Project Number	**AC095**
Project Title	**ASPeCT:** Advanced Security for Personal Communications Technologies
Deliverable Type	**Major**
Deliverable Number	**D09**
Deliverable Title	**Trusted Third Parties: First Implementation**
Nature of the Deliverable	**Prototype**
Security Class	**Public** (but see Page 3: *Important Note on Limitations and Restrictions*)
Date due	February 1997 (Y2M12)
Reference	**AC095/RHUL/W23/DS/P/09**

Document Title	**Trusted Third Parties: First Implementation - Overview**
Document reference	ASPeCT/DOC/RHUL/WP23/078/
Issue status	Issue: 1.5
Contributing WPs	WP2.3;
Date of Completion	18-March-1997
Editor	Keith Howker

Abstract	This document describes the overall structure and content of the documentation provided as part of D09.
Keywords	ACTS, ASPeCT, TTP, security, integrity, key, escrow, recovery

Table of Contents

Executive Summary

Background and Requirements

Workpackage WP2.3 concerns the research and development of appropriate measures, based on Trusted Third Parties (TTPs), to meet the needs for security and protection in future (UMTS) mobile tele-communications services and environments.

ASPeCT deliverable D09 provides a demonstration of a working TTP service, with generation and distribution of key material to support protected communications between two users in different domains, each using a TTP server in the home domain.

Objectives

The goal for D09 is to implement a specification in ASPeCT Deliverable 7 [D07] to give a simple demonstration of the types of services that could be provided and verify the operation of the key management mechanisms.

Purpose of this Document

This document provides an overview of ASPeCT Deliverable D09.

Deliverable D09 provides a demonstrable version of the first implementation of ASPeCT Trusted Third Party service; in addition, the following documentation is available to authorized users:

- Annexe A Technical specification;
- Annexe B Operational description and guidelines for a demonstration.
- Annexe C Index and archive of the implementation and its detailed documentation (listings etc.);

Outline of Content

Identifies the above components of the deliverable, and their location and relationship.

Provides a brief technical description and outline of the demonstration.

Important Note on Limitations and Restrictions

The ***public*** nature of this deliverable is restricted to this document and the demonstration itself.

No general rights to the programmes and the libraries which constitute the demonstrator/prototype are given or implied.

Certain components may be

- proprietary,
- subject to non-disclosure agreements,
- claimed and acknowledged as background material,
- subject to governmental controls on export or re-export.

Specific enquiries or requests for clarification may be addressed, in the first instance, to the editor.

1 Document Control

1.1 Document History

This is the first issue.

1.2 Changes Forecast

No update of this document is intended unless errors are found subsequent to delivery to DG XIII; however the documentation of enhanced versions of the demonstrator will relate to it

1.3 Change Control

All ASPeCT documentation is under change control once given *issued* status; however, this document is for information about the deliverable only, and is unlikely to change subsequent to delivery.

1.4 Document Cross References

[D07]	ASPeCT Deliverable D07 - Security Services: First Specification ref. AC095/RHUL/W23/DS/I/07/1 (internal reference R050_1)
[TA]	Technical Annexe to the ASPeCT Contract (A095 - Amendment Number 1, 13.06.96)
[RBI]	Register of Background Information - (Internal project document ASP/PMN/024)

2 Introduction

Deliverable D09 consists of a demonstrable prototype and the documentation components shown below.

This document provides an overview of the deliverable and a directory to the main components.

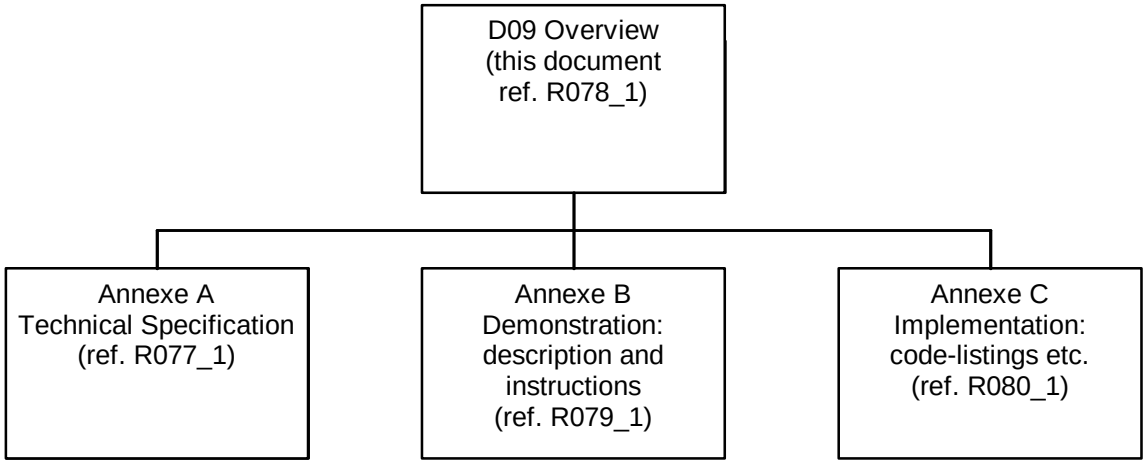

Figure 2.1 - Structure of Deliverable D09 documentation

3 Components of D09

3.1 Overview *(this document)*

3.2 Demonstration

Details of the demonstrable system are specified in Annexe C, below.
(This is a snapshot which is declared as the deliverable; development and enhancement will continue beyond this.)

3.3 Annexe A : Technical Specification

The technical specification for the demonstrator is ASPeCT document R077, Issue 1.

It does not form part of the public deliverable.

The document provides the implementation specification of the technical and operational aspects of the first demonstration of the use and operation of TTPs.

Outline of Content

The technical descriptions and specifications of the TTP itself are given in Section 4

Cryptographic aspects are given in Section 5.

Supporting software, and communications technology are covered in Section 6.

An outline of the Demonstration is given in Section 7.

3.4 Annexe B: Demonstration documentation

Instructions for running a demonstration are provided in ASPeCT document R079, Issue 1.

It does not form part of the public deliverable.

This contains

- specification of software and hardware requirements;
- guidelines for setting up the demonstration;
- description and scenario of the demonstration;
- operating instructions;
- expected results.

3.5 Annexe C: Implementation details

The implementation details are collected together as ASPeCT document R080 Issue 1.

It does not form part of the public deliverable.

This specifies a tape archive of the demonstrator at the time of delivery to DG XIII.

This defines the specific details of the components used in the demonstrator at the date of delivery:

- source files;
- libraries;
- tools;
- linked files;
- executable files;
- hardware configurations.

Note that subsequent versions of the demonstrator (general or specific to some requirement) may include appropriate improvements, corrections or simplifications.

3.5.1 Demonstrator code

The demonstrator consists of the components given in Section 3.5.2, below.

3.5.2 Source material

Component	Provider/Source	Status
TTP service and protocols	P05 - RHUL	developed within ASPeCT
Graphical User Interface (GUI)	P04 - PFN	developed within ASPeCT
Tracer	P-06 - SAG	developed within ASPeCT
Finite State Machine	P02 - ATEA	developed within ASPeCT
Communications	P02 - ATEA	proprietary/background
ASPeCT Certificate	P07 - KUL	developed within ASPeCT
ACRYL Crypto Library	P06 - SAG	proprietary/background
Compilers & Tools	WATCOM C/C++, V10.6	proprietary

4 TTP technical description

4.1 TTP Architecture

The entities involved in this implementation of TTP architecture are TTP servers and two types of clients, namely mobile users and VASPs. The administrative roles specified in [D07] are not explicitly provided in this version.

The logical and physical connections between the above entities in the demonstration are shown in Figure 4.1

All three physical configurations shown may be set up, but the single PC (3) version is not recommended. (Other physical configurations - 3 PCs, or different distributions of roles in 2 PCs, say - are also technically feasible.)

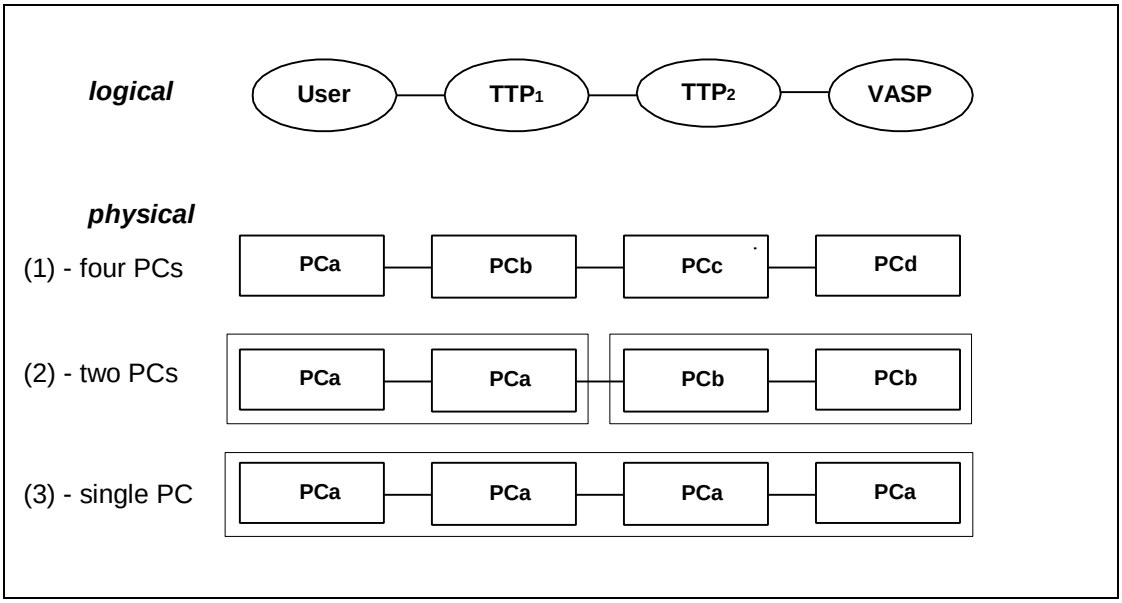

Figure 4.1 - Logical and physical connections between the entities involved in the demonstration

4.2 TTP Structure

The TTP uses the general software structure described in Section 6, below

Note: This first TTP demonstrator uses no external applications or services, but consists only of software provided by the project. The demonstration "application" comprises the execution of a protocol establishing an escrowed key supporting end-to-end protected communication.

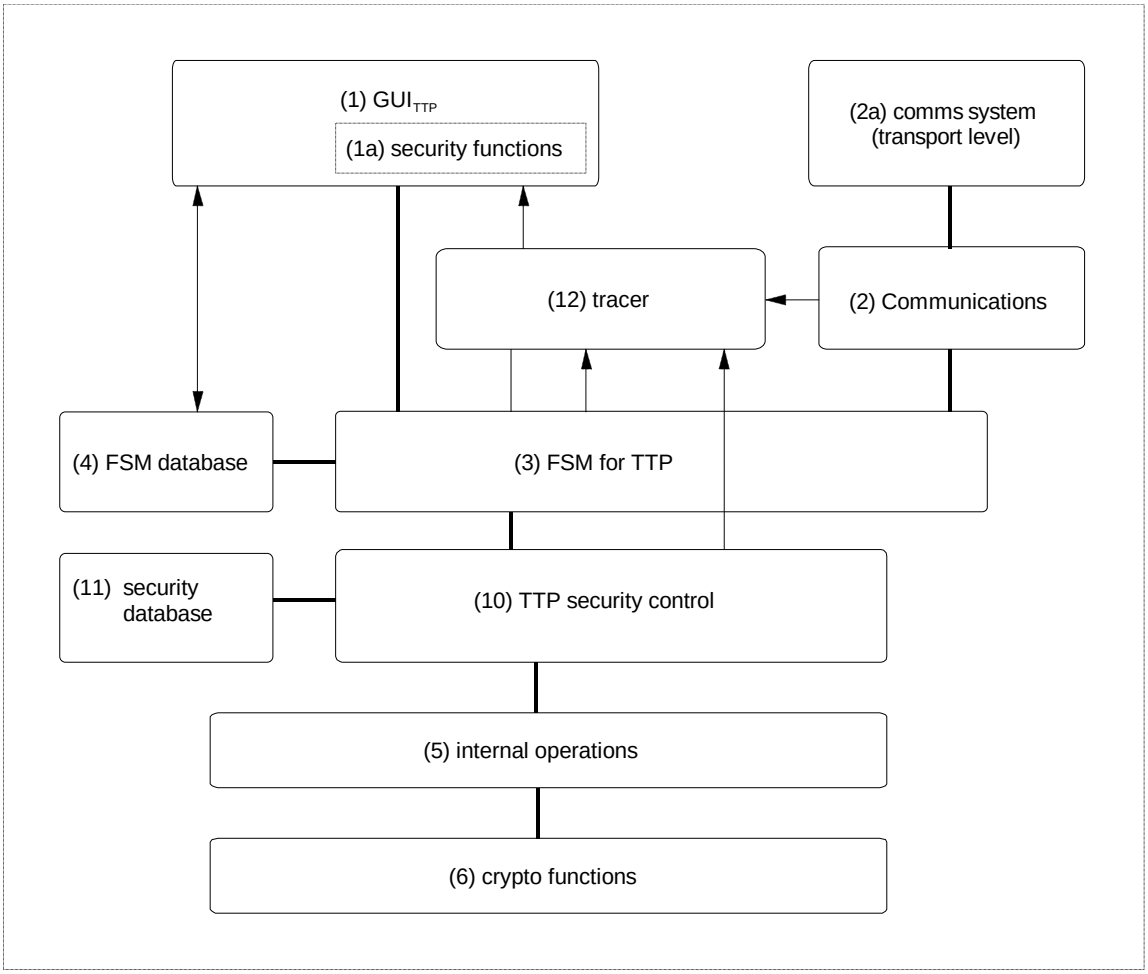

Figure 4.2 - TTP S/W Architecture

4.3 Protocol for the initial TTP demonstration

Within the ASPeCT project, we are concerned with the role of TTPs in supporting future mobile telecommunications security services using public key cryptography, and more specifically, the provision of end to end security services and secure billing services. One well established role for TTPs in supporting public key cryptographic techniques is in the generation of public key certificates. Another important role for TTPs in future mobile networks will be in supporting the particularly sensitive issue of end to end confidentiality with key escrow. In parallel with the use of the TTPs to provide public key certificates, we will be implementing the key escrow functionality within the ASPeCT TTP infrastructure.

In the first TTP demonstration, we are concerned with the generation of public key certificates and the implementation of key escrow functionality.

The following protocol, which is based on JMW proposal, will be used in the initial TTP demonstration. This protocol makes use of the Diffie-Hellman algorithm for key exchange. In order to simplify our demonstration, we implement the protocol to provide only one-way communication (e.g. for store-and-forward). The adaptation of the scheme for two-way communication is very straightforward.

More specifically the protocol is in the context of a pair of clients, where one client wishes to send the other a confidential message and needs to be provided with a session key to protect it. Suppose that these two clients have associated two TTPs respectively, where these two TTPs are distinct. Note that, since this protocol is intended to provide warranted access to communications between users of the TTP service by authorized

recovery of key material, we assume that each TTP is located within the jurisdiction of some intercepting authority, and that each TTP operates subject to the regulations of that authority.

Start situations for the demonstration are as follows.

1. Four participants, namely two clients (one as a sender, say A, and the other as a receiver, say B) and two TTPs (each for one client, say TA and TB), are involved.
2. The two TTPs agree upon a key generating function *f* that takes as input the identifier of the receiver (in this case, *B*) and a secret key (shared by the TTPs). The TTPs have agreed between them values *g* and *p*. These values must have the usual properties required for operation of the Diffie-Hellman key exchange mechanism, namely that *g* must be a primitive element modulo *p*, where *p* is a large prime and (*p-1)* can be divided by another large prime *q*. These values have been passed to the two clients.
3. Each TTP has an asymmetric signature verification key pair. The private signature key is known only to itself, and an authenticated copy of the public verification key can be accessed by its own client and the other TTP.
4. Each client and its TTP have access to a protected channel between them, which provides origin authentication, data integrity and confidentiality.

Figure 4.3 shows the message exchanges of the protocol among the four participants. The protocol includes four parts, each of which can optionally be run separately.

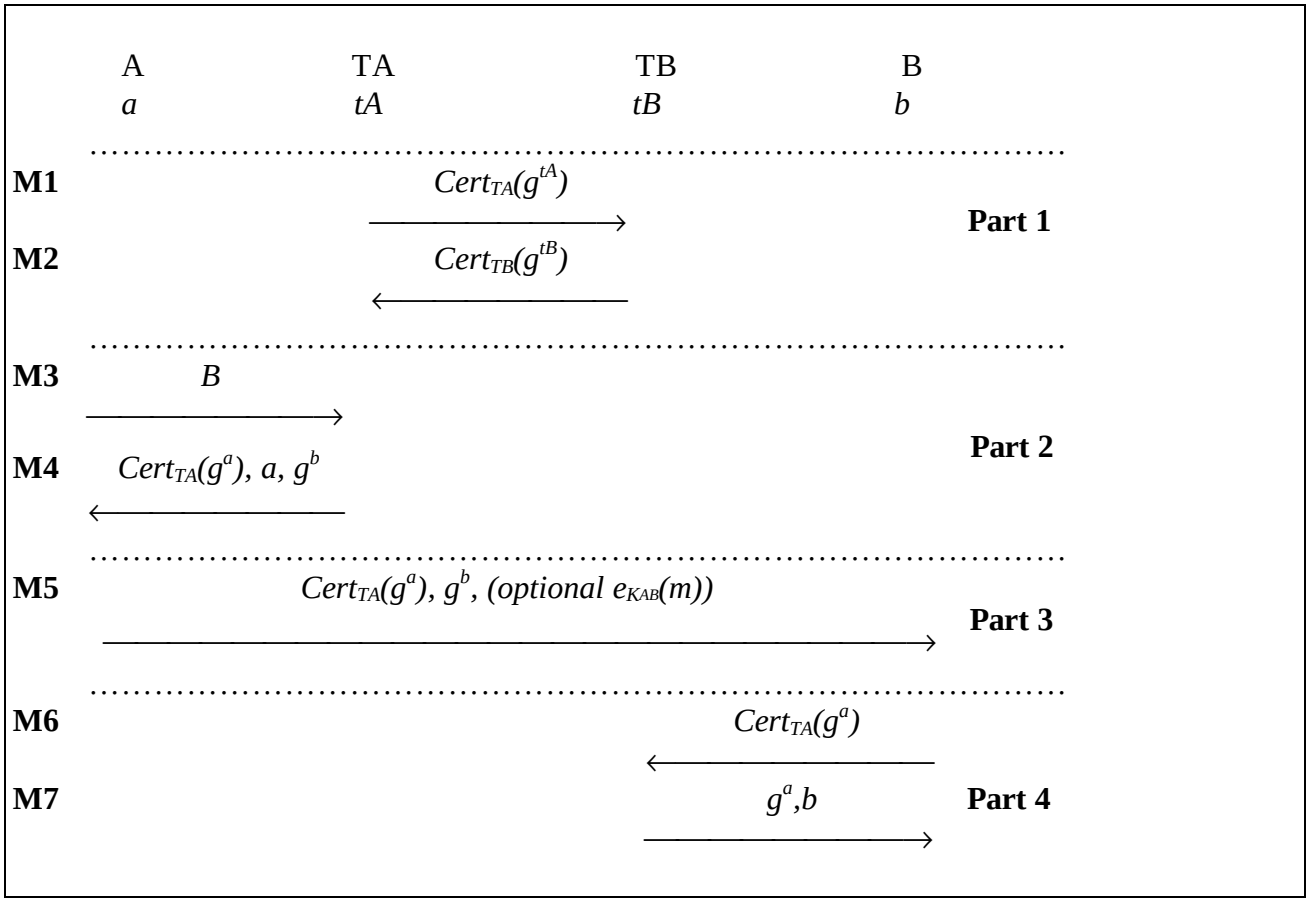

Figure 4.3 - End-to-end encryption key establishment

The following is the description of the protocol.

Part 1: share a secret between two TTPs.

1. Each TTP, TA or TB, generates a private and public key-agreement key pair (tA, g^{tA}) or (tB, g^{tB}).

2. Each TTP sends its public key-agreement key signed using its private signature key to the other TTP in M1: $Cert_{TA}(g^{tA})$ or in M2: $Cert_{TB}(g^{tB})$.
3. Each TTP verifies the received public key-agreement key using the other's authenticated public signature verification key.
4. Each TTP computes a shared secret, $K_{TATB} = g^{tAtB}$, using its own private key-agreement key and the other's public key-agreement key in Diffie-Hellman key establishment algorithm.

Part 2: Certificate generation in A's domain

1. A sends TA a request in M3 including B's name.
2. TA generates a random number a as A's private send key and a corresponding public send key, g^a.
3. TA makes a certificate for A's public send key, $Cert_{TA}(g^a)$.
4. TA computes B's private receive key, $b = f(K_{TATB}, B)$, and the corresponding public receive key, g^b.
5. TA sends A's public send key certificate, A's private send key and B's public receive key to A in M4.
6. A computes a shared key, $K_{AB} = g^{ba}$, using his own private send key and B's public receive key.

Part 3: message transmission from A to B

A sends the following information to B in M5:

- his public send key certificate issued by TA,
- B's public receive key, and
- an optional message encrypted by the shared key, K_{AB}.

Part 4: Certificate generation in B's domain

1. After receiving M5, B sends TB a request including A's public send key certificate issued by TA, and B's public receive key sent by A.
2. TB computes B's private receive key, $b = f(K_{TATB}, B)$, and verifies g^b.
3. TB verifies A's public send key certificate using the public signature verification key of TA.
4. TB sends A's public send key and B's private receive key to B. Note that if B has already got his current private receive key, this key does not need to be sent in M7.
5. B computes the shared key, $K_{AB} = g^{ab}$, using his own private receive key and A's public send key.

After running the protocol successfully, a shared key K_{AB} will be established between the sender and receiver, and this key will be escrowed by both TTPs.

Further considerations:

- Assume that there is a protected channel between each client and its own TTP. How to implement such a channel will be considered in the second TTP demonstration.
- Time-stamping may be needed in each key, which will be considered in the second TTP demonstration.
- All options will be chosen during the program design.

Note that there are other variants of the protocols which are also compatible with the JMW scheme, and that possible modifications of the proposed protocol for the second demonstrator and trial in a mobile environment are for further study.

4.4 TTP services and interfaces

4.4.1 TTP , Functions Operations and Components

Two levels of TTP component are provided to allow for a set of *functions*, which may correspond with a user/application visible API, together with a set of more primitive *internal operations* which invoke cryptographic actions. The reason for maintaining the two level, although somewhat redundant with the reduced facilities required for the first demonstration, is that a single *function* may require a number of *internal operations*

Table 4.1 - TTP Functions

	Function	**used in first demonstrator**
1.	user identification	N
2.	credit checking	N
3.	key generation	Y
4.	key distribution (on-line)	N
5.	key archiving	N
6.	key escrow functionality	Y
7.	privilege attribute generation	N
8.	privilege attribute distribution	N
9.	privilege attribute archiving	N
10.	certification functions	Y
11.	directory functions	N
12.	revocation functions	N
13.	alert management functions	N
14.	claimant and verifier functionality	N
15.	evidence generation	N
16.	evidence recording	N
17.	evidence verification	N
18.	time-stamping functions	N
19.	audit functions	N
20.	delivery authority	N
21.	fraud detection and management.	N

4.4.2 TTP Internal operations

The following types of internal operation will be required to support the complete list of functions identified in Section 4.4.1. Note that this list is not intended to imply that all TTPs should provide all these operations: rather the selection of functions to be supported by a TTP will determine what types of operation it must be capable of performing. The operations implemented in the first demonstrator are marked below.

Table 4.2 - TTP Internal Operations

	Internal operation	**used in first demonstrator**
1.	cryptographic key generation	Y
2.	cryptographic computation	Y
3.	cryptographic key storage	Y
4.	user information storage	Y
5.	event information storage	Y
6.	access control information generation	N
7.	certificate generation	Y
8.	event analysis	N
9.	time-stamp generation.	N

4.5 TTP operation and administration

Only minimal administrative facilities are provided for the first demonstrator: set-up is mainly off-line.

Trace points will be provided to form a history trail of operations and protocol-steps, and to allow for display by the *tracer*.

5 Cryptographic data structures - Certificates

5.1 Types of certificate

Two types of certificates are provided for ASPeCT, depending on use of different signature mechanisms. The left-most byte of the certificate string is the certificate type identifier.

The first type makes use of RSA-signature based on ISO/IEC 9796-2 (Draft International Standard). Its certificate type identifier is 00 (hex). Its signed certificate information sequence includes a signed recoverable string and a non-recoverable part.

The second type makes use of AMV-signature based on ISO/IEC 14888-3 . Its certificate type identifier is 01 (hex). Its signed certificate information sequence is the certificate information sequence itself together with an appendix, which is the signature of the sequence.

Within ASPeCT, these two types of certificates make use of one single certificate information sequence format as described below.

5.2 Certificate information sequence

In the description of the certificate information sequence, *issuer* denotes the entity issuing the certificate authoritatively; *subject* denotes the entity holding the private key, for which the corresponding public key is being certified.

The certificate information sequence includes a basic certificate information and a set of extended attributes providing other optional information about both the subject and the issuer.

5.3 Signature

Within ASPeCT we have two models of signature used to sign a certificate information sequence. The related certificate type identifiers are assigned as 00 and 01 respectively.

- RSA-signature based on ISO/IEC 9796-2 (Draft International Standard)
- AMV-signature based on ISO/IEC 14888-3

The first demonstrator uses only the first of these.

5.4 Certificate format for the first TTP demonstration

Table 5.1 - A certificate information sequence format for the first TTP demonstration

Field	Contents	Description	Length
1.	Map field	For public key – 10111000 For secret key – 00111000	1 byte binary
2.	Version	10 (hex)	1 byte binary
3.	Serial Number	Unique number of the certificate, assigned by the issuer	12 bytes ASCII
4.	Public key identifier	No	
5.	Issuer Identifier	The hashed issuer identifier	16 bytes binary
1	Validity	Including two dates: 1 the date before the certificate is not valid, 2 the date after the certificate is no longer valid.	 6 byte ASCII 6 byte ASCII
2	Subject Identifier	The hashed subject identifier	16 bytes binary
3	Subject key usage	The usage of the subject key being certified includes: 1 digital signature, 2 data encryption, 3 key agreement, 4 key certificate signature, 5 CRL signature.	1 byte binary 0 1 2 3 4
4	Cross certificate attributes	No	
5	Certificate path attributes	No	

6	Subject public key information	An algorithm type identifier plus a public key value for the subject.	
		Subfield 1: algorithm type identifier 0 = RSA 1 = elliptic curve 2 = Diffie-Hellman other unspecified	1 byte binary
		If algorithm type identifier = 0	
		Subfield 2: modulus length of key in bits	2 bytes binary
		Subfield 3: exponent length of key in bits	2 bytes binary
		Subfield 4: key value: first modulus, then exponent of key	sum of values of subfields 2 and 3 bytes binary
		If algorithm type identifier = 1	
		Subfield 2: length of x-coordinate of key	2 bytes binary
		Subfield 3: key value: first x-coordinate, then y-coordinate of key	2 times (value of subfield 2) bytes binary
		Subfield 4: parameter set identifier	1 byte binary
		If algorithm type identifier = 2	
		Subfield 2: length of key value in bits	2 bytes binary
		Subfield 3: key value: g^x mod p	value of subfield 2 bytes binary
		Subfield 4: parameter set identifier	1 byte binary

The first TTP demonstration makes use of only the RSA-signature based certificate.

To make matters less complicated, in the first TTP demonstration, we use a basic certificate information as the certificate information sequence, and ignore most options. Table 5.1 shows this certificate information sequence format.

If this basic certificate is used, the length of the non-recoverable part will be 87 bytes;
the length of the certificate will be 152 bytes if a modulus of 512 bits is used, and 216 bytes if a modulus of 1024 bits is used.

6 Software Structure

6.1 Introduction

The generic ASPeCT software architecture consists of several functional blocks as laid out in Figure 6.1.

There are two types of demonstrators foreseen in our software structure :

- Type 1 :

  The first type of demonstrator is one where existing applications like Netscape are used. The existing application will be extended with an ASPeCT specific security layer. How is this done ? The existing applications make use of the WinSock interface to communicate with others. It calls the WinSock Dynamic Link Library. We can patch this call to the WinSock DLL so that another DLL will be called. The patch can be written to call the ASPeCT security functions DLL instead of the WinSock DLL. But the original WinSock interface must still be called also. This can be achieved by calling the WinSock DLL from within the ASPECT DLL.

  This approach will by used by some applications in Workpackages 2.3 and 2.5.

- Type 2 :

The second type is where the whole application is written by ASPeCT. For these applications a software structure is designed, which allows reusability of software in the domains of communication between applications. This type of applications will be written for Workpackages 2.1 and 2.4 (which have a common demonstrator) and for the applications in Workpackages 2.3 and 2.5 that don't use existing applications like Netscape.

Some terminology :

- **Application** : is one instance of a program controlled separately by the operating system. In C or C++ each application corresponds with one execution of a "main" function. An example of an application is Netscape Navigator.
- **Entity** : is a defined set of functions at a single location, needed to simulate the different elements in the UMTS environment, such as Trusted Third Party, Interception Authority, Service Provider, ... Every activated entity within the application is represented by an opened window. Each entity acts like a finite state machine. It receives an event (a communication message over TCP/IP, serial link or message queue or a user message from the GUI) and responses to that event by taking some actions like calculation of an algorithm, preparing and sending new messages. The PC representing the Mobile Terminal and the intelligent card reader with smart card are seen as one entity.

During design of the software structure the following considerations are taken into account :

- One application may represent or run one or more entities. This is simply achieved by routing messages to procedures in the own application. It allows testing of the software without extra software or hardware needed for TCP/IP communication.
- the type of communication used between two entities should be invisible for the software that represents the entities, whether TCP/IP is used, or whether the entities run in one application.
- TCP/IP connections are activated at start of the demonstrations and remain active during the tests. When a finite state machine wants to simulate a connection-set-up flow, then it has to define its own connect messages.
  The TCP/IP connections are used as signalling channels.

The software structure identifies all the functional blocks that can be used by all workpackages WP2.1, WP2.3, WP2.4 and WP2.5. Arrows between different blocks indicate an interface.

6.2 Software implementation

With the exception of the ACRYL crypto library, all software is constructed by the project using WATCOM C++ Version 10.6 tool kit, this provides:

- compiler
- development aids and diagnostics
- linker
- loader
- window creation and management.

For compatibility and availability of support within partners' organizations, Windows 3.11 is used as the common platform for development and demonstration.

6.3 Functional blocks in the software structure

This section describes each functional block in the software structure and the interfaces to other functional blocks.

6.3.1 Block 1 : GUI

This block contains different modules (a module is represented by a file), each module contains functions that may be used for one ore more of the other blocks in the software structure. Block 1 is not yet split up in different modules in this document.

The Graphical User Interface provides the following facilities:

- The main application is defined in the GUI
- Administration of the communication database (See Section 6.3.2.5) :
    - creating a new entity with its address.
    - configuring the TCP/IP network.
    - activating the TCP/IP connections.

  This administration occurs via procedures defined in software block 2.1.
- Allows a user to start execution of a protocol via block 2.5.
- Allows the user to set trace-points and levels of trace-points before and during the tests or demonstrations. See Section 6.3.12 for a description of trace-points.
- Offers procedures to the tracer (block 4) to open windows and display messages in them.

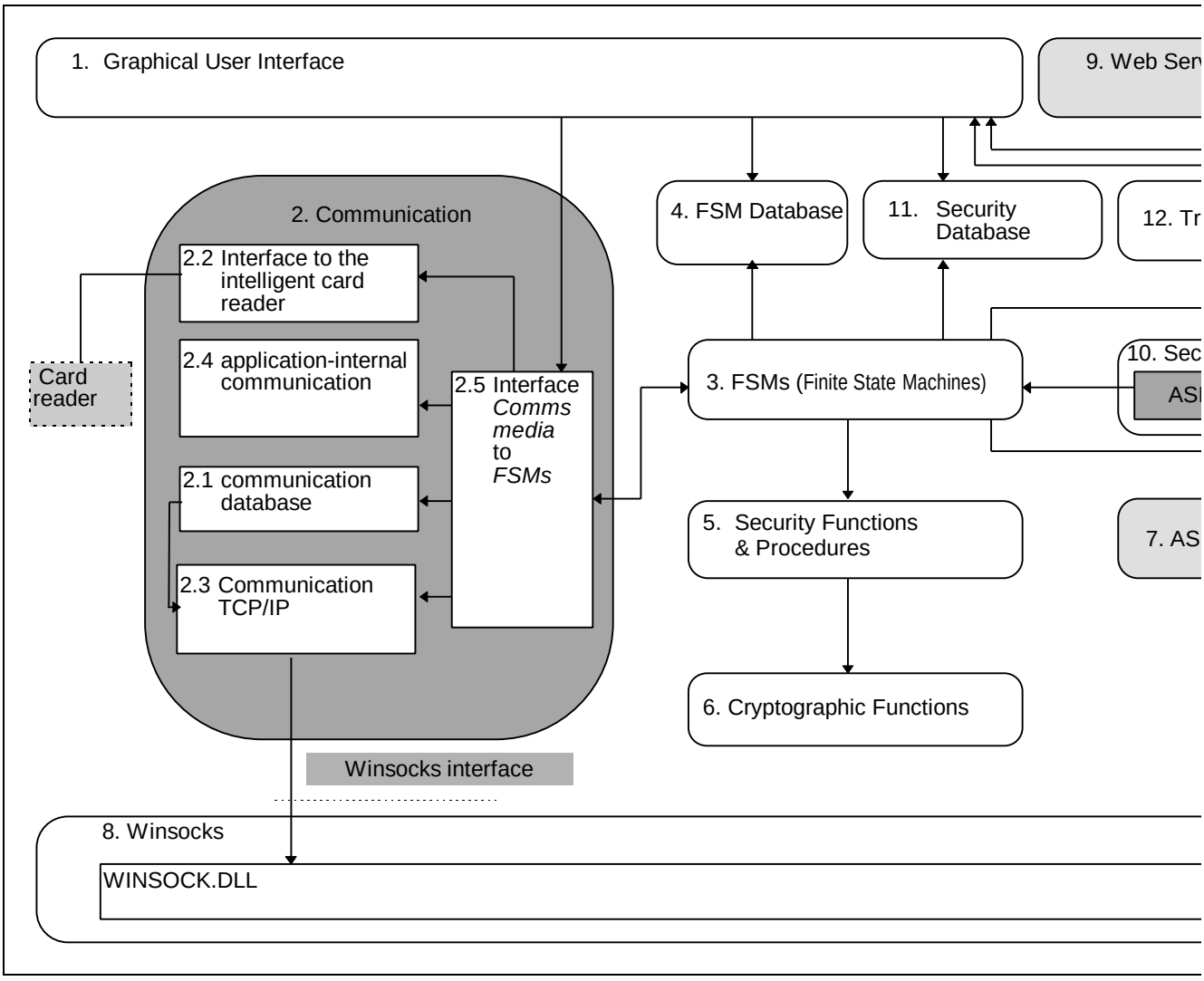

Figure 6.1- Software Structure

6.3.2 Block 2 : Communication

A more detailed design of block 2 is given in Section 6.4.

6.3.2.1 Block 2.1 : Communication database

This is a database containing all the data necessary for the communication.

This database holds a table with one entry for every entity in the configuration. Each entry in the table has following attributes :

- Name of the entity : e.g. TTP1 (Trusted Third Party), TTP2, IA1 (Interception Authority), IA2, SP1 (Service Provider), SP2, NO1 (Network Operator), NO2, VASP (Value Added Service Provider), MT (Mobile Terminal), IC1 (Intelligent Card reader), IC2.

  There is a predefined list of possible entity-names. This list must contain all the possible entities for all the workpackages.

- Number of the application that runs the entity. One or more applications may run on one or more PC's. Each application has a number. There is one server application and there may be 9 client applications.

- A reference to an instance of the finite state machine class.
  When the user creates an entity then he has to indicate which Finite State Machine defined in block 3 corresponds to that new entity. The corresponding class will be instantiated and a reference to this instantiation is held in the communication database. When the application is not the server application in the network then a TCP/IP message will be sent to the server. This message will indicate the application number and new entity name. The server will update its database and send a similar message to all the other clients in the network, allowing them to update their database. One application may represent more than one entity. The communication database also makes use of procedures defined in block 2.3 to send messages to other TCP/IP nodes, as just mentioned.

6.3.2.2 Block 2.2 : Interface to the intelligent card reader

This block is not used in the first demonstrator.

6.3.2.3 Block 2.3 : Communications TCP/IP

This block establishes the TCP/IP connections at initialisation phase and receives from and sends to the other PC's via TCP/IP.

6.3.2.4 Block 2.4 : Application-internal communication

This block is used when two entities run on the same application. There will be as good as no functionality in this block. The application sends a message to an own defined buffer.

6.3.2.5 Block 2.5 : Interface between the communication media and finite state machines

This block

- initialises the database on request from the GUI
- activates the TCP/IP connections on request from the GUI
- offers a common interface between the finite state machines created on the PC and the different communication media (serial, TCP/IP, internal).
- It accesses the communication database to get the entity (=finite state machine) attributes during protocol execution.
- It receives input from the GUI in order to activate one of the finite state machines.
  This block also defines the abstract base class for a finite state machine which will be used by modules in block 3. In block 3 one derivation of the base class will be defined for each type of finite state machine. The

base class will define a few basic functions e.g. ProcessIncomingMessage, which will handle incoming messages destined for the finite state machine.

6.3.2.6 Block 2.6 : Interface to Diskette

This block is not used in the first demonstrator.

6.3.3 Block 3 : Finite State Machines

This software block defines all the finite state machine classes.

One instantiation of a finite state machine class represents one entity in the configuration. One application may represent one or more entities.

As explained in Sections 6.3.1 and 6.3.2.1 one of the available finite state machines is chosen for each entity running on the PC. This choice is made via the GUI.

At the same time, additional parameters may be set, like choice of algorithm, length of keys. Requirements on the GUI should be identified from the different workpackages. It is up to each workpackage whether or not to define administrable parameters. Databases necessary to hold these parameters are defined in block 4.

Each finite state machine interprets the messages or events received from Block 2.5 and executes the corresponding protocol. It may send one or more messages via Block 2.5 and it goes to a new state.

The software interface between 2.5 and each defined finite state machine is common as mentioned in section 6.3.2.5.

The list of finite state machines is :

- User
- Trusted Third Party

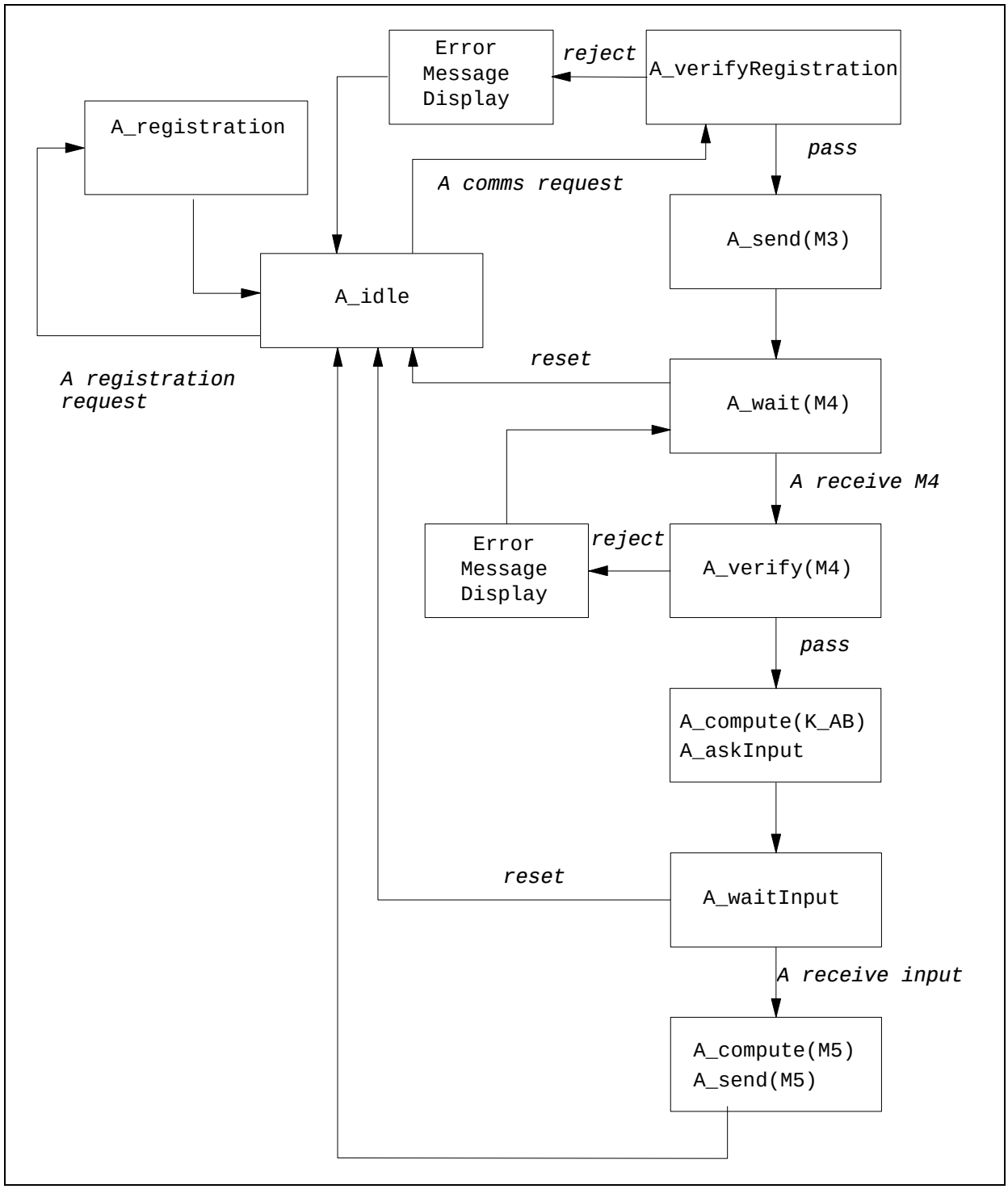

Figure 6.2: State-Event Diagram: User

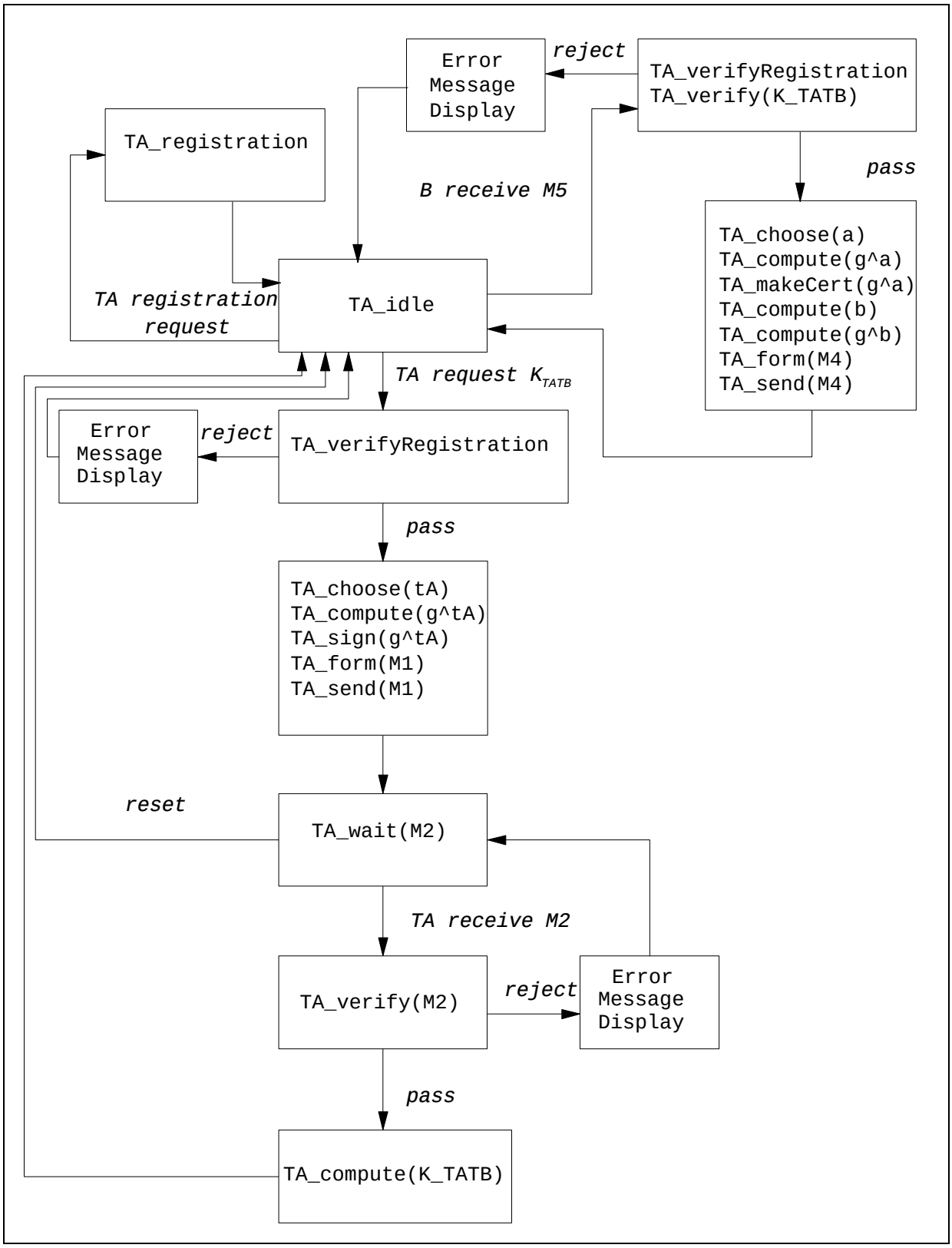

Figure 6.3: State-Event Diagram: TTP

6.3.4 Block 4 : Finite State Machine Databases

This block defines data needed by the finite state machines. e.g. length of keys, value of keys, ...

The data is initialised via the GUI during start-up of the application.

6.3.5 Block 5 : Security Functions/Procedures

This block builds an ASPeCT specific interface towards basic cryptographic functions.

6.3.6 Block 6 : Cryptographic Functions

The basic cryptographic functions will be in this block.

6.3.7 Block 7 : ASN.1

This block contains C++ classes for ASN.1 definitions and procedures for encoding, decoding and displaying.

6.3.8 Block 8 : Winsocks

This block contains the standard WinSock libraries.

6.3.9 Block 9 : Existing applications

These are existing applications like the Netscape software that use standard WinSock interface. This application has its own graphical user interface, separate from the one represented by block 1.

6.3.10 Block 10 : Security Layer

This block is only used for applications of type 1 where existing applications are used. The security layer interfaces with block 9 (existing applications) via the standard WinSock library. This interface will be patched to call the ASPECT.DLL security layer. The security layer also interface with the TCP/IP protocol stack.

Block 10 also interfaces with :

- block 1 : the GUI
- block 3 : the finite state machine, to be able to interface with the intelligent card reader, the TCP/IP protocol stack, the tracer, the security functions and procedures and the security database.

6.3.11 Block 11 : Security Database

The security database holds parameters specific to the security layer and is administered via the GUI.

6.3.12 Block 12 : Tracer

The tracer can display messages on a screen or write them in a file. The tracer is configured via the GUI during initialisation of the demonstration. The finite state machines instruct the tracer to display or save messages.

6.4 Communications Subsystem

The communication subsystem will provide TCP/IP connectivity and an interface for communication with the intelligent card reader, for the ASPeCT demonstrators.

The TCP/IP protocol suite allows computers of all sizes, from many different vendors, running totally different operating systems, to communicate with each other. The de facto standard for TCP/IP implementations is the one from the Computer Systems Research Group at the University of California at Berkeley. It is called “sockets”.

The basic building block for communication is the socket. A socket is an endpoint of communication to which a name may be bound. Each socket in use has a type and an associated process. Sockets are typed according to the communication properties visible to a user. Applications are presumed to communicate only between sockets of the same type.

Two types of sockets are a stream socket and a datagram socket.

A *stream socket* provides for the bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries.

A *datagram socket* supports bi-directional flow of data which is not promised to be sequenced, reliable, or unduplicated. That is, a process receiving messages on a datagram socket may find messages duplicated, and, possibly, in an order different from the order in which it was sent. Record boundaries are preserved.

The use of stream sockets is recommended for communications within the ASPeCT package.

Since we are going to use Windows as a platform for the demonstrators, we can use the Windows Sockets interface, which is based on the "socket" paradigm, and supports both stream (TCP) and datagram (UDP) sockets.

6.5 GUI

A full description of the Graphical User Interface (GUI) is given in Annexe B of this document set.

6.6 Tracer

A general character-oriented tracer is used to display details of the protocol messages, e.g.:

```
TTP A:    Waiting for connect requests
```

giving the current state of the demonstrator, or

```
VASP:     Received parameter g^u:
(269450332567843258901347302562214467825,28107647577622821956341002075533875831 1)
```

giving the real values of the secure billing protocol parameters, enabling the verification of the protocol being run. The tracer is also an important debugging toll during the development of demonstrator SW.

The displayed tracing messages are also saved to disk for further examination.

A full description of the tracer is given in Part B.

6.7 Cryptographic support

The applications programming interface is provided by a proprietary cryptographic library ACRYL (Advanced Cryptographic Library). This library has been registered as background information to the project [RBI].

The ACRYL library consists in an administration layer containing the abstract programming interfaces and a library of algorithms containing the concrete algorithms. The administration layer consists in the following functional groups:

- time functions (e. g. for time stamps);
- block ciphers and stream ciphers;
- random number generation;
- computation of hash values;
- generation and management of keys;
- signatures.

The individual groups may contain various algorithms.
It should be emphasized that the user does not call the concrete algorithm, rather the concrete algorithm is selected in an abstract call by an identifier conformant to ISO.

Details of ACRYL interfaces and calls are given in more detail in Part B.

6.7.1.1 ACRYL functions for ASPeCT

The following functions are provided:

1. random number generator based on DES-OFB / triple DES-OFB
2. hash functions RIPEMD - 128 and RIPEMD - 160
3. RSA signature generation and verification
4. encryption with DES-CBC
5. Exponentiation in GF(p)
6. Key generation for RSA and for DES

6.7.2 Algorithms

6.7.2.1 Algorithms used in first TTP demonstration

The following six algorithms are used in the first TTP demonstration. A number of open questions will be answered after analysing the application of this working set; currently, only a subset of the full algorithm library is available..

6.7.2.1.1 Random number generation

There are three random numbers needed in the above protocol: *a*, *tA* and *tB*. They all are used as private key agreement keys, which should have the size between 512 and 1024 bits. The generation of the TTP keys *tA* and *tB* probably does not need to be demonstrated, so that we do not specify an algorithm here. The choice of random number generator (RNG) is for further study.

It is suggested by KUL that an alternative to using DES-OFB as a RNG is to use the hash function already used later in the protocol. To generate the user key *a*, the TTP could use a secret IV and then hash A's identity string together with the date and a serial number, say. This then requires further evaluation of the hash function.

6.7.2.1.2 Key generation

The Diffie-Hellman keys will be generated by the above RNG. The values *g* and *p*, which have the usual properties required for operation of the Diffie-Hellman key exchange mechanism, can be fixed in the first demonstration.

6.7.2.1.3 Key agreement

Diffie-Hellman key agreement algorithm will be used. An elliptic curve method can be used to implement Diffie-Hellman key exchange

6.7.2.1.4 Hash function

It is proposed to use RIPEMD-160 in certificates, if an RSA based certificate type is used. In the other cases RIPEMD-128 will be used.

6.7.2.1.5 Signature and verification

Two choices are RSA-signature based on ISO/IEC 9796-2 and AMV-signature based on ISO/IEC 14888.

6.7.2.1.6 Symmetric encryption and decryption (optional)

Actual end-to-end encipherment of user-data messages is not part of the demonstrator. Optionally, encryption may be used to protect the channel between client and TTP; the choice of encryption algorithm is for further study.

7 TTP Demonstrator

The number of PCs to be used in the first TTP demonstration may be chosen during setup. There are three main options:

- using only one PC for the four participants,
- using two PCs, one for A and TA, and the other for B and TB,
- using four PCs, one for each participant.

The main objective of the first WP2.3 demonstration is to present the procedure necessary for communication, supporting end-to-end encryption, through the use of TTP services. It comprises two clients, User A and User B, located in different domains and two TTPs, TTP A and TTP B, one for each domain. User A communicates securely with User B with the intervention of their respective TTPs which collaboratively perform the role of providing the users with key management services.

The GUI provides drop-down menus and prompt boxes which guide the user through the demonstration. Full details are given in Annexe B.

Figure 4.1 illustrates the messages exchanged during the demonstration.

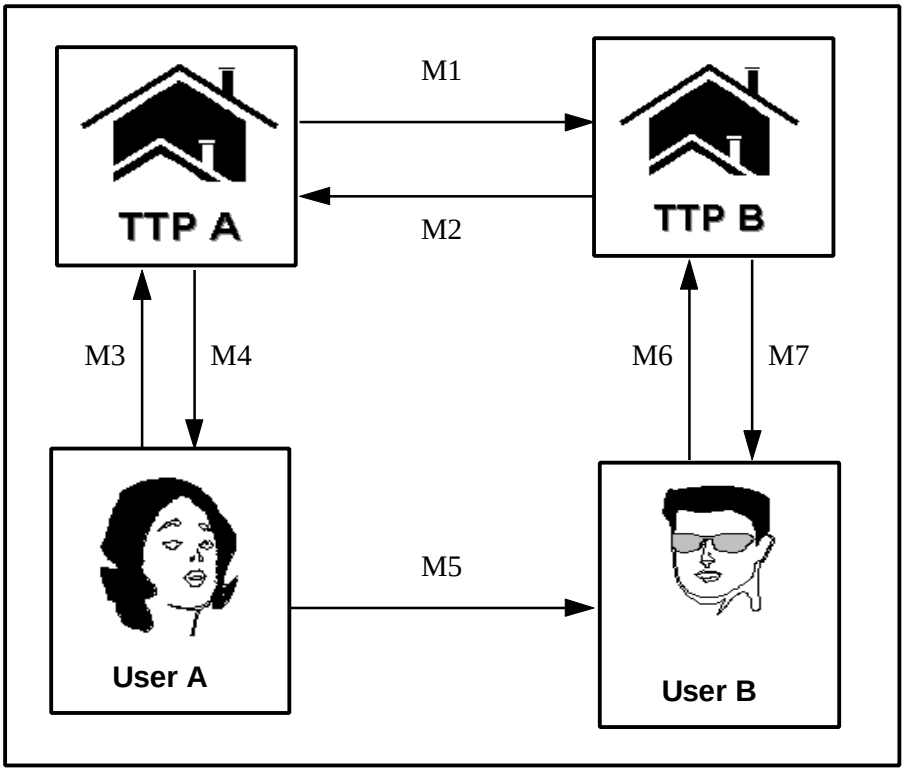

Figure 7.1 - Message exchanges (see also Figure 4.3)

The procedure is initiated and conducted by the demonstration users who can act as the demonstrated entities. Every entity is entitled to a set of specific actions that the user may select from the appropriate Action Menu. For example, the demonstration user that acts as User A, will be prompted to write the message he wants to transmit to User B. The whole process is viewed through two options: the Tracer, that displays the actual message exchange, and the Monitor, that provides visual representation.

7.1 Summary of messages and actions

A1. send M3: B
A2. wait for M4: $Cert_{TA}(g^a),a,g^b$ – received
A3. verify $Cert_{TA}(g^a)$ – pass or reject
A4. compute K_{AB}
A5. (optional) input a message – m (which will be enter by PC's keypad and display in the window)
A6. (optional) encrypt m to get $e_{KAB}(m)$
A7. form M5: $Cert_{TA}(g^a),g^b$, *(optional* $e_{KAB}(m)$*)*
A8. send M5

Figure 7.2 - Sender A - sequence of events

TA1. choose tA
TA2. check tA
TA3. compute g^{tA}
TA4. sign g^{tA}
TA5. form M1: $Cert_{TA}(g^{tA})$
TA6. send M1
TA7. wait for M2: $Cert_{TB}(g^{tB})$ – received
TA8. verify $Cert_{TB}(g^{tB})$ – pass or reject
TA9. compute K_{TATB}
TA10. wait for M3: B – received
TA11. choose a
TA12. check a
TA13. compute g^a
TA14. sign g^a
TA15. form $Cert_{TA}(g^a)$
TA16. compute b
TA17. compute g^b
TA18. form M4:$Cert_{TA}(g^a)$, a, g^b
TA19. send M4
TA20. store a and b

Figure 7.3 - TTP TA - sequence of events

TB1. choose *tB*
TB2. check *tB*
TB3. compute g^{tB}
TB4. sign g^{tB}
TB5. form $Cert_{TB}(g^{tB})$
TB6. wait for M1: $Cert_{TA}(g^{tA})$ –received
TB7. verify $Cert_{TA}(g^{tA})$ – pass or reject
TB8. sending M2:$Cert_{TB}(g^{tB})$
TB9. compute K_{TATB}
TB10. wait for M6:$Cert_{TA}(g^{a})$, g^{b} – received
TB11. compute *b*
TB12. verify g^{b}
TB13. verify $Cert_{TA}(g^{a})$ – pass or reject
TB14. form M7: g^{a}, *b*
TB15. send M7
TB16. store *b* and g^{a}

Figure 7.4 - TTP TB - sequence of events

B1. wait for M5: $Cert_{TA}(g^{a})$, g^{b}, $e_{KAB}(m)$ – received
B2. send M6: $Cert_{TA}(g^{a})$, g^{b}
B3. wait for M7: $Cert_{TB}(g^{a})$, *b* – received
B4. compute K_{AB}
B5. (optional) decrypt $e_{KAB}(m)$ (*m* will be displayed in the window)

Figure 7.5 - Receiver B - sequence of events

In order to express if the protocol has be run successfully, optionally, we let A send a message to B. This message will be displayed in B's window.

7.2 The Configuration Part

The following terms are defined:

- **Application:** is one instance of a program controlled separately by the operating system.
- **Entity:** is a defined set of functions at a single location, such as TTPs and Users.

One application may represent or run one or more entities. One or more applications may run on one or more PCs. In case that more than one PCs are used, the communication between the applications is done via an Ethernet LAN, using TCP/IP protocol. Each application has a number. There is one server application and there may be up to 9 client applications. In the first demonstration, the user is able to configure the communication database. Through the Graphical User Interface he/she may determine the number of clients and define the server and the client IP addresses. The user may also create or destroy entities, define their

identities and assign them to the existing applications. General information about the TCP/IP related values and the state of all entities is provided and can be displayed at any instant. More detailed information about the configuration procedure is included in section 7.3.

7.3 The Action Part

The end-to-end confidential communication establishment is done via a message exchange process according to the protocol described in section 4.3, above. The message exchange of the protocol, initiated from the Action menu, includes four parts:

- The **initial TTP / TTP set-up** in which the two Trusted Third Parties exchange secret keys in a reliable and secure way to support end-to-end encryption for their respective clients.
- The **initial TTP / sender set-up** in which the sender requests and obtains a private key from the corresponding TTP.
- The **direct sender / receiver communication** in which the encrypted message transmission takes place.
- The **TTP / receiver set-up** in which the receiver requests and obtains the required secret and public key information in order to compute the sender / receiver shared key.

Through the GUI the user may conduct and observe the message exchange process either in a detailed or in a more general way.

7.4 Further information

A detailed user guide to the demonstration is to be found in Annexe C to this document. This will take the form of a brochure, mainly for use in demonstrations to be given at the IS&N conference.