

Prototype of a Selected Scenario on Privacy Throughout Life

Editors: Katrin Borcea-Pfitzmann (TUD)
Stefan Köpsell (TUD)

Reviewers: Fatih Gey (EMIC)
Slim Trabelsi (SAP)

Identifier: D1.3.2

Type: Deliverable

Version: 1.0

Class: Public

Date: Wednesday 18th May, 2011

Abstract

This deliverable describes the technical background and design of the demonstrator. Thereby it describes which functionality is already implemented fully and which is still under construction. Finally, an installation guide and a reference to an online usage guide will be given.

Members of the PrimeLife Consortium

1.	IBM Research GmbH	IBM	Switzerland
2.	Unabhängiges Landeszentrum für Datenschutz	ULD	Germany
3.	Technische Universität Dresden	TUD	Germany
4.	Karlstads Universitet	KAU	Sweden
5.	Università degli Studi di Milano	UNIMI	Italy
6.	Johann Wolfgang Goethe - Universität Frankfurt am Main	GUF	Germany
7.	Stichting Katholieke Universiteit Brabant	TILT	Netherlands
8.	GEIE ERCIM	W3C	France
9.	Katholieke Universiteit Leuven	K.U.Leuven	Belgium
10.	Università degli Studi di Bergamo	UNIBG	Italy
11.	Giesecke & Devrient GmbH	GD	Germany
12.	Center for Usability Research & Engineering	CURE	Austria
13.	Europäisches Microsoft Innovations Center GmbH	EMIC	Germany
14.	SAP AG	SAP	Germany
15.	Brown University	UBR	USA

***Disclaimer:** The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The below referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. Copyright 2010, 2011 by Technische Universität Dresden.*

Versions

Version	Date	Description
0.1	March 29, 2011	Skeleton of document
0.2	April 1, 2011	First content integrated
0.3	April 4, 2011	More content integrated and first revisions of content
0.4	April 30, 2011	First consolidated version
0.7	May 2nd, 2011	Review version
1.0	May 18th, 2011	Final Version

List of Contributors

Contributions from several PrimeLife partners are contained in this document. The following list presents the contributors for the chapters of this deliverable.

Chapter	Author(s)
<i>Chapter 1:</i> Introduction	Jaromir Dobiáš (TUD), Katrin Borcea-Pfitzmann (TUD)
<i>Chapter 2:</i> Description of the Demonstrator's Concepts and Functionality	Hagen Wahrig (TUD)
<i>Chapter 3:</i> Architecture of the Demonstrator	Hagen Wahrig (TUD)
<i>Chapter 4:</i> Further Concepts of Lifelong Backup & Synchronisation	Hagen Wahrig (TUD), Jaromir Dobiáš (TUD)
<i>Chapter 5:</i> Installation Guide	Hagen Wahrig (TUD), Katrin Borcea-Pfitzmann (TUD)
<i>Chapter 6:</i> Conclusion	Katrin Borcea-Pfitzmann (TUD), Stefan Köpsell (TUD)

Contents

1	Introduction	9
2	Architecture of the PrimeLife Backup Demonstrator	11
2.1	General Overview of the PrimeLife Backup Demonstrator	11
2.2	Implementation Details	12
3	Description of the Demonstrator’s Concepts and Functionality	17
3.1	Storage Provider	17
3.2	Backup Procedure	18
3.2.1	Packing	18
3.2.2	Encryption	19
3.2.3	Partitioning and Uploading	19
3.3	Restore of Backup	19
3.3.1	Download and Rejoin of Backup Content	20
3.3.2	Decryption	20
3.3.3	Extraction	20
3.4	Delegation	20
3.4.1	Delegation of Access Rights	20
3.4.2	Update of the Delegator Status Stored within the eCV	22
3.4.3	Revocation of Delegated Access Rights	22
3.4.4	Execution of Delegated Access Rights	22
3.5	Trusted Third Party / Electronic Curriculum Vitae	24
3.5.1	Proxy Server	24
3.5.2	eCV Client	25
3.5.3	TTP Client	25
3.6	Configuration	25
3.7	Administrative Data	26
3.7.1	Storage	26
3.7.2	Data Structures	26
4	Further Concepts of Lifelong Backup	29
4.1	Access to Delegated Backup Data Based on Collective Decision	29
4.1.1	Basic Idea of the Collective Decision	29
4.1.2	Technical Implementation of the Collective Decision	30
4.1.3	Reconstruction of the Secret Sharing Sub-key	31
4.2	Concept of Deletion	31
4.2.1	Deletion of Backup	32

4.2.2	Deletion of Backup Task	33
4.2.3	Deletion of Backup Item	34
4.2.4	Deletion of Storage Provider	35
5	Installation & Usage Guide	37
5.1	Installation	37
5.2	Usage	38
6	Summary & Outlook	41
	Glossary	44
	Bibliography	45

List of Figures

1	UML class diagram of the demonstrator's architecture.	13
2	Screenshot of the list of core functions.	14
3	Screenshot of a Web-form of an example core function.	15
4	Internal metadata-structure containing information about a storage provider.	18
5	State diagram of the process of backup.	19
6	Process of restore of backups.	20
7	Distribution of delegation to delegates.	21
8	Update of eCV status by the delegator.	22
9	Revocation of a delegation.	23
10	Restoring a delegation.	23
11	Dependencies between TTP and eCV.	24
12	Data structure of primary data.	26
13	Data structure of delegation.	28
14	Data structure of the address book.	28
15	Relation between backup task and backup.	32

List of Tables

3	Configuration Properties.	27
---	-----------------------------------	----

Chapter 1

Introduction

The technological progress of the last decades triggers transformation of our society towards a computerised social community highly dependent on information. The more structures of our society depend on information the more important is the role that data plays in our everyday lives. During decades of the technological evolution, several methods were invented for storing the data in various forms and on various types of media. However, the processes and events in the nature, society, and in the life of the data subject cause failures, which might lead to loss of its data during the lifetime of the data subject. Even if unwanted data loss might not be a common phenomenon encountered within the life of every data subject, it may become an evident and serious problem, which emerges in the lifelong extent of time.

Many backup systems and backup strategies, which have been available for many years, are already dealing with the problem of unwanted data loss. However, they are mostly protecting the raw data only and do not involve the data subject, his¹ specific characteristics, social relations and interactions as a part of their scope. Existing backup systems and backup strategies also do not reflect the process of evolution of the data subject with respect to possible different states he might pass through during his lifetime and which might have an immense influence on his ability to manage his data on his own behalf (e.g., illness, hospitalisation, or death). Additionally, existing systems and strategies dealing with the problem of unwanted data loss do also not cope with boundaries among distinct areas of the data subject's social interactions. However, these aspects are nowadays becoming more and more sensible on the level of the data, hand in hand with the massive expansion of the technology.

Therefore, we decided to analyse the problem of unwanted data loss from the perspective of lifelong privacy. We found out that current solutions do not provide a sufficient level of data protection when it comes to the lifelong extent of time and privacy of the data subject holding the data. Based on our findings, we decided to demonstrate that it is possible to cope with problems amplified by the requirements on lifelong privacy

¹For the purpose of readability, this document refrains from using gender-neutral pronouns such as “he/she”. Accordingly, gendered pronouns are used in a non-discriminatory sense and are meant to represent both genders.

when protecting the data subject against unwanted data loss.

Our development of the PrimeLife Backup Demonstrator focuses on the following problems that are closely linked together in the light of lifelong privacy:

1. Protection of the data subject against unwanted data loss during his lifetime by redundancy and physical distribution of the data;
2. Assurance of lifelong confidentiality of the data subject's data stored in a distributed environment;
3. Delegation of access rights to the data subject's backup data allowing other parties to operate with his data if specific conditions are fulfilled;
4. Distribution of the backup data according to different areas of life of the data subject and his different partial identities.

Note that this deliverable describes only the (technical) design and implementation of the PrimeLife Backup Demonstrator as well as its installation and usage. For background information related to general requirements, design decisions etc, the interested reader is referred to D1.3.1 "Scenario, Analysis, and Design of Privacy Throughout Life Demonstrator" [WP111b] as well as D1.3.3 "Final Report on Analysis, Scenarios, Requirements and Concepts, Evaluation Results of 'Privacy in Life'" [WP111a].

This deliverable consists of 6 chapters. The problems that are addressed by the PrimeLife Backup Demonstrator and the structure of this document is outlined in this chapter. Chapter 2 explains the architecture of the PrimeLife Backup Demonstrator. In particular, it describes the modular structure of the demonstrator, functional components of the demonstrator and how they interact together. The main concepts and the functionality implemented in the current version of the PrimeLife Backup Demonstrator are described in Chapter 3. Subsequently, Chapter 4 focuses on further concepts of the demonstrator. That is, the concepts and ideas which are still under development and which are gradually being incorporated in the demonstrator are discussed in this chapter. The instruction on how to install the PrimeLife Backup Demonstrator and how to use it is provided in Chapter 5. Finally, this deliverable is summarised in Chapter 6.

Chapter 2

Architecture of the PrimeLife Backup Demonstrator

2.1 General Overview of the PrimeLife Backup Demonstrator

The PrimeLife Backup Demonstrator is an experimental software developed within the PrimeLife project aiming to demonstrate practical applications and concepts of lifelong privacy, data security and identity management in real-life. The PrimeLife Backup Demonstrator is conceived as a backup tool utilising online storage space and providing enhanced delegation capabilities.

It stores the backup data of a primary user in remote storage spaces provided by storage providers. The physical dislocation of the primary user's backup data to the server side protects the data from being lost in case that the corresponding primary items[†] become unintentionally lost (e.g. damaged by a disaster).

Additionally, storing the backup data on the server side makes also delegations possible because, the PrimeLife Backup Demonstrator needs the data to be stored in the online medium in order to be able to delegate it. The delegation protects the primary user's backup data from being lost in such cases when the primary user is temporarily or permanently unable to access his own data.

Storing the backup data in an online storage space, on the other hand, poses a risk to the user's privacy. In order to assure confidentiality of the primary user's backup data, the backup data is encrypted by the PrimeLife Backup Demonstrator prior to storing it in the online media. The encryption/decryption key is then available to the primary user.

In case of delegation the delegates[†] involved gain access to the key (and thus also to the backup data itself) but only if all conditions specified by the primary user for accessing his backup data by the delegates are fulfilled. With respect to delegation the demonstrator deals especially with a scenario where the primary user allows delegates to access his backup under the condition when he is unable to access his data by himself.

This could, e.g., be the case if the primary user is ill, hospitalised or even dead. In our setting such a “status” of the primary user is stored at a third party component called Electronic Curriculum Vitae (eCV). We imagine that the update of the status of the primary user is done by, e.g., his doctor. Concerning the demonstrator, the update can be done by accessing a certain website as described in the online help (cf. Section 5.2).

2.2 Implementation Details

The demonstrator consists of multiple modules corresponding to the individual functionalities (cf. Fig. 1). The two main modules are the user interface, which is a web application and the actual core functionality of the demonstrator which is implemented using Java. The latter runs as background process on the user’s machine.

Data classes use JAXB¹ annotations for specifying primary user’s permission preferences and in order to be easily convertible to XML² or JSON² documents. These are to be stored or to be exchanged between client and server.

The interface to the core functionality is provided by a web-based REST-like API. The list of available functions, grouped according to the modules they belong to, as well as a documentation of them can be found using a web browser under the path `/rest/` of the URL pointing to the internal web server of the core component, i.e. `http://127.0.0.1:8081/rest/` if the default configuration is used (cf. Fig. 2). Moreover, each function can be directly called from within a web browser by providing the required input parameters either in XML or JSON format or by entering the parameters directly in the web forms provided by the web pages generated for each function (cf. Fig. 3). The documentation and the web forms are automatically generated from annotations given for each function within the Java source code. By this means, we would achieve that the documentation of the REST-like API stays in sync with the implementation. Therefore, are not going to include the documentation of the REST-like API in this deliverable as it would be already outdated by the time of reading this document.

Below the main modules and corresponding Java packages are described in more detail.

User interface. The front-end side of the demonstrator, i.e. the graphical user interface of the demonstrator which the users interact with, is implemented using HTML and JavaScript. The related files are provided by the integrated web server of the demonstrator’s core component (back-end). The front-end communicates with the back-end by using the REST-like API.

Main. The main module is implemented in the Java package `bu1`. It provides the code that initialises the background process of the demonstrator. It contains the application’s Main class and a helper class to start the PRIME server which is used for storing administrative data.

¹<http://jaxb.java.net/>

²<http://www.json.org/>

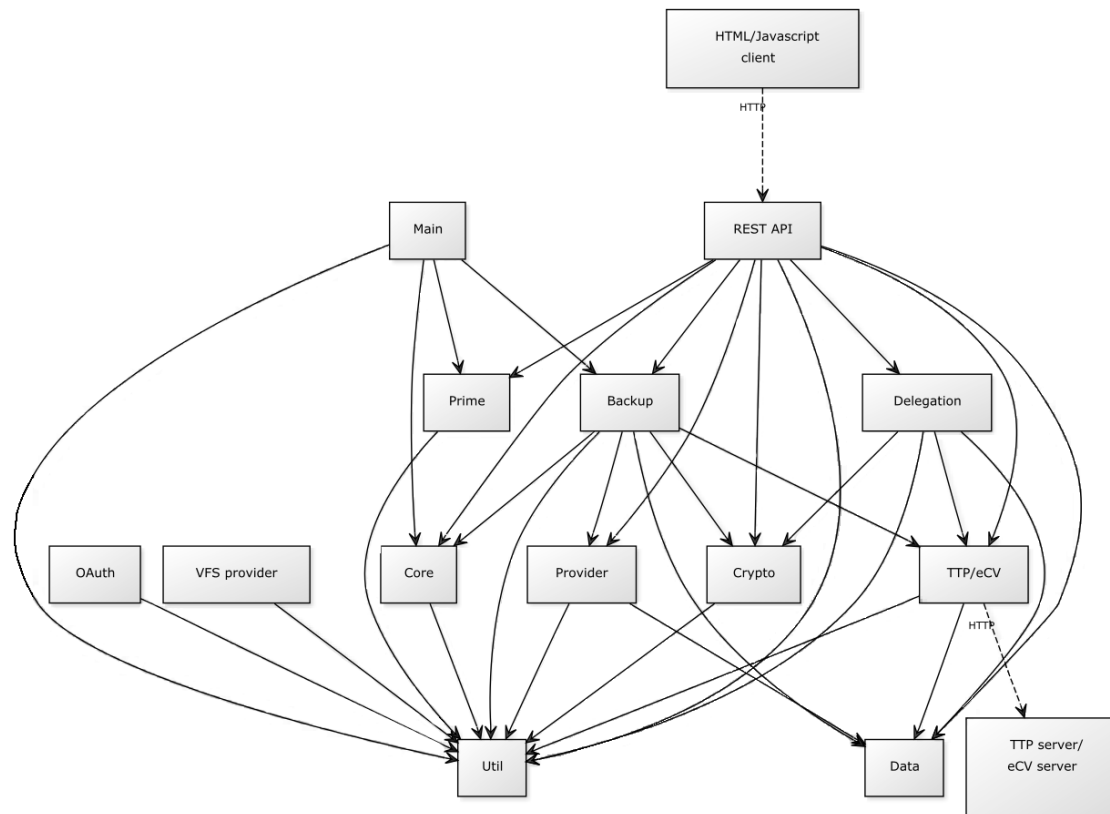


Figure 1: UML class diagram of the demonstrator’s architecture.

Core. The core module is implemented in the Java package `bul.core`. It handles basic functionality like providing tray menu and a web server which delivers the files of the user interface and provides the REST-like API.

REST API. The REST API module is implemented in the Java package `bul.rest`. It bundles the classes implementing the REST-like API. These classes provide methods accessible via HTTP↑ requests, call the methods of the modules implementing the needed functionality, and deliver the response data. To support the data structures expected by the client, the package `bul.rest.data` contains the related data classes.

Backup. The backup module in package `bul.backup` implements the backup and restore functionality. The main functionality for backup is the following:

- packing of the primary items (files) to be backed-up into an archive file;
- encryption of that archive file;
- partitioning and uploading of the archive file to the storage providers.

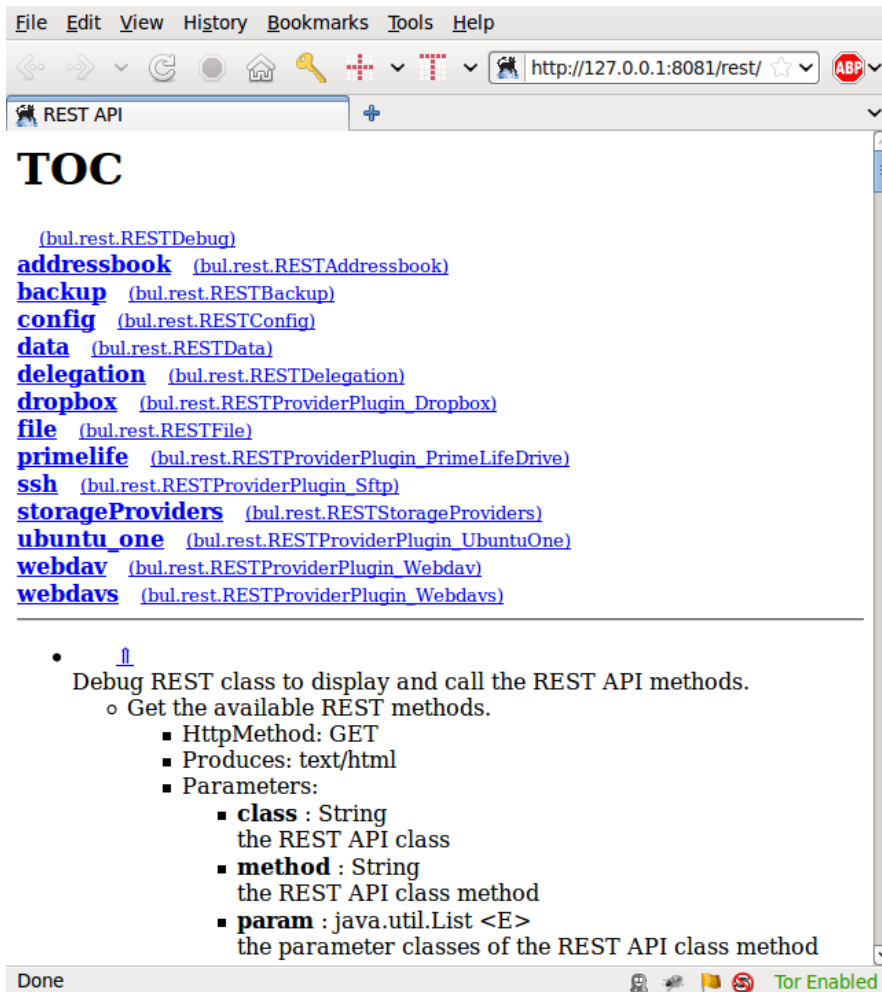


Figure 2: Screenshot of the list of core functions.

The functionality for restoration is correspondingly:

- download and rejoin of the archive file;
- decryption of the archive file;
- extraction of the backed up files from the archive file.

Cryptography. The cryptography module in package `bul.crypto` implements easy-to-use methods for data encryption. Asymmetric encryption is used for delegation of the backup data. The transferred delegation request and response data is encrypted by the public key of the receiver. Symmetric encryption is used for encryption of the archive files. More details about utilisation of various cryptographic mechanisms can be found in Chapters 3 and 4.

The screenshot shows a web browser window with the following elements:

- Address Bar:** `http://127.0.0.1:8081/rest/?class=bul.rest.RESTData&method=createBackupTask¶m`
- Tab:** REST API
- Page Title:** data / createBackupTask
- Description:** Create a new backup task.
- name:** the name of the task (text input field)
- container:** the ID of the container (text input field)
- encrypt:** Determine whether backups have to be encrypted. Default value is 'true'. (dropdown menu)
- providerCredentialId:** the ID of the storage provider credential (three text input fields)
- format:** The data format (json or xml; default: xml) (dropdown menu)
- Submit Query:** A button to submit the form.
- Status Bar:** Done, Tor Enabled

Figure 3: Screenshot of a Web-form of an example core function.

Data. The data module in package `bul.data` implements common used data classes to handle the administrative data, like addressbooks, backups, or delegations.

Delegation. The delegation module in package `bul.delegation` implements functionality to delegate access to a backup to other users (delegates).

OAuth. The OAuth[↑] module in package `bul.oauth` implements functionality for OAuth authentication of the VFS[↑] providers (currently only used by the Dropbox VFS provider).

PRIME. The PRIME module in package `bul.prime` implements functionality for storing data using the PRIME core server.

Provider. The provider module in package `bul.provider` implements functionality to manage the storage providers.

Trusted Third Party / Electronic Curriculum Vitae. This module in package `bul.ttp` implements functionality to access the Trusted Third Party (TTP) and the Electronic Curriculum Vitae (eCV) server.

Util. The `util` module in package `bul.util` implements shared util functionality, mainly in static methods. Important functions are handling configuration data, marshaling XML and JSON data, and an interface to store the administrative data.

VFS providers. The demonstrator utilises the Apache Commons VFS library³ for accessing the online storage providers. This open source library enables access to various different local and remote file systems by a single API. The VFS providers module in the Java package `bul.vfs.providers` contains implementations of additional VFS providers not contained in the Apache VFS implementation. One of them is the webdav (“Secure WebDAV”, WebDAV↑ with SSL↑) VFS provider. The other one is the Dropbox VFS provider to store the backups with the Dropbox service. We also tried to implement a VFS module for the Ubuntu One storage provider but we did not succeed because the authentication using OAuth does not work as expected.

Trusted Third Party service / Electronic Curriculum Vitae service. This is software component which is not part of the PrimeLife Backup Demonstrator software itself. It can be rather seen as some kind of mockup service implementing functionality which is, on the one hand, needed for executing the delegation use case, and on the other hand, would be – in real life – provided by external service providers. The main functionality of this component can be described as external storage which utilises PrimeLife access control policies for controlling the access to the stored data. This feature makes the difference compared to the legacy online storage providers used for storing the backup data.

³<http://commons.apache.org/vfs/>

Chapter 3

Description of the Demonstrator's Concepts and Functionality

3.1 Storage Provider

Storage providers are external entities, i.e., not developed as components of the PrimeLife Backup Demonstrator, providing remote storage space accessible online. The PrimeLife Backup Demonstrator supports several storage providers which can be used for storing the backup data in remote locations.

Storage providers are providing their services independently from the PrimeLife Backup Demonstrator. As there is no common interface for communication with the storage providers, the PrimeLife Backup Demonstrator has a dedicated plugin for each supported storage provider. These plugins are acting as a middleware assuring that the backup core functionality implemented by the PrimeLife Backup Demonstrator is translated into a set of protocol-specific calls interpreted by the corresponding storage provider.

In the current version of the PrimeLife Backup Demonstrator, the plugins of the WebDAV and Dropbox storage providers are already integrated enabling to utilise WebDAV and Dropbox services as the underlying storage for the backup data. In the current implementation, the plugins of supported storage providers are tightly coupled with the core of the PrimeLife Backup Demonstrator. However, the future concept of the demonstrator assumes that there is an online lookup service providing a suitable plugin on demand. In the simplest case, the plugin could be provided by the online storage provider itself.

At least one storage provider must be associated with the PrimeLife Backup Demonstrator in order to be able to create backups.

As soon as a storage provider is associated with the PrimeLife Backup Demonstrator, the information describing the storage provider is contained in an internal metadata-structure. The object of the internal metadata-structure describing a storage provider consists of three parts (see Figure 4):

Provider type: defines the general protocols to be used for the communication with the remote storage service like *WebDAV* or *Dropbox*.

Provider: represents a concrete storage provider. It is in relation with *provider type* object and contains information about the hosting storage location, e.g., the information about the company operating an associated server is contained in the *provider* object.

Provider credential: is an object containing credentials enabling access to the particular provider's storage space. It is in relation with the *provider* object.

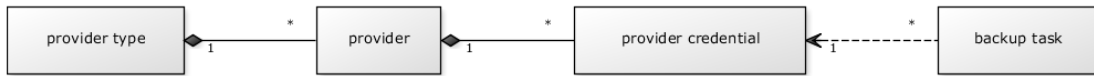


Figure 4: Internal metadata-structure containing information about a storage provider.

The *Provider credential* object is referenced by the *backup task* object as the *provider credential* object represents the actual storage provider entity that is ready for use. It means that, in order to be able to perform backup operations, the presence of a *provider credential* as part of the metadata-structure describing a given storage provider, is necessary. Deletion of a storage provider, or in other words, de-association of an existing user account related to a particular storage provider, from the PrimeLife Backup Demonstrator is therefore a non-trivial task.

3.2 Backup Procedure

Conceptually different logical entities play together in the process of creating a backup of a selection of primary items. A *backup task* describes which primary items should be backed up to which online storage providers in which situations (e.g. on a daily or weekly base, only once etc.). The output of a run of a backup task is called *backup*. Primary items can be grouped together within so called *containers* whereas a container could contain other containers. Note that the selection of primary items which will become part of a given container is not only based on files (as in today's existing backup solutions) but additionally on partial identities, stages of life and areas of life.

Backed-up files can be found in archives. In this way, the set of files that are of a single backup task are stored together. The archive file(s) are then encrypted¹ and uploaded into the remote storage of the selected storage provider(s) (see Figure 5).

3.2.1 Packing

The archive file containing the backup data is a *jar* file. As far as *jar* is in principle an enhanced *zip* format with the ability to store metadata, all data belonging to a backup

¹Note that for debugging reasons it is possible to create and upload unencrypted backups.

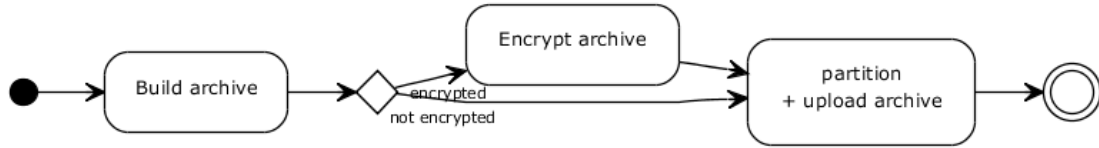


Figure 5: State diagram of the process of backup.

can be put into a single `jar` file. Metadata contains file names of the already processed backup items and information about the container used for the corresponding backup items.

3.2.2 Encryption

To ensure confidentiality of the backup data independently from the trustworthiness of the storage provider, the archive file is encrypted before it is uploaded.

Every archive file uses a unique symmetric key k_{Backup} . By default, an AES encryption algorithm with the key size of 128 bits is used for encryption. Encryption is implicitly enabled but, if explicitly required, the archive file can remain unencrypted.

3.2.3 Partitioning and Uploading

Archive files are stored at a storage space provided by the supported storage providers. If multiple storage providers are selected for the backup, then the archive file is split into parts of equal size (depending on the archive file size, the last split may be shorter), and each part is then uploaded to its corresponding storage space.

Future versions of the PrimeLife Backup Demonstrator ² will incorporate an advanced sharing mechanisms which not only considers the desired level of confidentiality but also the desired level of availability. Thus the output of that mechanism would be two parameters k and n whereas k out of n parts are necessary and sufficient to reconstruct the backup.

The upload of the archive files is realised by utilising the Apache Commons Virtual File System (VFS) in cooperation with the provider specific plugins.

3.3 Restore of Backup

To restore a backup, the inverse way of backup procedure has to be passed, i.e. the archive file(s) will be downloaded, decrypted and the requested files will be extracted (cf. Fig. 6).

²We plan to continue developing the PrimeLife Backup Demonstrator even after the end of the PrimeLife project.

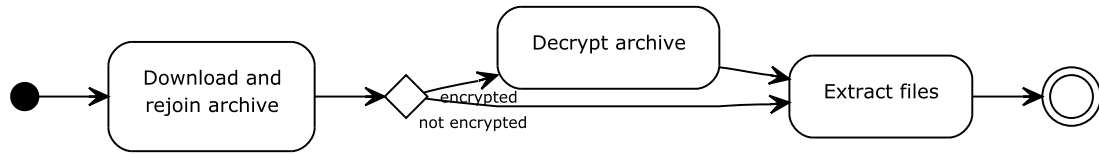


Figure 6: Process of restore of backups.

3.3.1 Download and Rejoin of Backup Content

The one or more parts of the archive file are downloaded from the storage space(s) and – if consisting of multiple parts – rejoined to the archive file.

3.3.2 Decryption

If the archive file is encrypted, then it has to be decrypted during this step. The decryption key is stored in the backup administration data in the user's database or (on delegate's side) reconstructed from the delegation data and the TTP key part.

3.3.3 Extraction

The items selected by the user are extracted from the archive file to a local directory of the choice of the user.

3.4 Delegation

Access to backups can be delegated to other users called delegates. If the delegation is bound to conditions, then a Trusted Third Party (TTP) is used to handle a part of the encryption key of the backup. This part of the key is protected by policies, so that the key part is only accessible if the conditions are fulfilled.

With respect to delegation, four different use cases can be differentiated: delegation of access rights, update of the delegator's status stored within the eCV, revocation of delegated access rights, and execution of delegated access rights.

3.4.1 Delegation of Access Rights

First, the delegator has to distribute the delegations to the delegates (cf. Fig. 7).

Therefore a `Delegation` object is created on the delegator's side containing:

- the backup administrative data,
- the address data of the delegator (name, e-mail address, public key) to get encrypted responses by the delegates,
- the access control policy,

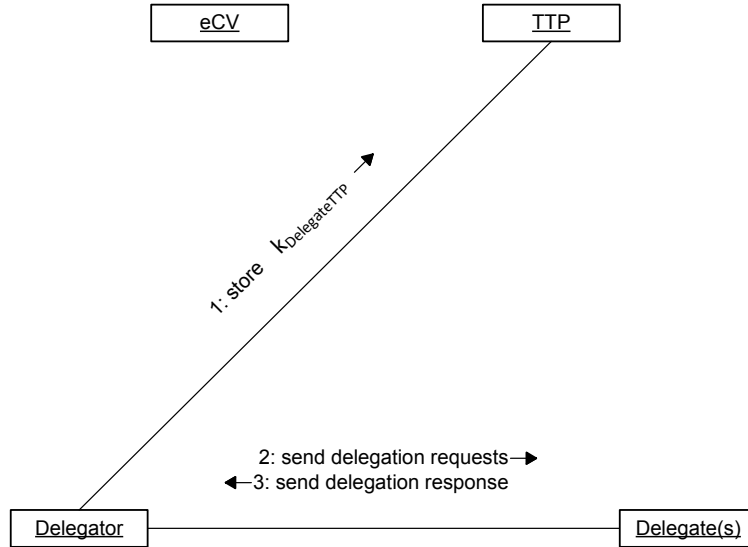


Figure 7: Distribution of delegation to delegates.

- a credential, used by the delegate to prove that he is in fact an authorised delegate,
- the number of delegates.

For every delegate, the encryption key of the backup k_{Backup} is split into two parts k_{Delegate} and $k_{\text{DelegateTTP}}$, whereas: $k_{\text{Backup}} = k_{\text{Delegate}} \oplus k_{\text{DelegateTTP}}$. $k_{\text{DelegateTTP}}$ is stored at the TTP protected by an access control policy. This policy basically describes under which circumstance the delegate can access $k_{\text{DelegateTTP}}$ (and therefore the backup of the delegator), e.g. only for a certain time frame, if the delegator has a certain status (ill, hospitalised, dead) etc. The other part (k_{Delegate}) is stored within the backup administrative data sent to the delegate. The address data of the delegator are put into the Delegation object as well as information how to reach the TTP and the eCV. The Delegation object is sent as encrypted e-mail attachment to the delegate that is called *delegation request*.

Finally, the Delegation object with the address data of all delegates is stored in a database within the delegator’s PrimeLife Backup Demonstrator. Thereby the acceptance status of the delegation itself as well as every related delegate is set to **pending**.

After a delegate opens the e-mail attachment, the Delegation object is put into the database of his PrimeLife Backup Demonstrator. Each delegate can decide if he accepts or rejects the delegation request. The corresponding acceptance status is therefore changed to **accepted** or **rejected**. The decision is sent to the delegator – again as encrypted e-mail attachment called *delegation response*.

Finally, after the delegator opens a delegation response attachment, the acceptance status of the related delegate of the corresponding delegation is updated within the delegator’s database.

3.4.2 Update of the Delegator Status Stored within the eCV

As stated in the previous section, the access control policy for $k_{\text{DelegateTTP}}$ might contain the status of the delegator (ill, hospitalised, dead). In our demonstration scenario, the status of the delegator is stored within the eCV of the delegator. Therefore, we implemented a simple mechanism which allows the delegator to update his status (cf. Fig. 8).

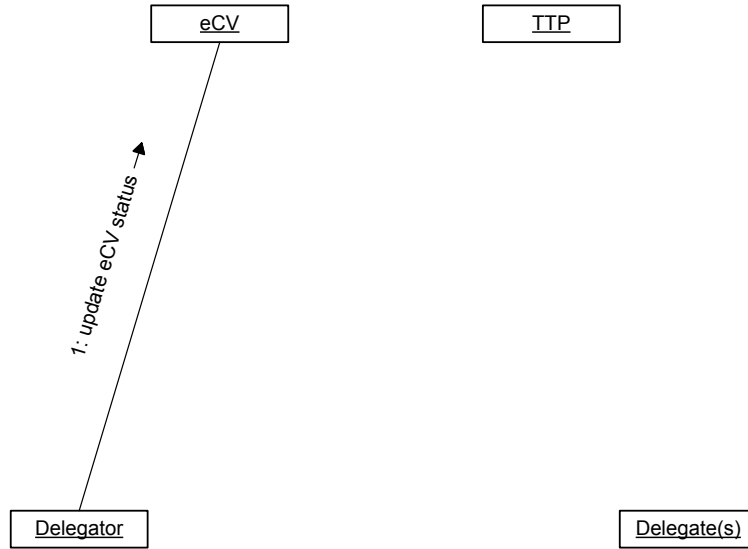


Figure 8: Update of eCV status by the delegator.

3.4.3 Revocation of Delegated Access Rights

The delegator can revoke the delegation so that the delegates do not have access to the backup anymore (cf. Fig. 9).

From a technical point of view, revocation means that the corresponding keys $k_{\text{DelegateTTP}}$ stored at the TTP are deleted – so the delegates cannot reconstruct the encryption key k_{Backup} . Moreover, a revocation message is sent to the delegates as e-mail attachment and the delegation is marked as “revoked” within the delegator’s database.

If a delegate opens the revocation e-mail attachment, the delegation stored within his database is also marked as “revoked”.

3.4.4 Execution of Delegated Access Rights

The goal of a delegation is that delegates can restore a backup of the delegator (cf. Fig. 10).

In order to restore a backup, a delegate has to reconstruct the encryption key of that backup. Thereafter the backup can be restored in the same way as ordinary backups.

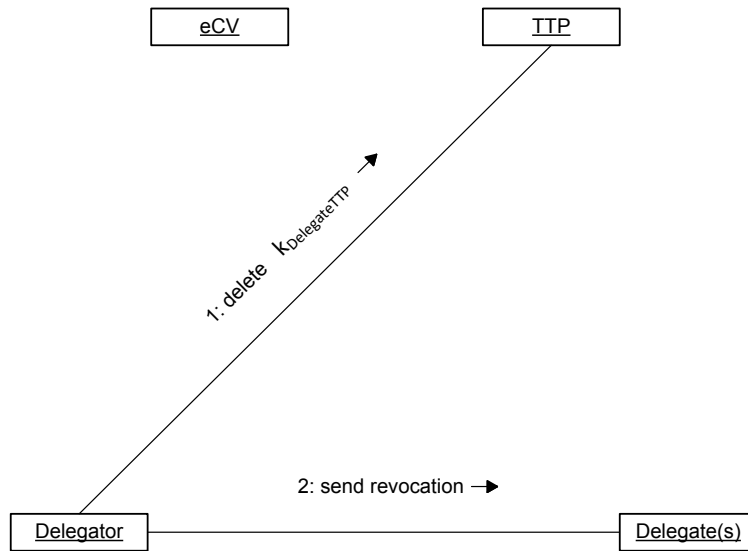


Figure 9: Revocation of a delegation.

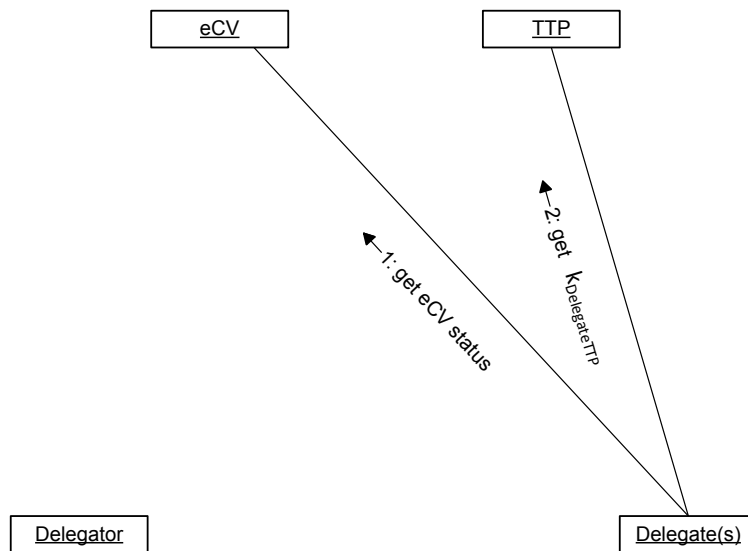


Figure 10: Restoring a delegation.

This in turn means that a delegate needs to get access to $k_{\text{DelegateTTP}}$. Therefore he communicates with the TTP. Depending on the access control policy, the delegate needs to show various (anonymous) credentials. This comprises on the one hand a credential proving that he is in fact an authorised delegate. On the other hand, a credential proving the current status of the delegator might be necessary. In such a case, the delegate will request the status of the delegator from the eCV. Again, the delegate needs to show a credential proving that he is allowed to access the status of the delegator. Finally the delegate will pass the status data received from the eCV to the TTP. If access is permitted, he will get $k_{\text{DelegateTTP}}$.

3.5 Trusted Third Party / Electronic Curriculum Vitae

Delegation as realised within the demonstrator (cf. Sec. 3.4) requires the availability of a Trusted Third Party (TTP). Further, it is supported by the Electronic Curriculum Vitae (eCV). This section gives an overview of the current implementation and its use in the PrimeLife Backup Demonstrator.

TTP and eCV consist of two parts: a client side in the PrimeLife Backup Demonstrator application, and a server side (currently identical for TTP and eCV) working as proxy to access the PrimeLife Policy Engine [WP511]. The proxy server is an extra application in an extra source code project and can run on a different host than the PrimeLife Backup Demonstrator application (cf. Fig. 11).

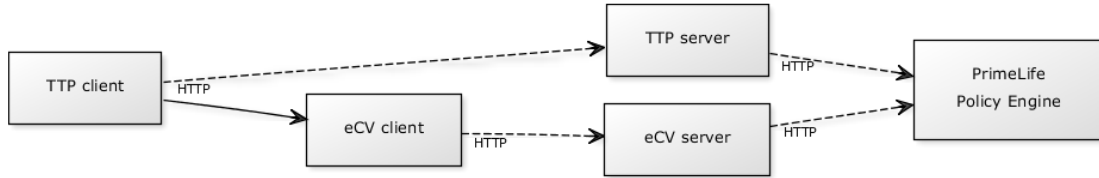


Figure 11: Dependencies between TTP and eCV.

3.5.1 Proxy Server

The proxy server has two functions: storing data in the PrimeLife Policy Engine using data handling policies and retrieving data from the PrimeLife Policy Engine showing the necessary credentials to fulfil the data handling policies.

If a $k_{\text{DelegateTTP}}$ should be stored at the TTP, the TTP client communicates with related access conditions to the TTP using so called *simple policies*. They can be seen as XACML-like policies but tailored to the functionality needed by the PrimeLife Backup Demonstrator. The TTP proxy server builds a PPL `sticky policy` containing the data handling policy generated from the passed simple policy. It stores the data ($k_{\text{DelegateTTP}}$) in the PrimeLife Policy Engine by creating a new PII with a new PII ID or – when an existing PII ID is passed – updates the existing PII.

In order to retrieve data, the proxy server builds a request policy from the passed credentials, and requests the data from the PrimeLife Policy Engine.

The REST API for TTP server and eCV server have the same methods. Although they share the source code, both servers can run at different hosts.

3.5.2 eCV Client

The status data stored at the eCV are not protected explicitly, everybody who knows the PII ID can read the data without passing a password or other credentials.

For privacy reasons, the status values are not stored as plain text. They are mapped to random strings which are stored as status values. Thus, in the conditions expressed in the simple policy concerning the status of the delegator (ill, hospitalised etc.), the status values are also replaced by these random strings while creating the PPL data handling policy. Only the owner of the eCV data knows its mapping.

3.5.3 TTP Client

The TTP client component ensures on storage that all preparations of the eCV are done, e.g., it will create a PII ID of the PII containing the eCV status relevant to access to the TTP data.

When retrieving data, the TTP client component will automatically access the eCV in order to retrieve the delegator's status if that status is needed as credential in order to access the data stored at the TTP.

3.6 Configuration

In order to be able to easily configure settings of the PrimeLife Backup Demonstrator, the demonstrator provides configuration.

Configuration consists of multiple levels, thereby a property set in a higher level hides all properties of the same name that has been potentially set in lower levels. Currently, the following levels exist (in ascending order):

- *default*: If a property value is necessary for functionality, then default values will be provided by the implementation itself. Default values are defined in property files (mostly `default.properties`) in the package of the modules using the property and loaded on initialisation of the module.
- *config file*: Default properties can be overridden by properties defined in property files in the file system. By default, `bul.properties` and `config/bul.properties` are loaded (if they exist). An additional file can be given by the `bul.configFile` property.
- *user specific*: User specific properties override config file properties and default properties. They are stored using the `Preferences` framework³. Values can be set using the `rest/config` REST API calls.

³<http://download.oracle.com/javase/6/docs/api/java/util/prefs/Preferences.html>; stored OS implementation depending, e.g. in the Registry (Windows) or flat files (Linux)

- *command line*: Command line properties are the top level. They are passed as command line arguments in the form of `-name=value` where **name** is the property name and **value** is the value to set.

Table 3 shows important properties used in the current implementation of the PrimeLife Backup Demonstrator.

3.7 Administrative Data

Administrative data are needed to manage the functionality. They contain, for instance, an address book, data about the sent and received delegations, data about storage providers, and data about the managed files and existing backups.

3.7.1 Storage

The administrative data are stored as XML strings in the PRIME core, which runs embedded in the PrimeLife Backup Demonstrator. An alternative implementation can be used to store the data using the Preferences framework instead of the PRIME core. Marshalling/Unmarshalling to and from XML is done by the *Java Architecture for XML Binding* (JAXB).

3.7.2 Data Structures

Primary backup data. The core functionality – namely backup and restore of files – deals with the **primary data** data structure depicted in Fig. 12.

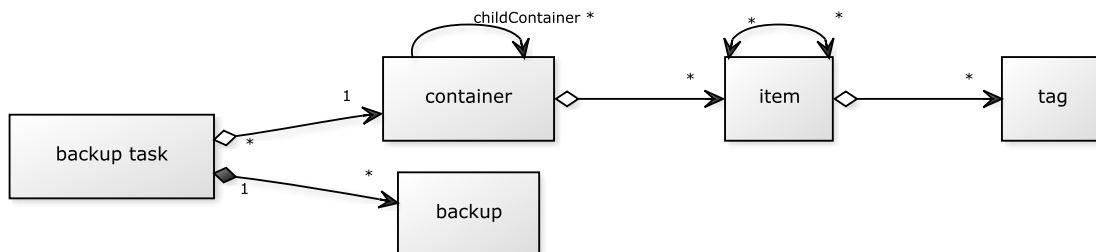


Figure 12: Data structure of primary data.

Accordingly, the **backup task** represents a description of **backups**. It indicates what (the container holding the items to be backed up) shall be stored where (the storage provider including the credentials to access it) and when (schedule of backups). Thus, it refers to the **container** which contains the **items** to be backed up. Items are one of the following types: areas of life, partial identities, files or folders. An **item** object itself can refer to other **items** of a different type. To give an example, an area of life refers to the partial identities used in this area of life, and both of them refer to files or folders used in the area of life or/and used by the partial identities. Items of type **files** or **folders**

Table 3: Configuration Properties.

Name	Default value	Description
bul.host	127.0.0.1	The host of the back-end web server. Needed to build the URL for the GUI.
bul.port	8081	The port of the back-end server
bul.encryption.symmetric.algorithm	AES	The algorithm used for symmetric encryption, e.g., used for encryption of the backup archive files
bul.encryption.symmetric.keysize	128	The key size used for creation of symmetric encryption keys
bul.encryption.asymmetric.algorithm	RSA	The algorithm used for asymmetric encryption, e.g., used for encryption of the delegation requests/responses
bul.encryption.asymmetric.keysize	1024	The key size used for creation of asymmetric encryption keys
ttp.address	http://localhost:8084/rest/ttp	The address of the TTP server used by the delegator for delegation.
ecv.address	http://localhost:8084/rest/ecv	The address of the eCV server holding the status of the delegator.
bul.userDataImpl	bul.util.UserData.DefaultImpl	The name of the class used to store administrative data. Existing implementations: - <code>bul.util.UserData.DefaultImpl</code> : stores data using the references framework - <code>bul.prime.PrimeUserData</code> : stores data using the PRIME core (set when the embedded PRIME core server is started)
mail.smtp.host	127.0.0.1	The SMTP server used for sending delegation e-mails (requests, responses, revocations). The mail properties are passed to the JavaMail framework, so other properties can be passed to JavaMail using the configuration mechanism.
java.util.logging.config.file		The configuration file for the Java Logging framework ⁴ .
prime.port	9907	The port of the PRIME server
prime.dataDir	my_temp	The directory where the PRIME server stores its data
prime.keystore	build/crypto/prime.ks	The keystore containing the SSL certificate of the PRIME server and other needed SSL certificates
in the TTP proxy:		
bulTTP.host	127.0.0.1	The host of the TTP/eCV proxy
bulTTP.port	8084	The port of the TTP/eCV proxy

designate files or folders of the file system, the absolute file path is contained within the data structure item. When a backup was accomplished, a new **backup** object is created and assigned to the corresponding **backup task** object. That way, the **backup** object can be used to perform a restore of the data.

Containers can consist of other containers to include also their items.

Delegation data. Delegation data contains a copy of the backup object to delegate and a policy describing the conditions under which the delegates shall have access to the backed up files. This delegation object is sent to the delegates on delegation (cf. Fig. 13).

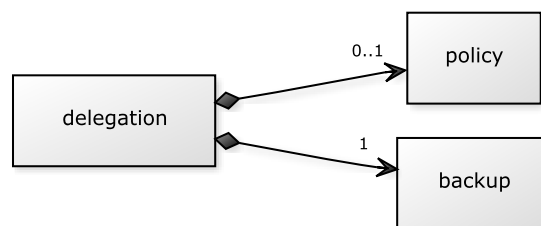


Figure 13: Data structure of delegation.

Address book data. The address book contains address data about all identities of a given user and about individuals, groups and roles known to him. All these address items can be tagged (cf. Fig. 14). These tags will be used by the user interface to allow easy grouping of and searching for address book entries.

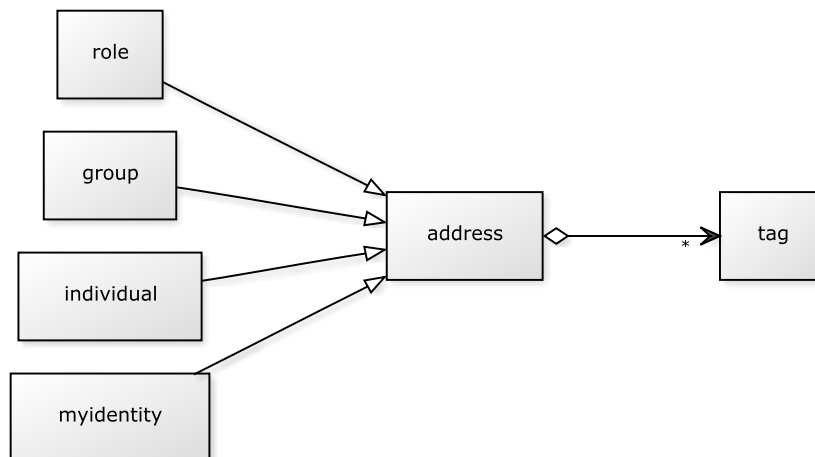


Figure 14: Data structure of the address book.

Further Concepts of Lifelong Backup

The previous chapters describe the current status of implementation of the PrimeLife Backup Demonstrator. Here, we will discuss further concepts on a theoretical basis that enhances the actual demonstrator. Please note that despite using the present tense in this chapter, the concepts and ideas presented here are currently under development and will appear in future versions of the PrimeLife Backup Demonstrator planned to be released until the end of the PrimeLife project.

4.1 Access to Delegated Backup Data Based on Collective Decision

4.1.1 Basic Idea of the Collective Decision

By delegating access rights to one or more delegates, a primary user gives these delegates conditional permission to access his own backup data. Even though the access to the primary user's backup data can be restricted to a certain time-frame or can be conditioned by the primary user's inactivity or by reaching a specific personal state of the primary user (e.g. illness, hospitalisation or death), in all of these cases, each delegate involved is able to recover the backup data of the primary user on his own as soon as the conditions specified by the primary user are fulfilled. However, in the real-life, specific situations might occur, where the decision whether the primary user's backup data should be accessed by a third party or not should not rely solely on the pure machine logic. If, e.g., the illness of the primary user does not limit the primary user's ability to operate with his data on his own or the hospitalisation of the primary user is only short-time, application of access rights based on the pure machine logic could even increase the risk of non-legitimate access to potentially sensitive backup data of the primary user.

It is currently difficult to automatically evaluate all aspects of real-life situations

which have an impact on the primary user’s backup data (e.g., inactivity, illness or hospitalisation of the primary user) and let the machine decide if there is a legitimate reason for giving a delegate permission for accessing the primary user’s backup data. This becomes especially apparent as soon as the life-long extent of time is concerned.

In order to eliminate the risk of providing the delegates non-legitimate access to the primary user’s backup data, the PrimeLife Backup Demonstrator provides the *collective decision access right* incorporating the factor of a collective human decision into the delegate-invoked backup restoration. Collective decision access rule enables to distribute the power of restoring the potentially sensitive backup data of the primary user to multiple delegates. The collective decision access rule requires that, for the delegates to be able to restore the primary user’s backup data, a predefined amount of delegates involved (specified by the primary user) must participate in the restoration. Thus, in case of the collective decision, the primary user’s backup data is accessible to delegates involved, but only if a sufficient number k of the n delegates participate in the restoration process.

4.1.2 Technical Implementation of the Collective Decision

The PrimeLife Backup Demonstrator utilises Shamir’s Secret Sharing algorithm as the underlying mechanism of the collective decision. Shamir’s Secret Sharing algorithm divides data D into n pieces in such a way that D is easily reconstructable from any k pieces, but even complete knowledge of $k-1$ pieces reveals no information about D .

In the collective decision scenario, the encryption key of the backup k_{Backup} is split into three parts (4.1): a part sent to the delegate (k_{Delegate}), a part stored at the TTP ($k_{\text{DelegateTTP}}$), and a part used for secret sharing ($k_{\text{SecretSharing}}$). All three parts XOR combined compose the encryption key of the backup. Shamir’s Secret Sharing algorithm is applied to the sub-key $k_{\text{SecretSharing}}$. Thus, the sub-key $k_{\text{SecretSharing}}$ is split into parts $S_1 \dots S_n$ called shares, where n is the number of delegates involved in the particular delegation. Note that the delegator also needs to decide about the parameter k , i.e. how many delegates are necessary and sufficient to restore the delegate backup data.

$$k_{\text{Backup}} = k_{\text{Delegate}} \oplus k_{\text{DelegateTTP}} \oplus k_{\text{SecretSharing}} \quad (4.1)$$

$$k_{\text{SecretSharing}} = f(S_1 \dots S_n) \quad (4.2)$$

with:

- k_{Backup} : symmetric encryption key of the backup
- k_{Delegate} : delegate specific key part sent in the delegation request
- $k_{\text{DelegateTTP}}$: delegate specific key part stored at the TTP
- $k_{\text{SecretSharing}}$: key part provided for secret sharing to realize conditions like “2 out of 3 delegates have to participate to restore the primary user’s backup data”
- $S_1 \dots S_n$: shares of the sub-key $k_{\text{SecretSharing}}$ provided to the delegates

Note that k_{Delegate} and $k_{\text{DelegateTTP}}$ are unique for each delegate involved (as described in section 3.4), but the result of $k_{\text{Delegate}} \oplus k_{\text{DelegateTTP}}$ is the same for each delegate.

In the process of delegation utilising the collective decision access condition, each share $S_1 \dots S_n$ is distributed to the corresponding delegate contained in the delegation request (each delegate receives exactly one share).

4.1.3 Reconstruction of the Secret Sharing Sub-key

The secret sharing algorithm can reconstruct the sub-key $k_{\text{SecretSharing}}$ from a subset of k shares of $S_1 \dots S_n$. It means, that usually not all shares are needed for the reconstruction of the sub-key $k_{\text{SecretSharing}}$. Thus not all n delegates must participate in the restoration process in order to restore the delegated backup data.

During restoration of the backup delegated by utilising the collective decision access condition, each delegate who decides to participate in the backup restoration uploads his share encrypted to the TTP. The encryption key used is $k_{\text{Delegate}} \oplus k_{\text{DelegateTTP}}$. Afterwards, participating delegates send the *secret sharing key request* to the TTP requesting the shares from other delegates involved, identified by the share IDs. The list of the share IDs is available to all delegates involved as far as it is contained in the delegation request when establishing the delegation. Afterwards, the TTP returns back the content corresponding to the requested share IDs, i.e. the encrypted shares.

After receiving the *secret sharing key response* from the TTP, the participating delegate processes the content and checks if there is a sufficient number of shares contained in the response. If not, the participating delegate sends a new *secret sharing key request* later on.

As soon as a sufficient amount of delegates involved participated in the backup restoration, the TTP provides them a sufficient amount of shares for reconstruction of the sub-key $k_{\text{SecretSharing}}$. Afterwards, this sub-key is reconstructed by each participating delegate involved locally. Then, the symmetric encryption key k_{Backup} is composed and the primary user's backup data downloaded from the external storage location is decrypted by the delegates.

4.2 Concept of Deletion

The PrimeLife Backup Demonstrator takes advantages of external resources¹ provided by the online storage providers, the TTP and the eCV. Apart from them, the demonstrator communicates with external entities acting as delegates of a particular backup.

By utilising the PrimeLife Backup Demonstrator in a full manner (i.e., creating the backup and delegating the corresponding access rights to the backup data to one or more delegates), the user of the demonstrator creates an explicit relation among his backup data, underlying storage provider(s) and one or more delegates receiving access rights to the backup data. This forms a distributed environment, where actions performed on the user's backup data are reflected to the underlying storage providers and to the

¹We refer to *external* resource in case the resource is not under direct control of the resource user.

delegates involved. Hence, all aspects and privacy implications of actions performed by utilising the PrimeLife Backup Demonstrator need to be concerned.

Deletion of personal data on request of its owner or on satisfaction of certain conditions (such as the purpose for storing personal data at a service's side is no longer afforded) is one of the main legal requirements as stated in [WP111b].

Thus in this section, we are particularly discussing the aspects of the *deletion* in the PrimeLife Backup Demonstrator from the functional and privacy point of view.

4.2.1 Deletion of Backup

An important function of the PrimeLife Backup Demonstrator is the deletion of a backup. A backup is conceived as a 1:1 copy of a defined set of the user's data performed at a particular time. The source data items from which the copy is made are called primary items, while 1:1 copies of the primary items contained in the backup stored online are referred to as backup items. In the core of the PrimeLife Backup Demonstrator, each backup is internally represented by its corresponding meta data. The meta data describing a backup is structured such that each backup is an element a single backup task only, while a backup task can contain several backups (see Figure 15).

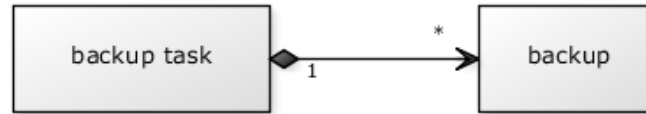


Figure 15: Relation between backup task and backup.

Soft deletion of the backup

From the perspective of the PrimeLife Backup Demonstrator, the backup is deleted, if no corresponding meta data related to that particular backup exists in the demonstrator. The deletion of meta data of a particular backup together with the deletion of its corresponding encryption key is utilised for the so called *soft deletion* of a backup in the PrimeLife Backup Demonstrator.

If the user performs soft deletion of his backup, the backup data still remains online in an encrypted form, but the (soft) deleted backup data can be no longer seen listed in the demonstrator.

The advantage of this type of deletion is that, as far as the backup data remains untouched online, it can be restored by the delegates involved, even after performing the deletion. This, however, makes sense only if the backup has been delegated to one or more delegates before it is deleted. A disadvantage of the soft deletion is that the backup data still consumes online storage space, though it no longer exists in the user's instance of the PrimeLife Backup Demonstrator.

Hard deletion of the backup

Physical deletion of the online backup data together with the deletion of the corresponding meta data is the second type of deletion (so called *hard deletion*) supported by the PrimeLife Backup Demonstrator. By performing the hard deletion of the backup data, not only the user loses access to his backup data but also the delegate(s) involved possessing rights to access the backup data. That is, as soon as the backup data is physically erased from the online storage, there is no reason for the involved delegates to have the right to access the backup data. Therefore, all delegations related to the backup data, which is physically deleted, are needed to be automatically revoked.

However, there is an additional aspect which needs to be taken into consideration as far as the automatic revocation is concerned. When the backup data is deleted from the online storage location, the automatic revocation triggered by the deletion of the backup data may affect the primary user's privacy. Therefore, the primary user of the PrimeLife Backup Demonstrator must have the ability for not letting the delegates know that the backup data was deleted. The proposed solutions of this problem are the following:

1. disguising the deletion of the backup data by sending the ordinary revocation request without revealing that the reason for revocation was the backup deletion;
2. postponing sending the revocation request to a predefined time in the future and possibly without revealing that the reason for revocation was the backup deletion;
3. postponing sending the revocation request to the time when the delegate tries to apply his delegation for accessing the deleted backup data and possibly without revealing that the reason for revocation was the backup deletion.

Regarding physical deletion of the backup data, one has to consider that the online storage providers implement their own data-handling strategies (e.g. backup/mirroring the user data), which are out of control of the PrimeLife Backup Demonstrator. This means that the PrimeLife Backup Demonstrator can not assure that the online data marked as deleted is really deleted. Therefore, the PrimeLife Backup Demonstrator considers the backup data to be physically deleted as soon as the encryption key and all of its parts are securely deleted (e.g. from the TTP) in addition to have an indication from the underlying storage providers that the requested backup data was physically deleted.

4.2.2 Deletion of Backup Task

The *Backup task* data structure is the main internal data structure of the PrimeLife Backup Demonstrator holding information about which data should be backed up to which storage provider(s) under which circumstances. A new backup task object is always created as soon as the user interacts with the user interface of the demonstrator in order to back up a set of his data (primary items). Usually one or more backups are associated with a given backup task (see Figure 15) where each backup reflects the current state of the involved backup items at a particular time.

By deleting a backup task, all backups associated with that particular backup task are deleted as well. The deletion of the backup task can be performed in soft or hard

manner, using the same principles as described in case of deletion of a single backup. This means, if the soft deletion of backup task is performed, all associated backups are deleted in the soft manner. Similarly, in case of a hard deletion of a backup task all associated backups are deleted in the hard way.

Apart from deletion, it is possible to put the backup task into inactive state. In an inactive state all associated backups remain reachable for the user and the delegates involved. However, all scheduled operations like creation of a new backup or an update of an existing backup are disabled.

4.2.3 Deletion of Backup Item

A Backup item is a labelled unit of backup data stored in the online backup storage space. A backup item retains the state of the corresponding primary item (the local data of the user) from a particular time. As far as the backup item can be physically distributed across different online storage spaces in redundant copies, the deletion of a backup item should remove each copy from every backup which stores it.

It is fairly easy to perform deletion of a backup item if that backup item remained all the time in the online storage space (in its encrypted form). The situation, however, becomes more complicated as soon as one or more delegates gain access to data contained in the backup item. The problem is that, once the backup data becomes available to a delegate, the data is no longer under control of the primary user. The delegate can always create a copy of the data, sent it to others, upload it or share it in many different ways. Therefore, if the primary user deletes a backup item which was already accessed by a delegate, the PrimeLife Backup Demonstrator can not guarantee that no copy of the backup item exists any longer. Therefore the access control policy created by the delegator can contain an obligation for the TTP saying that the TTP should inform the delegator, if the TTP grants access to $k_{\text{DelegateTTP}}$ of any delegate.

The Virtual File System (VFS) API utilised by the PrimeLife Backup Demonstrator operates with files atomically. It means that in order to delete a simple backup item from the backup, the entire backup file omitting that particular backup item must be re-uploaded. This leads to significant amount of communication overhead especially in those cases when the hosting backup file is of non-trivial size and/or if the copies of the deleted backup item are distributed across several backups.

If a storage provider would provide a special communication interface enabling for only selected parts of the online file to be deleted or modified, it would be possible to perform deletion of selected backup item(s) without the need to re-upload the entire backup file.

Breaking the single backup file into multiple files could also solve the problem with an undesired overhead caused by the deletion of a backup task. In an ideal case, each backup item should be stored in a separate file encrypted with a separate key. However, in that case a possible huge amount of keys needed to be handled could lead to different kinds of problems. Last but not least, the size of the file and the amount of files might also leak some information about the content stored in the backup item even if encryption is applied.

4.2.4 Deletion of Storage Provider

There are four options provided by the PrimeLife Backup Demonstrator for removing a storage provider from the PrimeLife Backup Demonstrator:

1. deactivation of the storage provider;
2. soft deletion of the storage provider;
3. hard deletion of the storage provider;
4. deletion of the storage provider with migration of the backup data.

Deactivation of a storage provider

By deactivating a storage provider, the backup data stored in the location of the deactivated storage provider remains untouched. The PrimeLife Backup Demonstrator disables any operations on the backup data performed by the user (e.g. the scheduled backups). On the other hand, deactivation of a storage provider does not influence the delegates involved. If the delegates fulfil the required conditions, they can further access the backup data.

Soft deletion of a storage provider

By performing *soft deletion* of a storage provider, all meta data corresponding to the deleted storage provider is deleted. However the backup data stored in the remote location of the storage provider remains untouched. Soft deletion of the storage provider follows the same principle as the soft deletion of backup (see section 4.2.1).

Hard deletion of a storage provider

By performing *hard deletion* of a storage provider, not only all meta data corresponding to the deleted storage provider is deleted, but the underlying backup data get erased as well. Hard deletion of a storage provider follows the same principle as the hard deletion of backup (see section 4.2.1).

Deletion of a storage provider with migration of the backup data

By performing a deletion of a storage provider with migration of the backup data, all meta data corresponding to the deleted storage provider is deleted. The underlying backup data stored in the remote storage location provided by the deleted storage provider is transferred to one or more locations provided by other storage providers.

The process of migration proceeds in one of the following ways regarding delegation of the migrated backup data:

1. Migration triggers revocation of the old delegations and new delegations are issued to the delegates involved.
2. Delegates involved are informed that the delegated backup data was migrated and their corresponding delegations are updated by the delegation update message.

3. Migration is performed such that no delegate involved notices that the data was migrated. This could be simply done with the help of the TTP which in this case would not only store $k_{\text{DelegateTTP}}$ but additionally the actual location of the backup data².

²If the primary user would decide to cancel his contract with the TTP and thus has to migrate the data stored at the TTP to another TTP, the delegation would again become unusable. Of course one could now introduce an additional redirection service meaning that the delegation does not contain the location of the TTP but just the location of that lookup service, which each delegate can ask for the actual location of the TTP. But nevertheless – as soon as the anchor of the chain of redirects changes the delegations have to be reissued.

Chapter 5

Installation & Usage Guide

5.1 Installation

One of the requirements of the PrimeLife Backup Demonstrator was to make it accessible from different platforms. Therefore the developers decided to provide a ready-to-use installation using a virtual machine. It uses VirtualBox¹, a virtualization machine monitor that is available for a wide range of operating systems. The product is Open Source software and, thus, freely available.

In order to ease the installation process, we created a screencast, which gives guidance through the steps of installation. It can be found here: <https://prime.inf.tu-dresden.de/backupdemo/>². Also, the individual steps of installation are described in the following.


Installing VirtualBox. The PrimeLife Backup Demonstrator was realized using version 4.0.6 of VirtualBox, which has to be installed on the local machine.

- ▶ Open your browser and go to address <http://www.virtualbox.org/>.
- ▶ Download the installation file for your operating system from the Downloads section.
- ▶ Start the installation of the retrieved file, follow the instructions and wait until the installation will have successfully finished.

Installing the prototype. In order to install the prototype, the file PrimeLifeBackup.ova has to be downloaded as well and registered with VirtualBox. Please have in mind that the size of this file consists of about 3.6 GByte. So, you have to ensure enough space on your hard drive. Also, you should be aware of possible long download time.

- ▶ Download the prototype file from <https://prime.inf.tu-dresden.de/backupdemo/>.
- ▶ Start the VirtualBox application if it is not already running.

¹<http://www.virtualbox.org/>

²Accessing the content behind the mentioned URL requires a login and password  

- ▶ In VirtualBox, go to the “File” menu and select the item “Import Appliance ...”.
- ▶ Select the prototype file `PrimeLifeBackup.ova` and follow the instructions.

Starting the virtual machine and running the prototype. Linux Ubuntu 10.04 LTS has been used as the hosting environment for the PrimeLife Backup Demonstrator running in the virtual machine.

- ▶ Start the VirtualBox application if it is not already running.
- ▶ Double-click on the entry `PrimeLife Backup` to start the virtual machine.
- ▶ Wait until the virtual machine and the PrimeLife Backup Demonstrator have both started. The completion status of the demonstrator is indicated by the message `PrimeLife Backup started successfully` that appears in the system tray (in the upper right corner).
- ▶ Right-click on the tray icon of the demonstrator indicated by the PrimeLife cat and select `Show Management Console` to get displayed the demonstrator’s user interface in the browser. In case that the PrimeLife Backup Demonstrator tray icon is not shown, you can restart the front-end by clicking on the `Backup Your Life` menu entry of the `Applications|Internet Ubuntu` menu.

Updates. During the boot of the virtual machine, it will automatically look for updates of the PrimeLife Backup Demonstrator. If an update is available, it will be fetched in the background. After the update process has finished, the new version of the demonstrator can be used. Additionally, a manual update procedure can be triggered by executing the `update_bul` command in a shell.

Runtime dependencies. The original idea risen in the initial phase of the work on the PrimeLife Backup Demonstrator was to deliver a self-contained virtual machine which contains all the necessary components to demonstrate various use cases. Unfortunately, this goal could not be reached fully yet. The reason is, that one of the PrimeLife components used (namely the Obligation Enforcement Engine) was developed using the Microsoft .Net framework which is only available on the Windows operating system. Although a freely available reimplementation of the .Net framework exists (called Mono), it turns out that the current state of Mono is not mature enough for running the Obligation Enforcement Engine. Therefore, the Obligation Enforcement Engine together with the PrimeLife Policy Engine are provided on an external server. This in turn means, that the real machine, which runs the virtual machine needs to be connected to the Internet in order to let the PrimeLife Backup Demonstrator communicate with the Obligation Enforcement Engine and the Policy Engine. Both components are needed in case a delegation is issued or exercised.

Internet connectivity is also needed, if one wants to use the DropBox storage provider.

5.2 Usage

Instead of including the usage guide within this document we decided to provide it on the one hand as online help within the demonstrator and on the other hand in the Web at <https://prime.inf.tu-dresden.de/backupdemo/>. This would ensure that the usage

guide stays in sync with the ongoing and further development of the PrimeLife Backup Demonstrator – even after the end of the PrimeLife project.

Chapter 6

Summary & Outlook

In this deliverable we described the technical background of the PrimeLife Backup Demonstrator. This demonstrator can be seen as a prototype-like application able to demonstrate important aspects of lifelong privacy, which covers three main characteristics: dynamics in the environment of an individual, stages of life as well as areas of life (cf. [WP111b, WP111a]).

For the purpose of demonstration, we focused especially on privacy-preserving delegation of access rights to backup data. Clearly the prototype is not meant to be a full-functional software application able to manage the every-day backup of an ordinary user. Besides the limited resources available for the development of the demonstrator, a simple obstacle was the non-availability of a full-fledged identity management system fulfilling all the requirements arising from a “real-world” privacy-preserving backup solution respecting lifelong aspects.

Although we cannot *prove* the validity of our solution with respect to the lifelong aspects as it is simply impossible to predict future evolvments in technology, it was possible to propose solutions for requirements coming from different disciplines such as law, sociology, technology etc.

Moreover, several enhancements are already foreseen. One of them is the integration of functionality provided by existing social networks and collaborative workspaces in order

1. to provide a more flexible collaborative environment and
2. to use characteristics provided by social networks for delegation purposes.

Further, a differentiation between private and business purposes needs also to be considered: what are the legal foundations, how to bring those together with the personal interests of the users, and how to reflect this in technology.

Glossary

A

API

Application Programming Interface.

B

backup item

A copy of a primary item stored in the backup. A backup item reflects the data of a primary item in the time when the backup item is created.

D

delegate

Is an entity, which receives access rights on the primary user's backup.

delegator

Is an entity, which has the privilege to delegate rights to delegates concerning a particular backup. In most applications of this demonstrator, the primary user acts as the delegator.

E

eCV

electronic Curriculum Vitae.

H

HTML

Hyper Text Markup Language.

HTTP

Hyper Text Transfer Protocol.

J

JAXB

Java Architecture for XML Binding.

JSON

JavaScript Object Notation.

O**OAuth**

Open Authorisation, a standard for authorisation used in the Internet.

P**primary item**

An original item for which one or more backup items are created during the back up process. In a general sense, a primary item can be referred to as any determinate set of data, which has one or more copies called backup items dedicated for backup purposes. A primary item can be a file but it can also be a more specific type of data as for instance an e-mail, a contact, or even settings of the TV.

primary user

Data subject who owns/holds primary items.

R**REST**

Representational State Transfer.

S**SSL**

Secure Socket Layer.

T**TTP**

Trusted Third Party.

V**VFS**

Virtual File System.

W**WebDAV**

Web-based Distributed Authoring and Versioning.

X**XML**

EXtensible Markup Language.

Bibliography

- [WP111a] PrimeLife WP1.3. Final Report on Analysis, Scenarios, Requirements and Concepts, Evaluation Results of “Privacy in Life”. In Katrin Borcea-Pfitzmann, Bibi van den Berg, and Jan Camenisch, editors, *PrimeLife Deliverable D1.3.3*. PrimeLife, <http://www.primelife.eu/results/documents>, May 2011.
- [WP111b] PrimeLife WP1.3. Scenario, Analysis, and Design of Privacy Throughout Life Demonstrator. In Katrin Borcea-Pfitzmann, editor, *PrimeLife Deliverable D1.3.1*. PrimeLife, <http://www.primelife.eu/results/documents>, February 2011.
- [WP511] PrimeLife WP5.3. Final Release of the Policy Engine. In Tobias Pulls, editor, *PrimeLife Deliverable D5.3.3*. PrimeLife, <http://www.primelife.eu/results/documents>, May 2011.