lang.RuntimeException:" c) "java.lang.RuntimeException: Cannot find object my:random:uri located at my:random:uri" d) "Failed to issue credential : java.lang.RuntimeException: java.lang.NullPointerException" e) "Failed to issue credential : this is text, not a date"
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 3.2.2: Perturbing the issuance policy within the `IssuanceMessage` parameter of the `issuanceProtocolStep()` call (Users)	
Summary	This perturbation aims to test the robustness of the issuance policy (embedded into the `IssuanceMessage` parameter) used by the API call that performs one step in the interactive issuance protocol (for both Users and Issuers).
Component Under Evaluation (ET)	ABCE API Call - `issuanceProtocolStep()` – User
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: For Users: `IssuanceMessage/CredentialDescription issuanceProtocolStep(IssuanceMessage m)` Please note that the `issuanceMessages` sent to the issuer will never contain an issuance policy, hence the perturbation is not relevant for issuers.
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the `issuanceProtocolStep` call, by implementing as individual tests the following perturbations to the issuance policy contained into the `IssuanceMessage` parameter: a) Create an issuance policy with a duplicate (i.e., existing/non-unique) value in `/abc:IssuancePolicy/abc:CredentialTemplate/abc:CredentialSpecUID` b) Create an issuance policy with a duplicated (i.e., existing/non-unique) value in `/abc:IssuancePolicy/abc:CredentialTemplate/abc:IssuerParametersUID` c) Specify a set of non-existing attributes to be carried of from existing credentials (i.e., `…/abc:UnknownAttributes/abc:CarriedOverAttribute/abc:SourceCredentialInfo`).

	This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**: (a-c) Compliant **DETAILS**: The method "runTest" in class "eu.abc4trust.abce.pertubationtests.section3.Test22" encountered the following exceptions: a) "Failed to issue credential : java.lang.RuntimeException: cannot extract cred spec" b) "Failed to issue credential : java.lang.NullPointerException" c) "Failed to issue credential : java.lang.RuntimeException: java.lang.NullPointerException"
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 3.2.3: Perturbing the issuance token within the `IssuanceMessage` parameter of the issuer ABCE API Call `issuanceProtocolStep()` call	
Summary	This perturbation aims to test the robustness of the issuance token (embedded into the `IssuanceMessage` parameter) used by the issuer API call that performs one step in the interactive issuance protocol.
Component Under Evaluation (ET)	ABCE API Call - `issuanceProtocolStep()` – User
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: For Issuers: `(IssuanceMessage, boolean, URI) issuanceProtocolStep(IssuanceMessage m)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms in the `issuanceProtocolStep` call by specifying a token with version (i.e. `/abc:IssuanceToken/@Version`) different to "1.0".
Perturbation Class	C-O

Output old arch.	**RESULT**: Non-compliant **DETAILS**: The test failed because upon using a token version different to 1.0, the test did not observe fail-safe behavior. The sequence of events produced by the component are: 1. Managed to issue a credential 2. Successfully created a issuance token 3. Successfully verified issuance token Additional tests may be performed to verify whether an exception is raised in latter phases.
Output new arch.	**RESULT**: Non-compliant **DETAILS**: The new crypto architecture has the same output.
Mitigation/Corrective action	The version is never checked. It should be either removed from the specification or implemented a check.

4 Presentation

As described in Deliverable 2.1 [2], during the presentation stage the user requests access to a protected resource, upon which the verifier sends a presentation policy describing which credentials the User should present to obtain access.

The API calls to be tested are `createPresentationToken()` and `verifyTokenAgainstPolicy()`. Interested readers are referenced to Deliverable 2.1 [2], for further details about the token presentation stage.

4.1 Data flow-level perturbations

This section presents the misuse case (and its corresponding results) targeting the Presentation stage of the privacy-ABC. In particular, the test presented in this section focuses on stressing the ABCE API call in charge of performing the token verification, by executing a high number of concurrent requests (which resembles a perturbation that can occur on a real deployment).

Scenario 4.1.1: Stress perturbations on the verifier's ABCE component	
Summary	A stress perturbation is tested on the verifier's ABCE, to assess its resilience against a denial of service (DoS).
Component Under Evaluation (ET)	ABCE – Verifying a presentation token - Verifier
ET Type	Arch
Normal flow	The ABCE component has been designed to support concurrent verification requests, which are triggered by the respective `verifyTokenAgainstPolicy()` ABCE API call [2].

Perturbation	Stress the ABCE component by keeping *k* concurrent verification requests during a period of *t* seconds. The goal is to monitor the resource consumption (i.e., heap memory consumption) and the availability of the service (i.e., time to process all the requests). *When applicable, this experiment should be repeated for all involved CEs.*
	
	The parameters *k* and *t* are the following:
	• k ∈ {500, 1000, 1500};
	• t ∈ {60, 120, 180}.
	
	The pseudo-algorithm is the following:
	`1 - Until time t is reached:`
	`2 - init = freeMemory();`
	`3 - Execute k concurrent requests;`
	`4 - mem = freeMemory() - init;`
	`5 - Log mem, t, and k`

	Where `freeMemory()` is the API call to obtain the amount of system memory available[6]. Document the outputs and assess the consumption of the resource for (a) Idemix and (b) U-Prove.
Perturbation Class	DF-S (subclass of DF-O)
Output old arch.	**RESULT**: (a) Compliant; (b) Non-Compliant **DETAILS**: We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of resources. **(a) Idemix** • The PPMCC between *t* and the memory usage is -0.06; • The PPMCC between *k* and the memory usage is -0.19; Memory usage has a negative correlation with k and t. This suggests that the component does not suffer from uncontrolled memory usage with regard to k and t. **(b) U-Prove** The test execution reached the timeout set for these tests. U-Prove becomes unresponsive after a set amount of requests have been sent (this is the same problem as in 4.1.1). U-Prove has had some issues with hardcoded (artificial) limitations
Output new arch.	**RESULT**: (a) Compliant; (b) Non-Compliant **DETAILS**: We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of resources. **(a) Idemix** The test case (a) was not executed against the new architecture because the result against the old architecture was Compliant. **(b) U-Prove** • The PPMCC between *t* and the memory usage is -0.73; • The PPMCC between *k* and the memory usage is 0.34; The results indicate that UProve tends to use less memory when tested for a longer period. The results also show that the memory usage has a positive correlation with the number of concurrent requests. Furthermore, the maximum memory usage peak is 237MB.
Mitigation/Corrective action	None

[6] The real implementation used the java.lang.Runtime API . See the following URL for more details: http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html

4.2 Component and Interface-level perturbations

In this section are presented the misuse case scenarios related with the ABCE API calls used during the Presentation stage, namely `createPresentationToken()` and `verifyTokenAgainstPolicy()`. Other calls e.g., `canBeSatisfied()`, `getToken()` and `deleteToken()` do not allow for a relevant perturbation that comprises the ABCE subsystem's robustness. For the presented misuse cases, the same considerations apply that for those shown in Section 2.

Scenario 4.2.1: Perturbing the user ABCE `createPresentationToken()` call	
Summary	This perturbation aims to test the robustness of the user ABCE API call that returns a presentation token satisfying the Verifier's presentation policy.
Component Under Evaluation (ET)	ABCE API Call - `createPresentationToken()` – User
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `PresentationToken createPresentationToken(PresentationPolicyAlternatives p, IdentitySelection idSelectionCallback)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms, by implementing as individual tests the following perturbations: a) Create a presentation policy with a `/abc:PresentationPolicyAlternatives/@Version` different than "1.0". b) Specify a credential attribute in `…/abc:Credentials/abc:Credential/abc:DisclosedAttribute/@AttributeType` that does not occur in at least one of the listed credential specifications (i.e., the multiple `abc:CredentialSpecUID` elements listed in the `abc:CredentialSpecAlternatives` child element of the ancestor `abc:Credential` element). c) Specify in …/abc:PresentationPolicy/abc:VerifierDrivenRevocation/abc:Attribute/@CredentialAlias a value that does not occur as an Alias attribute in an `abc:Credential` element within this `abc:PresentationPolicy`. This feature in implemented only in Idemix.
Perturbation Class	C-O

Output old arch.	**RESULT**: (a-b) Compliant, (c) Compliant **DETAILS**: **(a) `Version = 2.0`** The method "createPresentation" in class "eu.abc4trust.abce.pertubationtests.section4.Test21" encountered an error of "Failed to create presentation token: java.lang.UnsupportedOperationException: Unknown version, expected '1.0', got '2.0': Unknown version, expected '1.0', got '2.0"' **(b) Disclose Non Existent Attribute `AttributeType`** The method "createPresentation" in class "eu.abc4trust.abce.pertubationtests.section4.Test21" encountered an error of "Failed to create presentation token : java.lang.RuntimeException: java.lang.RuntimeException: java.lang.NullPointerException: java.lang.RuntimeException: java.lang.NullPointerException". **(c) Unknown credential alias** The method `createPresentationToken` in class "eu.abc4trust.abce.pertubationtests.section4.Test21" encountered an error of "Failed to verify presentation token: eu.abc4trust.exceptions.TokenVerificationException: The presented token does not satisfy the policy: The presented token does not satisfy the policy".
Output new arch.	N/A
Mitigation/Corrective action	The test is Compliant from the point of view of our analysis. However, it must be noted that the exception returned is wrong. The exception is raised during the generation of the token and not during the verification. According to that, a possible mitigation is to raise a proper exception or, alternatively, to formally deprecate the method as it is de-facto deprecated.

Scenario 4.2.2: Perturbing the `PresentationPolicyAlternatives` parameter of the `verifyTokenAgainstPolicy()` call	
Summary	This perturbation to the `PresentationPolicyAlternatives` parameter aims to test the robustness of the API call that given a single presentation policy and a single presentation token, checks whatever the latter satisfies the former and the validity of the cryptographic evidence contained in the token.
Component Under Evaluation (ET)	ABCE API Call - `verifyTokenAgainstPolicy()` - Verifier
ET Type	Comp

Normal flow	As documented in H2.1 the specification of this API call is: `PresentationTokenDescription verifyTokenAgainstPolicy( PresentationPolicyAlternatives p, PresentationToken t, boolean store)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms, by implementing as individual tests the following perturbations to the `PresentationPolicyAlternatives` parameter: a) Create a presentation policy with a `/abc:PresentationPolicyAlternatives/@Version` different than "1.0". b) Specify in …/abc:PresentationPolicy/abc:VerifierDrivenRevocation/abc:Attribute/@CredentialAlias a values that does not occur as an Alias attribute in an `abc:Credential` element within this `abc:PresentationPolicy`. This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULTS**: (a) Non-compliant, (b) Compliant; **DETAILS**: **(a) Version = 2.0** The test case (a) failed upon providing a presentation policy version equal to 2.0, the component does not raise an exception. The implemented test produced the following sequence of events: 1. Managed to issue a credential 2. Issued credential 3. Successfully created a presentation token 4. Successfully verified presentation token **(b) Unknown credential alias** The method "verifyToken" in class "Failed to verify presentation token : eu.abc4trust.exceptions.TokenVerificationException: The presented token does not satisfy the policy: The presented token does not satisfy the policy".
Output new arch.	**RESULTS**: (a) Non-compliant, (b) N/A **DETAILS:** **(a) Version = 2.0** The new crypto architecture has the same output **Test cases (b)** This test case was not executed against the new architecture.
Mitigation/Corrective action	The version is never checked. It should be either removed from the specification or implemented a check.

Scenario 4.2.3: Perturbing the `PresentationToken` parameter of the `verifyTokenAgainstPolicy()` call	
Summary	This perturbation to the `PresentationToken` parameter aims to test the robustness of the API call that given a presentation policy and presentation token, checks whatever the latter satisfies the former and the validity of the cryptographic evidence contained in the token.
Component Under Evaluation (ET)	ABCE API Call - `verifyTokenAgainstPolicy()` - Verifier
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: `PresentationTokenDescription verifyTokenAgainstPolicy( PresentationPolicyAlternatives p, PresentationToken t, boolean store)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms, by implementing as individual tests the following perturbations to the `PresentationToken` parameter: a) Create a presentation token with a `/abc:PresentationToken/@Version` different than "1.0". This feature in implemented only in Idemix.
Perturbation Class	C-O
Output old arch.	**RESULT**: (a) Non-compliant **DETAILS**: **(a) Version = 2.0** The test (a) failed upon providing a presentation policy version 2.0, the component does not raise an exception. The implemented test produced the following sequence of events: 1. Managed to issue a credential 2. Issued credential 3. Successfully created a presentation token 4. Successfully verified presentation token
Output new arch.	**RESULT**: (a) Non-Compliant **DETAILS**: **(a) Version = 2.0** The new crypto architecture has the same output (for (a)).
Mitigation/Corrective action	The version is never checked. It should be either removed from the specification or implemented a check.

5 Revocation

Within the ABCE layer have been implemented two basic revocation-related functionalities: querying revocation information from the Revocation Authorities (RA) and, requesting a credential revocation (driven either by Issuers or Verifiers). Interested readers are referred to [2] for further details related with the revocation process. For the purposes of this document, we will only focus on *(i)* the currently implemented issuer-driven revocation and, *(ii)* the ABC-architecture components that interact during the revocation stage. Please notice that the RA is invoked only in the following cases:

1. The Issuer queries the RA during the issuance for the revocation handle attribute.
2. While retrieving the revocation information during the Presentation.
3. While revoking credentials via the RA on request from the Issuer, since only issuer-driven revocation is supported currently.

5.1 Data flow-level perturbations

In this section are presented two misuse cases targeting the data flows involved during the operation of the Revocation Authority (RA). The first scenario is a stress test analogous to those presented in sections 2.1, 3.1 and 4.1

The second scenario considers three special cases based on the unreachability of the RA, and its unavailability to service different classes of requests from the clients (i.e., issuing a revocation handle, retrieving revocation information and, revoking credentials).

Scenario 5.1.1: Stress perturbations on the Revocation Authority's ABCE component	
Summary	A stress perturbation is tested on the RA's ABCE, to assess its resilience against a denial of service (DoS).
Component Under Evaluation (ET)	ABCE – Querying revocation information or Requesting credential revocation. The API calls are: `generateNonRevocationEvidence()`, `revoke()` and `updateRevocationInformation()` – Revocation Authority
ET Type	Arch
Normal flow	The ABCE component has been designed to support both concurrent revocation information updates (requested by Users and Verifiers) and concurrent credential revocation requests (Issuers/Verifiers).
Perturbation	Stress the ABCE component by keeping *k* concurrent revocation requests during a period of *t* seconds. The goal is to monitor the resource consumption (i.e., heap memory consumption) and the availability of the service (i.e., time to process all the requests). This scenario is applicable only to Idemix. The parameters *k* and *t* are the following: • k ∈ {500, 1000, 1500}; • t ∈ {60, 120, 180}.

	The pseudo-algorithm is the following:
	`1 – Until time t is reached:`
	`2 – init = freeMemory();`
	`3 – Execute k concurrent requests;`
	`4 – mem = freeMemory() – init;`
	`5 – Log mem, t, and k`
	Where `freeMemory()` is the API call to obtain the amount of system memory available[7].
	Document the outputs and assess the consumption of the resource for (a) Idemix and (b) U-Prove.

Perturbation Class	DF-S (subclass of DF-O)
Output old arch.	**RESULTS**: (a) Compliant, (b) Inconclusive **DETAILS**: We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of resources. **(a) Idemix** • The PPMCC between *t* and the memory usage is -0.72; • The PPMCC between *k* and the memory usage is 0.32; The result shows that the memory usage increases with the number of concurrent requests, while over the time the memory used tends to be released. This can be caused by the garbage collector which releases unused objected. **(b) U-Prove** The test execution reached the timeout set for these tests. This happens because the feature under test is supported only by Idemix.
Output new arch.	N/A
Mitigation/Corrective action	None

Scenario 5.1.2: RA unreachable	
Summary	A perturbation to test Issuer's resilience against unavailability of the RA
Component Under Evaluation (ET)	ABCE – Issuer's data flows that involve the RA - Issuer, Revocation Authority
ET Type	Arch
Normal flow	The RA is invoked by the Issuer (using its Revocation Proxy) in the following cases:

[7] The real implementation used the java.lang.Runtime API . See the following URL for more details: http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html

	a) While retrieving the revocation handle attribute during issuance. b) While retrieving the revocation information during the Presentation protocol. c) While revoking users via the RA on request from the Issuer, since only issuer-driven revocation is supported currently.
Perturbation	Stop the operation of the RA and document the output when: a) The Issuer queries the RA during the issuance for the revocation handle attribute. b) Retrieving the user's revocation information during the Presentation protocol. c) Revoking credentials via the RA on request from the Issuer.
Perturbation Class	DF-O
Output old arch.	**RESULTS**: (a-c) Compliant **DETAILS**: The method "presentIDCard" in class "eu.abc4trust.abce.pertubationtests.section5.Test12" encountered the following exceptions: a) Failed to issue credential : com.sun.jersey.api.client.ClientHandlerException: java.net.SocketTimeoutException: Read timed out b) "Failed to create presentation token : java.lang.RuntimeException: eu.abc4trust.cryptoEngine.CryptoEngineException: eu.abc4trust.keyManager.KeyManagerException: eu.abc4trust.keyManager.KeyManagerException: com.sun.jersey.api.client.ClientHandlerException: java.net.SocketTimeoutException: Read timed out: eu.abc4trust.cryptoEngine.CryptoEngineException: eu.abc4trust.keyManager.KeyManagerException: eu.abc4trust.keyManager.KeyManagerException: com.sun.jersey.api.client.ClientHandlerException: java.net.SocketTimeoutException: Read timed out" c) "Failed to issue credential : com.sun.jersey.api.client.ClientHandlerException: java.net.SocketTimeoutException: Read timed out".
Output new arch.	N/A
Mitigation/Corrective action	None

5.2 Component and Interface-level perturbations

The ABCE API calls related with the revocation stage of the privacy-ABC life-cycle (e.g., revoke(), updateNonRevocationEvidence(), getCurrentRevocationInformatio()) do not allow for a relevant perturbation that comprises the ABCE subsystem's robustness. Take for example the following two cases:

1. A successful perturbation applied to calls like getRevocationHistory() might affect the behavior of the application running on top of the ABCE API (e.g, obtain a "false" value where a "true" value was expected), although even in that case it would be still compliant with the API specification (i.e., a valid value is being obtained from the call).

2. The revoke() call is similar to the previous example, possible perturbations (wrong URI/oversized parameters) cannot compromise the robustness of the RA reference implementation.

6 Inspection

Credential's inspection is usually needed in order to lift the full anonymity granted by privacy-ABCs, for example in the case of misbehaving users [2].

From the ABC API perspective the relevant call to test is `inspect()`, given its importance to the overall system's security.

6.1 Data flow-level perturbations

This section presents a stress test case for the Inspector, similar to those shown in sections 2.1, 3.1, 4.1 and 5.1. This perturbation aims to compromise the availability of the Inspector by depleting its available resources through a set of concurrent requests (up to 1500 in a maximum of 180s), and testing both Idemix and U-Prove in order to compare their behavior.

Scenario 6.1.1: Stress perturbations on the Inspector's ABCE component	
Summary	A stress perturbation is tested on the Inspector's ABCE, to assess its resilience against a denial of service (DoS).
Component Under Evaluation (ET)	ABCE – Requesting the inspection of a presentation token - Inspector
ET Type	Arch
Normal flow	The ABCE component has been designed to support concurrent inspection requests.

Perturbation	Stress the ABCE component by keeping *k* concurrent inspection requests during a period of *t* seconds. The goal is to monitor the resource consumption (i.e., heap memory consumption) and the availability of the service (i.e., time to process all the requests). *When applicable, this experiment should be repeated for all involved CEs.*
	The parameters *k* and *t* are the following:
	• k ∈ {500, 1000, 1500};
	• t ∈ {60, 120, 180}.
	The pseudo-algorithm is the following:
	`1 - Until time t is reached:`
	`2 - init = freeMemory();`
	`3 - Execute k concurrent requests;`
	`4 - mem = freeMemory() - init;`
	`5 - Log mem, t, and k`
	Where `freeMemory()` is the API call to obtain the amount of system memory available[8].
	Document the outputs and assess the consumption of the resource for (a) Idemix and (b) U-Prove.

[8] The real implementation used the java.lang.Runtime API . See the following URL for more details: http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html

Perturbation Class	DF-S (subclass of DF-O)
Output old arch.	**RESULT**: (a) Compliant, (b) Inconclusive; **DETAILS**: We applied the PPMCC (Pearson Product-Moment Correlation Coefficient) to identify correlation between the parameters *k* and *t,* and the consumption of resources. **(a) Idemix** • The PPMCC between *t* and the memory usage is -0.28; • The PPMCC between *k* and the memory usage is -0.44; The result shows that the memory usage decreases with both *k* and *t*. **(b) U-Prove** The test execution reached the timeout set for these tests. This feature in implemented only in Idemix.
Output new arch.	N/A
Mitigation/Corrective action	None

6.2 Component and Interface-level perturbations

In this section is presented the misuse case scenario related with the ABCE API call used during the inspection of a presentation token, namely `inspect()`. The presented misuse case mainly consists of a series of tests targeting the robustness of this ABCE API call against buffer overflows, by performing a series of perturbations that make use of unbounded parameters (i.e., non-compliant with the API specification). Apart from testing its resilience against buffer overflows, the presented misuse case also tests invalid parameter values in order to find out the operational behavior of the reference implementation.

Scenario 6.2.1: Perturbing the `PresentationToken` parameter of the `inspect()` call	
Summary	This perturbation to the `PresentationToken` parameter aims to test the robustness of the API call that is in charge of inspecting a presentation token.
Component Under Evaluation (ET)	ABCE API Call - `inspect()` - Inspector
ET Type	Comp
Normal flow	As documented in H2.1 the specification of this API call is: • `Attribute[] inspect(PresentationToken t)`
Perturbation	Test and document the correctness of the implemented exception catching mechanisms, by implementing as individual tests the following perturbations to the `PresentationToken` parameter: a) Create a presentation token with a `/abc:PresentationToken/@Version` different than "1.0".
Perturbation Class	C-O

Output old arch.	**RESULT**: (a) Non-compliant **DETAILS**: **(a) `Version = 2.0`** The test case (a) fails because the inspection function does not raise an exception while inspecting a perturbed token.
Output new arch.	**RESULT**: (a) Non-Compliant **DETAILS**: **(a) `Version = 2.0`** The new crypto architecture has the same output for (a).
Mitigation/Corrective action	The version is never checked. It should be either removed from the specification or implemented a check.

7 Summary

This deliverable detailed the PA conducted on the ABCE component (and other core-components that are invoked through the ABCE API calls) of the reference implementation documented in Deliverable D2.1 and Heartbeat H2.2 (i.e., "old crypto architecture") in order to assess its robustness. The goal is to identify those elements that need to be further analyzed and improved (from a robustness perspective), before integrating into the next version of the implementation (i.e., the "new crypto architecture" in Deliverable D4.2, and Heartbeats H2.2 and H4.1). The PA started with the analysis of the whole system documented in D2.1/H2.2 in order to identify the ET that can compromise the overall robustness of the system. Then, the PA classified the ET into architecture flow, implementation component/interface, and usage[9]. The classification allows selecting the type of perturbations to apply. Third, tests are executed against the implementation. The results of a test can be one of the following: *Compliant*, *Non-compliant,* and *Inconclusive*. A test is *Compliant* if its execution detected a fail-safe behavior, or, alternatively, if the observed behavior does not show any evidence of uncontrolled resource consumption. If a test does not detect a fail-safe, then the test *is Non-compliant*. A test is *Inconclusive*, e.g., it cannot be applied to a component, or its execution time exceeds a prefixed timeout. Finally, the PA identified and suggested the proper action to be taken in order to mitigate the findings in the next version of the reference implementation (i.e., Deliverable D4.2).

In total, the PA consisted of 25 perturbation scenarios containing in total 43 test cases. Tests were designed starting from valid functional test cases, and then introducing perturbation inputs. The selection of inputs is done using both a uniform distribution function and manual selection over a set of outlier inputs. Invalid inputs are identified by combining the syntax and semantics of API function parameters. Table 3 reports the total number of perturbation scenarios (column *Scen.*) and test cases (column *TC*) grouped by ET Type. Test cases are distributed in: *(i)* 16 flow and stress test cases, *(ii)* 27 component and interface test cases.

As mentioned before, the scope of this deliverable is to assess the robustness of the reference implementation of ABC4Trust. The PA does not apply security testing techniques, such as penetration testing. Moreover, this deliverable does not perform any benchmark and does not define metrics for it. Software benchmarks and metrics are addressed in WP2 and WP3.

Table 3: Scenarios and test cases (TC) grouped by ET Type

Class	***ET***									
	Setup (2.x.y)		***Issuance (3.x.y)***		***Presentation (4.x.y)***		***Revocation (5.x.y)***		***Inspection (6.x.y)***	
	Scen.	*TC*	*Scen.*	*TC*	*Scen.*	*TC*	*Scen.*	*TC*	*Scen.*	*TC*
Data flow (x.1.y)	1	2	4	5	1	2	2	5	1	2
Component (x.2.y)	9	13	3	9	3	4	0	-	1	1
Total	10	15	7	14	4	6	2	5	2	3

[9] Perturbations related with the misuse (functional bugs and failed installations) of the reference implementation were documented by the WP6 and WP7 pilots. The reported results are shown in Appendix A and Appendix B.

7.1 Detailed overview of the results

Table 1 shows the results of the test execution. The result of a test can be *Compliant* (column *S*), *Non-compliant* (column *T*), and *Inconclusive* (column *I*). The number of Compliant tests is 31 out of 43, while the number of Non-compliant tests is 8. The remaining 4 tests are inconclusive. Inconclusive tests can be classified in unresponsive tests (i.e., reached the timeout condition), and the component under test is not implemented (i.e., it is specified, but the implementation is missing. Table 5 and Table 6 report the results for each test case. We summarize the results of the tests below.

Table 4: Summary of results grouped by ET Type

Class	***Setup (2.x.y)***				***Issuance (3.x.y)***				***Presentation (4.x.y)***				***Revocation (5.x.y)***				***Inspection (6.x.y)***			
	TC	*C*	*N*	*I*	*TC*	*C*	*N*	*I*	*TC*	*C*	*N*	*I*	*TC*	*C*	*N*	*I*	*TC*	*C*	*N*	*I*
Data flow (x.1.y)	2	1	1	0	5	4	0	1	2	1	0	1	5	4	0	1	2	1	0	1
Component (x.2.y)	13	10	3	0	9	8	1	0	4	2	2	0	0[10]	-	-	-	1	0	1	0
Total	15	11	4	0	14	12	1	1	6	3	2	1	5	4	0	1	3	1	1	1

Legend: C=Compliant, N=Non-compliant, I=Inconclusive

Table 5: Detailed view of data flow perturbations

Test (Scenario ID + case)	***Old crypto***	***New crypto***
2.1.1 (a)	Compliant	
2.1.1 (b)	Non-compliant	Non-compliant
3.1.1 (a)	Compliant	
3.1.1 (b)	Inconclusive (Timeout)	
3.1.2	Compliant	
3.1.3	Compliant	
3.1.4	Compliant	
4.1.1 (a)	Compliant	
4.1.1 (b)	Inconclusive (Timeout)	
5.1.1 (a)	Compliant	
5.1.1 (b)	Inconclusive (Timeout)	
5.1.2 (a)	Compliant	
5.1.2 (b)	Compliant	
5.1.2 (c)	Compliant	
6.1.1 (a)	Compliant	
6.1.1 (b)	Inconclusive (Timeout)	

[10] Please refer to Section 5.2 for an explanation related to the lack of perturbations on this stage.

Table 6: Detailed view of component and interface perturbations

Test (Scenario ID + case)	Old crypto	New crypto
2.2.1 (a)	Compliant	
2.2.1 (b)	Compliant	
2.2.2 (a)	Compliant	
2.2.2 (b)	Compliant	
2.2.2 (c)	Non-compliant	Compliant
2.2.4	Compliant	
2.2.5	Compliant	
2.2.6	Compliant	
2.2.7 (a)	Non-compliant	Compliant
2.2.7 (b)	Compliant	
2.2.7 (c)	Compliant	
2.2.8	Non-compliant	Inconclusive (not applicable)
2.2.9	Compliant	
3.2.1 (a)	Compliant	
3.2.1 (b)	Compliant	
3.2.1 (c)	Compliant	
3.2.1 (d)	Compliant	
3.2.1 (e)	Compliant	
3.2.2 (a)	Compliant	
3.2.2 (b)	Compliant	
3.2.2 (c)	Compliant	
3.2.3	Non-compliant	Non-compliant
4.2.1 (a)	Compliant	
4.2.1 (b)	Compliant	
4.2.1 (c)	Compliant	
4.2.2 (a)	Non-compliant	Non-compliant
4.2.2 (b)	Compliant	
4.2.3 (a)	Non-compliant	Non-compliant
6.2.1 (a)	Non-compliant	Non-compliant

7.1.1 Compliant

31 tests out of 43 were compliant. It is important to consider that (a) the PA does not claim completeness as an experimental methodology and (b) the test platform restrictions often do not allow

tracing the stress cases and loads reaching the ABCE and its CE. This is a natural limitation of any PA approach where one needs to constrain elements such as length of inter-component propagation flows or the level of detail of a perturbation case. Thus the validity of the PA and designated success is based around the class of considered perturbations either as outliers, high-likelihood or nature of interface/data-flows for a target. Naturally, it must also be noted that these results do not imply that the implementation is secure. As shown in Figure 1, the PA aims at the robustness of the implementation. Although robustness issues may imply security issues as well, the PA did not explicitly target security properties of the implementation. Indeed, the PA did not apply security testing techniques, such as penetration testing, and they were considered out of scope as reported earlier in H4.1.

7.1.2 Non-compliant

The total number of non-compliant tests in the **old architecture** is 8. The tests that failed are:

1. Test case 2.1.1 (b) shows that U-Prove is at risk of memory exhaustion
2. Test case 2.2.2 (c) fails because random maximum lengths for the Attribute Description are accepted
3. Test case 2.2.7 (a) fails because the component accepts a null crypto mechanism
4. Test case 2.2.8 fails because the component accepts randomly chosen negative security levels and crypto mechanisms
5. Test cases 3.2.3 fail when testing the components with an invalid token version number. A different version number was tested with the test cases 4.2.2 (a), 4.2.3 (a), and 6.2.1 (a). All these tests failed as well

These tests indicate the presence of issues that may affect the robustness of the application. They were executed also against the **new architecture**. The results are the following:

1. Test case 2.1.1 (b) still shows a memory exhaustion risk, however it must be noted that the component under test should not be accessible by external attacker. This should mitigate the risk of denial of service attacks
2. Test case 2.2.2 (c) and Test case 2.2.7 (a) now succeeds because the component raises an exception
3. Test case 2.2.8 is inconclusive because the scenario is no longer relevant for the new architecture
4. Test cases 3.2.3, 4.2.2 (a), 4.2.3 (a) and 6.2.1 (a) still fail. The corrective action to be taken is to either remove the behavior from the specification or to implement a version check.

The PA spotted eight issues in the old architecture: 2 of them were solved in the new architecture, 1 is considered unreachable by an external entity (e.g., an attacker), 1 is no longer applicable, and the remaining 4 are still marked as fail however the corrective actions have been identified.

7.1.3 Inconclusive

The total number of inconclusive results is 4 that are due to a test execution timeout when testing the U-Prove cryptographic engine (see Scenarios 3.1.1 (b), 4.1.1 (b), 5.1.1 (b), and 6.1.1 (b)).

8 Bibliography

[1] Voas J.M and Miller K.W., "Software Testability: The New Verification," Proc. of IEEE Software, vol.12, no.3, pp: 17-28, May 1995.

[2] Krontiris I. (ed.), “Architecture for Attribute-based Credential Technologies - Version 1,” ABC4Trust Deliverable D2.1, Tech. Rep., 2011.

[3] Voas J. M. and McGraw G., “Software Fault Injection: Inoculating Programs Against Errors”. John Wiley & Sons, Inc., 1997

[4] Nik L., Munro M. and Xu J., "Simulating errors in web services." International Journal of Simulation Systems, Science & Technology. pp: 29-37, 2004

[5] Nik L., Munro M. and Xu J. "Assessing the Dependability of SOAP RPC-Based Web Services by Fault Injection," Proc. of Intl. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), 2003

[6] Koopman P., et. al., "Comparing operating systems using robustness benchmarks," Proc. of the Symposium on Reliable Distributed Systems (SRDS), pp: 72-79, 1997.

[7] Krontiris I. (ed.), “ABC4Trust Architecture for Developers” ABC4Trust Heartbeat H2.1, Tech. Rep., 2012.

Appendix A: Pilot misuse cases as reported by WP6

Note: These tests were not part of PA – are being provided as supplemental information only.

The following misuse cases have been reported by the Soderhamn pilot (WP6). Only during the first round of the pilot, operators misuse cases were detected related with checking for functional bugs and failed installations. Further details about used test bed (hardware and software) can be found in:

D6.1 Application Description for the school deployment

By Souheil Bcheri, Norbert Goetze, Monika Orski, Harald Zwingelberg

Scenario 1: UProve cards don't store credentials	
Summary	UProve Smart Cart doesn't save credentials due to new issuer parameters
Evaluation Target (ET)	User Client or User Client to Issuer interface Test PC: Asus A55A (Core i7 Quad, 8Gb DDR3, 256Gb SSD, 1Gb VRAM), Windows 7 x64
ET Type	Usage
Normal flow	1. Initialize U-Prove Smart Card 2. Add the user record to IdM 3. Register in IdM or add pseudonym to IdM 4. Login to IdM Portal 5. Click button to save credential to the card (credSchool) 6. Enter PIN and confirm in dialog 7. Use "View credentials" to check the credential
Perturbation	Credential was not saved to the card
Perturbation Class	Misuse (M-U)
Base functional test case (Task 4.6)	1. Trying steps 1 to 6 from normal flow 2. Checking the credential according to step 7 3. Discovering that credential was not saved but IdM has an issuance record 4. Rebooting, reinstalling, starting over from 1, no change of result
Output	IdM LDAP Tool shows the data which corresponds to artifacts that credential was issued by ABCE Issuer, but Smart Card is empty
Mitigation/ Corrective action	Check the User Client against correctness of issuance parameters and ability to work with different length of keys for UProve or rollback to previous versions

Appendix B: Pilot misuse cases as reported by WP7

Note: These tests were not part of PA – are being provided as supplemental information only.

The following misuse cases were found and reported during both rounds of the Patras' pilot (WP7). Further details about used test bed (hardware and software) can be found in:

D7.1 Application Description for students

By Joerg Abendroth, Vasiliki Liagkou, Apostolis Pyrgelis, Christoforos Raptopoulos, Ahmad Sabouri, Eva Schlehahn, Yannis Stamatiou,Harald Zwingelberg

D7.2 Necessay hardware and software package for the student pilot deployment

By Kasper Damgaard, Hamza Ghani, Norbert Goetze, Anja Lehmann, Vasiliki Liagkou, Jesus Luna, Gert Læssøe Mikkelsen, Apostolos Pyrgelis, Yannis Stamatiou

Misuse cases reported during the first round:

Scenario 1: DoS on Smart Card due to "Out of RAM Error"	
Summary	When the user collected his *credUniv* and *credCourse* credentials more than twice from the University Registration System, his smart card could not be accessed anymore due to an "Out of RAM" error (640E status word).
Component Under Evaluation (ET)	ABC4Trust Smart Card (Basic Card ZC7.5)
ET Type	Usage/Deployment (Usage).
Normal flow	The user can get issued *credUniv* and *credCourse* as many times as the smart card's available RAM memory allows.
Perturbation	The users tried to collect both *credUniv* and *credCourse* more than twice from the University Registration System.
Perturbation Class	U-M
Preconditions	The user requests to be issued both *credUniv* and *credCourse* multiple times (e.g. more than twice).
Output	Due to the "Out of RAM" error the student's smart card was unusable. Thus, he could not perform basic operations of the pilot e.g. log in to the Course Evaluation System and submit his evaluation, or log in to the University Registration System with an ABC token.
Mitigation/Corrective action	The users who faced this problem had to visit the pilot administrators with their smart card so that they would re-initialize it. The new smart card's pseudonym had to be registered at the IDM database and, the student database attributes (e.g. crypto engine) had to be re-initialized. After that, the students had to collect once again their credentials from the University Registration System (but only one time), so they were able to perform the course evaluation.

Scenario 2: DoS on Issuer/Verifier due to concurrent access to the ABCE layer services.	
Summary	When users access any of the Issuer or Verifier ABCE simultaneously (e.g. when 2 users try to log in at the Course Evaluation System at the same time) the issuance/verification service might catch an exception and reply with an error response code.
Component Under Evaluation (ET)	Issuer/Verifier ABCE layer.
ET Type	Comp
Normal flow	When two valid users contact the Issuer/Verifier ABCE layer service at exactly the same time, they should be able to get a valid response from the web service.
Perturbation	When two users with valid smart cards access an ABCE layer service (e.g. verification service when they try to log in to the Course Evaluation System) one of them might get rejected because of an ABCE exception.
Perturbation Class	DF-O
Preconditions	Multiple users access an Issuer/Verifier ABCE layer web service (e.g. verification web service) at exactly the same time.
Output	Any of the valid users trying to access the Issuer/Verifier ABCE web service concurrently with another user, could get an error response from the web service. This way he would not be able to access a resource (e.g. the course evaluation webform) at that exact point in time.
Mitigation/Corrective action	The rejected user should retry accessing the issuance/verification service once again. If there are not any other users accessing it, then the request should be successful.

Misuse cases reported during the second round of WP7:

Scenario 1: Smart Card Reader Drivers	
Summary	On some user PC's, the smart card reader (Omnikey 3021 USB) drivers were not installed properly by the Windows Driver Installation Manager. As a result, the User Application could not communicate with the smart card and the User interaction with the pilot systems was problematic.
Evaluation Target (ET)	User PC, User Application
ET Type	Usage/Deployment (Usage).
Normal flow	Normally, the Windows OS would recognize successfully the smart card reader and would install the appropriate drivers.
Perturbation	When a User interacted with the pilot systems (e.g. registering her smart card at the University Registration System), the User Application tried to transmit some commands to the smart card. These commands could not reach the smart card and as a result the student could not complete the requested operation.
Perturbation Class	⇒ User-level: any of Misuse (U-M).

Base functional test case (Task 4.6)	Testing the User Application – Installing the Smart Card Reader Drivers on the User pc.
Output	The User could not perform any Privacy-ABC operation (e.g. register her smart card, obtain credentials etc.) since the User Application could not communicate properly with the smart card.
Mitigation/Corrective action	The Users had to download and install manually the smart card reader drivers from the Omnikey website. Then, the User Application would communicate successfully with the smart card and the users could interact with the pilot systems without any problems.

Scenario 2: Unlocking the Smart Card Using the PUK	
Summary	When a smart card would get locked (by inserting the wrong PIN, 3 times in a row), the User could unlock it through the User Application by entering the PUK. If the PUK value was shorter than 8 digits (such a case is possible when using the smart card initialization script), the User Application could not unlock the smart card since it was programmed to handle only PUK values that were exactly 8 digits long. As a result, the User could not unlock her smart card and could no longer interact with the pilot systems.
Evaluation Target (ET)	User Application
ET Type	⇒ Usage/Deployment (Usage).
Normal flow	When the User would utilize the User Application in order to unlock her smart card, she would be requested to enter her PUK value. As soon as she did so, the User Application would request from the User to enter the new PIN value and the smart card would get unlocked.
Perturbation	Since the User could not unlock her smart card, it was impossible for her to interact with the pilot systems or even obtain attendance data during the course lecture.
Perturbation Class	⇒ User-level: any of Misuse (U-M).
Base functional test case (Task 4.6)	Testing the User Application.
Output	With the smart card in locked mode, the User could not interact with any of the pilot systems until she was able to unlock it.
Mitigation/Corrective action	In order to solve this issue, the pilot administrators implemented a script in Java that could handle PUK values shorter than 8 digits and assist the Users in unlocking their smart cards. When executed, the script requested from the User to enter the PUK value and then it would require from the User to enter the new PIN value. Finally, the smart card would be unlocked and the User could continue interacting with the pilot systems.

Scenario 3: Inconsistent State of University Credential after Trying to Perform a Proof Towards the Course Evaluation System with Insufficient Counter Value	
Summary	When a User who possessed a smart card that had the University and Course credentials stored on it - but not a sufficient attendance counter value, tried to log-in at the Course Evaluation System (the presentation policy asked for both credUniv and credCourse), the state of the University Credential changed from 'presentable' to 'presentation committed' but did not change back when the proof failed. As a result, the University credential remained in an inconsistent state and it could not be used in future proofs.

Evaluation Target (ET)	User Smart Card
ET Type	Usage
Normal flow	When a student with a smart card which contained both University and Course credentials, but not a sufficient attendance value, tried to log-in to the Course Evaluation System, the proof should fail (due to the insufficient value) and the student should not be able to log-in for submitting her evaluation for the course. However, when the counter would reach the pre-defined attendance threshold, the User should be able to log-in to the Course Evaluation System by creating a presentation token based on the existing University and Course credentials.
Perturbation	The student's University credential was in an inconsistent state and it could not be used in future proofs. As a result, the User could not use this credential for logging in the Course Evaluation System and submit her evaluation for the course.
Perturbation Class	⇒ User-level: any of Misuse (U-M)
Base functional test case (Task 4.6)	Logging in to the Course Evaluation System (by presenting both credUniv and credCourse).
Output	With the University credential in an inconsistent state the User could not create a presentation token based on it and an exception would show up at the User ABCE layer. As a result, the User could not log-in to the Course Evaluation System (which required both credUniv and credCourse) and submit her evaluation for the pilot course.
Mitigation/Corrective action	In order to deal with this issue, the User had to obtain a new University credential from the University Registration System and delete the old one from her smart card. The state of the newly obtained credential would be consistent. Thus, when the counter on her smart card had reached the threshold she could perform a proof towards the Course Evaluation System and submit her evaluation for the course.

Scenario 4: Unable to Obtain a Tombola Credential due to Insufficient Smart Card Storage Space	
Summary	When some Users obtained their University and Course credentials from the University Registration System multiple times, their smart card memory space was insufficient to store the Tombola credential. As a result, when a User evaluated the course and tried to obtain the Tombola credential from the Course Evaluation System the issuance protocol would fail, since there was not enough space on the smart card to store the credential.
Evaluation Target (ET)	User Application / User Smart Card
ET Type	⇒ Usage/Deployment (Usage).
Normal flow	When the User would request the Tombola credential from the Course Evaluation System, the issuance protocol would complete without any problems and the credential would be stored on the User smart card.
Perturbation	The Tombola credential cannot be obtained due to insufficient space on the smart card.
Perturbation Class	Classify the perturbation to test in any of the following: ⇒ User-level: Misuse (U-M)
Base functional test case (Task 4.6)	Obtaining the Tombola Credential from the Course Evaluation System.

Output	Since the student could not obtain the Tombola credential from the Course Evaluation System, she could not register for the lottery through the Tombola System.
Mitigation/Corrective action	In order to resolve this issue, the Users had to make some space on their smart card by deleting the redundant credentials from it (the ones they had obtained multiple times). This operation was possible by the User Application which allowed the User to handle the contents of her smart card.

Appendix C: CSV formatted results of stress perturbations

As the following raw data is only available on the restricted Wiki, we provide it here in its basic form for completeness of information access to the reviewer.

Scenario 2.1.1: Stress perturbations on the ABCE component

Old Architecture

The column "CE" stands for the Crypto Engine. Columns "k" and "t" are, respectively, the number of concurrent requests and the duration of the test. The column "reqs" is the total number of requests performed during the test. The column "Errs" is the number of exception reported. The column "real t" is the real execution time, and, finally, the column "mem" reports the memory (MB) consumed during the test execution.

SetupSystemParameters():

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	500	60	40843	0	61	279
Idemix	1000	60	44888	170	60	101
Idemix	1500	60	46500	403	60	106
Idemix	500	120	85269	494	120	193
Idemix	1000	120	99171	150	120	67
Idemix	1500	120	98269	552	120	245
Idemix	500	180	144611	705	180	211
Idemix	1000	180	141579	100	180	212
Idemix	1500	180	141821	365	180	122
Idemix	10	60	46529	659	60	30
U-Prove	500	60	191111	0	60	19
U-Prove	1000	60	191882	0	60	163
U-Prove	1500	60	191152	0	60	101
U-Prove	500	120	387594	0	120	52
U-Prove	1000	120	387967	0	120	51
U-Prove	1500	120	372831	0	120	135
U-Prove	500	180	581024	0	180	54
U-Prove	1000	180	581433	0	180	113
U-Prove	1500	180	583374	0	180	166

[11] „mem" is the difference between the amount of memory used before and after the execution of the test. Due to the interference of the Java Garbage Collector, it may happen that "mem" is negative.

SetupIssuerParameters():

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	500	60	525	0	60	46
Idemix	1000	60	1018	0	60	35
Idemix	1500	60	1517	0	60	38
Idemix	500	120	524	0	120	216
Idemix	1000	120	1015	0	121	42
Idemix	1500	120	1514	0	120	175
Idemix	500	180	511	0	181	270
Idemix	1000	180	1011	0	180	-33
Idemix	1500	180	1511	0	180	60
U-Prove	500	60	510	0	60	156
U-Prove	1000	60	1008	0	60	98
U-Prove	1500	60	1506	0	60	80
U-Prove	500	120	508	0	121	137
U-Prove	1000	120	1006	0	120	162
U-Prove	1500	120	1501	0	121	90
U-Prove	500	180	504	0	181	140
U-Prove	1000	180	1004	0	180	63
U-Prove	1500	180	1502	0	181	-26

SetupRevocationAuthorityParameters():

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	500	60	505	1	60	-7
Idemix	1000	60	1003	0	61	247
Idemix	1500	60	1503	0	60	178
Idemix	500	120	506	0	121	189
Idemix	1000	120	1004	0	120	178
Idemix	1500	120	1507	0	120	240
Idemix	500	180	509	0	181	159
Idemix	1000	180	1006	0	183	79
Idemix	1500	180	1503	0	182	-180

SetupInspectionPublicKey():

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	500	60	44372	0	62	144
Idemix	1000	60	43209	0	61	15
Idemix	1500	60	6479	0	61	256
Idemix	500	120	89337	0	122	124
Idemix	1000	120	79381	0	122	263
Idemix	1500	120	88102	0	122	48
Idemix	500	180	130410	0	182	101
Idemix	1000	180	132341	0	183	181
Idemix	1500	180	109491	0	183	256

New Architecture

The column "CE" stands for the Crypto Engine. Columns "k" and "t" are, respectively, the number of concurrent requests and the duration of the test. The column "reqs" is the total number of requests performed during the test. The column "errs" is the number of exception reported. The column "real t" is the real execution time, and, finally, the column "mem" report the memory consumed during the test execution.

SetupSystemParameters():

CE	k	t	reqs.	errs	real t	mem[11]
U-Prove	500	60	539	0	60	11
U-Prove	1000	60	1037	0	60	3
U-Prove	1500	60	1516	0	60	9
U-Prove	500	120	546	0	120	40
U-Prove	1000	120	1026	0	120	21
U-Prove	1500	120	1524	0	121	9
U-Prove	500	180	533	0	180	41
U-Prove	1000	180	1035	0	180	34
U-Prove	1500	180	1522	0	180	34

SetupIssuerParameters():

CE	k	t	reqs.	errs	real t	mem[11]
U-Prove	500	60	513	0	60	62
U-Prove	1000	60	1004	0	60	64
U-Prove	1500	60	1505	0	60	37
U-Prove	500	120	507	0	120	130
U-Prove	1000	120	1003	0	120	11

U-Prove	1500	120	1507	0	120	71
U-Prove	500	180	504	0	180	121
U-Prove	1000	180	1006	0	180	256
U-Prove	1500	180	1504	0	180	137

Scenario 3.1.1: Stress perturbations on the ABCE component (Advanced Issuance)

The column "CE" stands for the Crypto Engine. Columns "k" and "t" are, respectively, the number of concurrent requests and the duration of the test. The column "reqs" is the number of requests performed during the test. The column "errs" is the number of exception reported. The column "real t" is the real execution time, and, finally, the column "mem" report the memory consumed during the test execution.

Old Architecture

CE	k	T	reqs.	errs	real t	mem[11]
Idemix	500	60	21953	21455	61	-15
Idemix	500	120	52712	52213	120	-28
Idemix	500	180	80545	80046	180	43
Idemix	1000	60	22993	21998	60	-160
Idemix	1000	120	51708	50714	120	60
Idemix	1000	180	78754	77760	180	0
Idemix	1500	60	22198	20699	60	39
Idemix	1500	120	40679	39179	120	-6
Idemix	1500	180	45199	43701	180	58

New Architecture

CE	k	T	reqs.	errs	real t	mem[11]
U-Prove	500	60	500	0	60	14
U-Prove	500	120	500	0	120	17
U-Prove	500	180	500	0	145	29
U-Prove	1000	60	1000	0	60	-1
U-Prove	1000	120	1000	0	120	11
U-Prove	1000	180	1000	0	180	23
U-Prove	1500	60	2944	1444	60	33
U-Prove	1500	120	3710	2210	120	47
U-Prove	1500	180	2910	1410	180	49

Scenario 4.1.1: Stress perturbations on the verifier's ABCE component

The column "CE" stands for the Crypto Engine. Columns "k" and "t" are, respectively, the number of concurrent requests and the duration of the test. The column "reqs" is the total number of requests performed during the test. The column "Errs" is the number of exception reported. The column "real t" is the real execution time, and, finally, the column "mem" report the memory consumed during the test execution.

Old Architecture

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	500	60	721	0	360	237
Idemix	500	120	804	0	420	63
Idemix	500	180	799	0	481	43
Idemix	1000	60	1085	0	361	30
Idemix	1000	120	1484	0	420	118
Idemix	1000	180	1573	0	480	128
Idemix	1500	60	1545	0	361	58
Idemix	1500	120	1923	0	420	75
Idemix	1500	180	2171	0	480	124

New Architecture

CE	k	t	reqs.	errs	real t	mem[11]
U-Prove	500	60	1755	0	60	311
U-Prove	500	120	3184	0	120	210
U-Prove	500	180	4516	0	180	131
U-Prove	1000	60	2365	0	60	339
U-Prove	1000	120	3732	0	120	256
U-Prove	1000	180	5033	0	180	269
U-Prove	1500	60	2827	0	60	326
U-Prove	1500	120	4095	0	120	228
U-Prove	1500	180	5368	0	180	249

Scenario 5.1.1: Stress perturbations on the Revocation Authority's ABCE component

The column "CE" stands for the Crypto Engine. Columns "k" and "t" are, respectively, the number of concurrent requests and the duration of the test. The column "reqs" is the total number of requests performed during the test. The column "Errs" is the number of exception reported. The column "real t" is the real execution time, and, finally, the column "mem" report the memory consumed during the test execution.

Old Architecture

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	500	60	4406	146	61	224
Idemix	500	120	6105	32	120	145
Idemix	500	180	7500	21	180	34
Idemix	1000	60	4885	18	60	168
Idemix	1000	120	6542	39	121	123
Idemix	1000	180	7862	38	180	120
Idemix	1500	60	5102	31	60	211
Idemix	1500	120	7066	36	121	151
Idemix	1500	180	8373	45	180	168

Scenario 6.1.1: Stress perturbations on the Inspector's ABCE component

The column "CE" stands for the Crypto Engine. Columns "k" and "t" are, respectively, the number of concurrent requests and the duration of the test. The column "reqs" is the total number of requests performed during the test. The column "Errs" is the number of exception reported. The column "real t" is the real execution time, and, finally, the column "mem" report the memory consumed during the test execution.

Old Architecture

CE	k	t	reqs.	errs	real t	mem[11]
Idemix	50	60	736	0	60	177
Idemix	500	120	1919	0	120	2
Idemix	500	180	2626	0	180	198
Idemix	1000	60	1702	0	60	239
Idemix	1000	120	2408	0	120	217
Idemix	1000	180	3132	0	180	17
Idemix	1500	60	2204	0	60	21
Idemix	1500	120	2917	0	120	28
Idemix	1500	180	3611	0	180	25