Structuring and Visualising an IC-card Security Standard

Hugh Glaser[1], Pieter H. Hartel[1,2], Eduard K. de Jong Frz[3]

[1] Department of Electronics and Computer Science, University of Southampton, UK,
Email: hg,phh@ecs.soton.ac.uk.
[2] Faculty of Mathematics, Computer Science, Physics and Astronomy,
University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands,
Email: pieter@fwi.uva.nl.
[3] QC Technology, Zaandam, The Netherlands, Email: eduard@q2c.nl.

Abstract. The standard way of visualising protocols using pictures with boxes and arrows is insufficient to study the protocols in detail. The problem is that the structuring of the protocols relies on elements not explicit in the usual visual rendering. To solve the problem one should visualise not only the operations and the messages but also the state and the security. This paper presents a system which can be used to visualise a protocol, and is applied to some of the protocols in the load purse transaction of the CEN Inter-sector electronic purse draft standard as an example. The resulting conformant prototype provides abstract and concrete views on the system at all significant levels. The prototype supports animation of the standard, giving the protocol designer feedback on design decisions.

Keywords: protocol visualisation, protocol structure, standards, smart cards.

1 Introduction.

The security of IC-card systems depends as much on the cryptography as it does on the quality of the system implementation. Provably correct implementations of the relatively complex transaction protocols are as yet infeasible because of their high cost. The costs of provably correct implementations of complex software will come down [11], but presently one has to compromise. The only technique available for building secure systems is to structure the system carefully. Many small building blocks and lean interfaces make it difficult for errors to permeate the entire system. If furthermore the building blocks can be composed easily, a high level of confidence in the reliability of the system is possible.

When given a particular application area, it is possible to produce structures that give the required degree of modularity without unduly constraining the system performance. The work described here is directed at producing an appropriate structure to support the security architecture in an emerging standard. It is important to scrutinise standards for potential problems in the implementations. Much effort is put into drafting standards, and once accepted, the standard will persist for a long time.

Our efforts form part of a larger project to build prototypes for IC-card systems so as to study the properties of these systems. In previous work, we have taken a rather more formal approach to prototyping smartcard and network architectures using similar transaction protocols [7]. The focus of the present paper is on the structural aspects of the transactions, and more precisely on the clarity of structure rendering. This does not preclude the use of formal methods to make additional statements about the correctness of the system, but this is not our aim here.

The present paper emphasizes the structural aspects of transaction protocol implementation through the use of a visual, object oriented, data flow programming language Prograph [6]. Important visual cues inherent in the work with protocols are most naturally rendered by a visual programming language. A full implementation of a protocol makes it possible to run simulations of the protocol, and even more importantly to animate the protocol. The visual feedback to the protocol designer and prototype builder is vital to a full understanding of the design.

A structure has been developed for the transaction protocols in the European draft standard for an Inter-sector electronic purse [5]. This document will henceforth be referred to as 'the standard'. The standard contains several different transaction protocols, which fit the proposed structure. We believe that other transaction protocols will also be accommodated by the proposed structure.

The next section introduces our case study, the protocols from the standard. In Sect. 3 we discuss principles of structuring synchronous protocols. These are used to describe a particular protocol from the standard in Sect. 4. This is followed by a description of a prototype animation in Sect. 5. Related work and conclusions are presented in the last two sections.

2 Inter-Sector Electronic Purse.

The standard [5] describes a security architecture for electronic purse systems. In the electronic purse system three parties are defined. The purse provider (for example a bank) issues the electronic purse. The service provider (for example a shop) accepts electronic payment for services. The card holder charges the electronic purse with funds and uses the purse to pay for services. The system works in a similar way to travellers cheques. These are also prepaid by the user and accepted as payment by a service provider. The organisation issuing the travellers cheques guarantees the cheques to both the user and the service provider. The electronic purse is not a real purse in the sense that two purse holders cannot exchange funds.

The electronic purse system defines a number of protocols for the three parties to use when communicating for a specific purpose. The major protocols are:

- *Load* the purse with the electronic counter value of a certain amount of conventional currency. This transaction involves the card holder and the purse provider.

- *Purchase* a service using the electronic purse to effect payment. This transaction involves the card holder and the service provider.
- *Collect* the totals of the electronic transactions. This protocol credits the service provider with the conventional counter value of the electronic money (similar to a shop keeper paying travellers cheques into the bank account). This transaction involves the purse provider and the service provider.

A number of other protocols are defined for maintenance purposes, such as monitoring the status of the purse, conversion of currency etc.

In the standard, the transaction protocols are partly described using natural language, partly using a BASIC-like programming language. The standardisation committee has spent considerable effort in clarifying the important issues. However, the transactions are sufficiently complicated to make the descriptions verbose, difficult to interpret and error prone. The use of adequate structuring tools would have been useful to increase the clarity of the descriptions.

2.1 Structure in the Standard

The standard is structured as a four part document. Within each part various structuring tools are used:

- *Part 1* describes the basic business relationships of the purse provider, the service provider and the purse holder. The three parties conduct their business through a number of logical components that interact. The standard specifies the functionality and interconnections of these logical components. The standard distinguishes Secure Application Modules (SAM) from other modules that are not secure.
- *Part 2* describes the security architecture of the electronic purse system. The description consists of a global overview and a detailed description of each protocol. The global overview is cryptographically neutral. The detailed description of a particular protocol will be shown specialised towards a particular cryptographic system. For most protocols, both an RSA and a DES version exist.
- *Part 3* describes the data elements and interchanges.
- *Part 4* describes the devices.

In the structures listed above it is Part 2 that provides a handle on the structuring of the actual protocols; It gives global and a detailed view on the protocols. The remaining structuring tools offered by the standard are aimed either at the levels above the protocols, such as the flow of electronic value and payment, or at levels below the protocol. They are thus not relevant to our presentation, which deals with the protocols.

3 Structure in a Transaction Protocol.

The structure of a protocol from the standard is highlighted by separating the essential elements of the protocol from the details. This results in a two part description consisting of a one page overview of the main elements of the protocol (see Fig. 2 for an example) and a listing of the details running over several pages (for a fragment see Fig. 3, a digest of these figures is given in Sect. 4).

The structure of the one page overview follows the exchange of messages that arises when a particular transaction is made. This is typical for the description of cryptographic protocols, which mostly follow the synchronous pattern as shown in Fig. 1a. Here two parties 'Alice' and 'Bob' exchange a number of messages whilst each performs some actions A1 ... A6 and generate some responses R2 ... R5.

Alice		Bob
A1:		
	⟶	
		A2:
		R2:
	⟵	
A3:		
R3:		
	⟶	
		A4:
		R4:
	⟵	
A5:		
R5:		
	⟶	
		A6:

(a) Disregarding the state owned by the parties.

Alice		Bob
A1:		
	⟶	
		A2:
	↓ a_1	↓ b_1
		R2:
	⟵	
A3:		
	↓ a_2	↓ b_2
R3:		
	⟶	
		A4:
	↓ a_3	↓ b_3
		R4:
	⟵	
A5:		
	↓ a_4	↓ b_4
R5:		
	⟶	
		A6:

(b) Explicitly tracing the state owned by the parties.

Fig. 1. Two abstract renderings of a synchronous protocol where two parties exchange messages. The A_i represent actions of the parties. The R_i are the responses and the a_i and b_i represent the state owned by the two parties.

The protocol of Fig. 1a does not expose much detail but it does show that we are dealing with a synchronous protocol between two parties. To obtain more information about the protocol one would have to zoom in on the A_i and R_i. The standard does exactly this, and it does so at two levels of detail. The first level exposes some detail (the one page overview) and the second level exposes all detail. The standard is reasonably successful when it describes the essential elements of a protocol in the one page overview. The data elements that are manipulated by the partners in the transaction are exposed as well as the data elements that are transmitted in the messages. Some of the relevant operations on the data elements are also exposed, such as 'sign' and 'verify' (these operations apply to signatures).

The standard uses no other structuring methods. We have identified two additional structuring tools that can successfully and usefully be applied to a protocol description. These tools will be presented in the next two sections.

3.1 Ownership of Data Imposes Structure

The first additional structuring tool extends the idea that in a protocol data is owned by one of the two parties. Data that is not transferred in a message is not accessible to the other party. We will often call the data that is owned by a party the state of that party. Only data that is carried explicitly from one action to the next is accessible. Data that is not explicitly traced becomes inaccessible. Tracing the state in an abstract protocol is schematically rendered in Fig. 1b. For example, the arrow labelled a_1 carries the state of Alice from action A1 to action A3.

In a concrete protocol one might trace all data elements explicitly from its producer to its consumer, so that it is clear where data is produced, accessed, modified and destroyed, thus making it possible to reason about the (mis)use of data elements.

3.2 Action Details Impose Structure and Sub-structure

The second structuring tool that we wish to impose on a protocol description concerns the internal structuring of the actions taken by a party. The following three steps are essential within each action:

- *Operational aspects* describe what the protocol is for and how the operations it supports take place. For example, The electronic purse protocols serve to transfer funds. The structure should make it easy to find out when and how funds are transferred.
- *Security aspects* describe what security is used for and how it is achieved. Security in the electronic purse protocol serves to authenticate the parties and to guarantee non-repudiation. It must be possible to uniquely identify transactions and parties involved in transactions. Security is delivered by cryptographic means. The structure should identify the cryptographic data and operations.
- *Protocol aspects* describe which data are transmitted and when the data is transmitted. The remaining data are kept locally by the parties involved in a transaction.

The low level details of each of these aspects should be independent of the high level aspects. For example it should be possible to exchange one cryptographic system for another without affecting the functionality of the protocol.

The three steps within each action are sufficient to describe transaction protocol in the ideal world. Adding security aspects to the protocols requires each step to be subdivided into several sub-steps. We identify six sub-steps in each action:

- *Protocol aspects – capture vars* receives the message. The contents of the message are captured in the state.
- *Operational aspects – operational conditions* check the state for operational errors. After checking the operational conditions, the state is known to be consistent.
- *Security aspects – protocol security* generates and checks the data used for authentication and non-repudiation. This includes the provision of unique identifiers for the components as well as the transaction.
- *Security aspects – cryptographic security* performs the cryptographic functions, such as encryption, hashing and signing of relevant data.
- *Operational aspects – operations* modify the state to reflect the progress made in the current transaction.
- *Protocol aspects – form response* gathers the data necessary to generate the response to the incoming message.

The structure of an action is sufficiently general to capture the pattern of computation imposed by the transaction protocols of the standard.

3.3 Using the Data Flow Paradigm to Make Structure Concrete

Having identified two new ways of imposing structure on a transaction protocol, we will now show that the data flow paradigm [9, 14] is ideal to support this structure. There are two reasons for this. The first is that both data flow programs and protocols are naturally highly visual. The second reason is that the data flow paradigm requires data to be traced explicitly from its source to its destination.

At the lowest level, data flow is based on the manipulation of data by state-less operators. An operator can have several inputs and outputs. An operator fires as soon as all its inputs are available. To pass data from one operator to another requires an explicit connection between the operators. This makes it possible to trace explicitly the origin and destination of all data. The basic state-less operators have no memory; state needs to be manipulated explicitly as data. Operators can be connected to form a new, more powerful operator. These constructed operators may have state, which at the implementation level of the constructed operator would be clearly visible. When viewed as a separate entity, the state of a constructed operator would not be visible.

The protocol of Fig. 1b can be given a data flow interpretation. This requires interpreting the actions A_i as (constructed) operators. The arrows in the protocol form the data flow edges.

A data flow network is most naturally represented as a picture. This corresponds well to the usual rendering of protocols. The extra edges (cf. the difference between Fig. 1a and b) needed for a data flow diagram to thread the 'state' of each of the parties in the protocol gives the picture an extra visual cue. It shows clearly that the state is not threaded arbitrarily through the system, and that the state is not accessible to just anyone. Instead the state is confined to particular actions. The state must be confined, because it may contain secret information. Obvious secrets that must not leak are cryptographic keys. In many applications the operational data is also regarded as sensitive. In all cases it must be clear where the sensitive information resides, so that proper protective measures can be made.

The graphical nature of the data flow model gives leverage in the prototyping of a protocol for a security architecture. In an interactive, GUI based system such as Prograph, the visual aspect allows the designer to interact directly with the prototype. Animations of the system make it possible to trace the flow of data. The animation can be stopped to allow data to be inspected and altered. This gives the designer immediate and valuable feed back.

4 Load Transaction from the Draft Standard.

The standard describes a number of transaction protocols in the style of Fig. 2. It is sufficient for the present discussion to describe only the essential elements of one of the transactions: the load purse transaction. The details the transaction may be found in the standard.

Figure 2 shows three parties engaged in a the load purse transaction: The Inter-sector Electronic Purse (IEP), the Load Device Application (LDA) and the Purse Provider Secure Application Module (PPSAM). The LDA acts as an intermediary driving the transaction because it is the device operated by the purse holder. From the security point of view, the LDA is transparent. This implies that our abstract two party protocol is applicable to describing the protocol from the standard. An important aspect of the driving function of the LDA is that it always checks the completion codes of the IEP and the PPSAM. If the completion code indicates an error, the LDA informs the purse holder and aborts the entire transaction.

Steps A1–C2

The transaction starts when the purse holder inserts the IEP in the LDA device. Step A1 acquires the amount to be loaded $\mathrm{M}_{\mathrm{LDA}}$ and currency $\mathrm{CURR}_{\mathrm{LDA}}$. Together these two values are referred to as $\overline{\mathrm{M}}_{\mathrm{LDA}}$. Step C2 sends a message to the PPSAM containing the command to 'initialise PPSAM for Load'. The parameter of the command is the amount and currency to be loaded ($\overline{\mathrm{M}}_{\mathrm{LDA}}$).

Steps A2–R2

The PPSAM at step A2 first checks that it is able to deal with the requested currency $\mathrm{CURR}_{\mathrm{LDA}}$. Step A2 then checks that the balance of the PPSAM, $\mathrm{BAL}_{\mathrm{PPSAM}}$, is sufficient for the load transaction. Then the PPSAM generates a random number R which will serve to uniquely identify the transaction. At step R2 the PPSAM forms the response message to the LDA. The response message contains as parameters the identity of the PPSAM and the random number.

	IEP			LDA			PPSAM
			A1:	Acquire $\overline{M}_{LDA}$			
			C2:	Initialise PPSAM for Load parameters($\overline{M}_{LDA}$)	⟶		

						A2:	verify($CURR_{LDA}$)
							verify($BAL_{PPSAM} \geq M_{LDA}$)
							Generate random R

					⟵	R2:	response(PP_{PPSAM}, PPSAM, R)
		⟵	C3:	Initialise IEP for Load parameters(PP_{PPSAM}, PPSAM, R, $\overline{M}_{LDA}$)			

A3:	verify($CURR_{LDA}$)
	verify($BAL_{IEP} + M_{LDA} \leq BALmax_{IEP}$)
	NT_{IEP}:=$NT_{IEP} + 1$
	S_1:=sign(PP_{IEP}, IEP, $DEXP_{IEP}$, NT_{IEP}, $\overline{M}_{LDA}$, BAL_{IEP}, PPSAM, R)						

R3:	response($\overline{FD}_{IEP}$, NT_{IEP}, BAL_{IEP}, S_1)	⟶					
			C4:	Debit PPSAM parameters($\overline{FD}_{IEP}$, NT_{IEP}, BAL_{IEP}, S_1)	⟶		
						A4:	verify(has got/will get conventional payment) verify($\overline{FD}_{IEP}$) verify(S_1) S_2:=sign(PP_{PPSAM}, IEP, NT_{IEP}, $\overline{M}_{LDA}$) BAL_{PPSAM}:= $BAL_{PPSAM} - M_{LDA}$
					⟵	R4:	response($\overline{IK}_{PPSAM}$, S_2)
		⟵	C5:	Credit IEP parameters($\overline{IK}_{PPSAM}$, S_2)			
A5:	verify(S_2) S_3:=sign(PP_{IEP}, PPSAM, R, CC_{IEP}) BAL_{IEP}:=$BAL_{IEP} + M_{LDA}$						
R5:	response(S_3, CC_{IEP})	⟶					
			C6:	PPSAM Load acknowledgement parameters(S_3, CC_{IEP})	⟶		

						A6:	verify(S_3)
							verify(CC_{IEP})

					⟵	R6:	response(CC_{PPSAM})

Fig. 2. Load transaction without an LSAM from the standard.

The identity of the PPSAM consists of two elements: the identity of the device, PPSAM, and the identity of the purse provider who owns the device $\mathrm{PP}_{\mathrm{PPSAM}}$.

Step C3

The LDA sends the 'initialise IEP for load' command to the IEP, with the amount and currency to be loaded. The LDA also sends the identification information of the PPSAM to the IEP.

Steps A3–R3

The IEP first checks that it can deal with the requested currency. It then checks whether its balance $\mathrm{BAL}_{\mathrm{IEP}}$ would not exceed a predetermined maximum $\mathrm{BALmax}_{\mathrm{IEP}}$. The IEP increments its transaction count $\mathrm{NT}_{\mathrm{IEP}}$ to generate a unique number of its own. The first authentication related step is the creation of the signature S_1. It contains a number of relevant data elements including the expiration date of the IEP, $\mathrm{DEXP}_{\mathrm{IEP}}$, and the identity of the IEP, the identity of the PPSAM and the unique numbers generated by the two parties. The response message generated at R3 packages all relevant information for the LDA. The notation $\overline{\mathrm{FD}}_{\mathrm{IEP}}$ is an abbreviation of a list of various data elements including identities, expiration date and keys for the security system.

Step C3

The LDA continues the transaction if the IEP is able to deal with the currency and amount to be loaded. It sends the 'debit PPSAM' command with the relevant parameters to the PPSAM.

Steps A4–R4

The PPSAM checks that the counter value of the money to be loaded has been supplied by conventional means. This could be done by, for example, requiring the purse holder to insert some bank notes into the LDA device. The data contained in $\overline{\mathrm{FD}}_{\mathrm{IEP}}$ will be checked. This involves testing operational conditions, such as whether the PPSAM and the IEP are indeed able to interact with one another, whether the IEP has perhaps been cancelled etc. The second verification step authenticates the IEP to the PPSAM via the signature S_1. The signature S_2 is then computed, so that later in step A5 the IEP is able to authenticate the PPSAM. The PPSAM has now been satisfied that all the operational conditions have been checked. Furthermore it knows that it is communicating with a genuine IEP, so the balance of the PPSAM can be decremented. This is the first 'real operation'. The response $\overline{\mathrm{IK}}_{\mathrm{PPSAM}}$ as generated at step R4 contains a list of relevant data for the IEP.

Step C5

The LDA sends the 'credit IEP' command along with relevant parameters to the IEP.

Steps A5–R5

The IEP authenticates the PPSAM. The IEP then computes its second signature, S_3, which is necessary for the PPSAM at step A6 to make sure that the load has indeed taken place. If one were to remove the IEP, for example, whilst it is carrying out step A5, the PPSAM needs to know about this so that it can undo the debit operation on its balance. The second 'real operation' of the transaction is to increment the balance of the IEP, just before the response at step R5.

Step C6
The LDA sends the 'PPSAM load acknowledgement' command. The completion code $\mathrm{CC}_{\mathrm{IEP}}$ is passed to the PPSAM so that it can check whether the IEP has been successful in incrementing its balance.
Steps A6–R6
The final step checks the completion of the IEP. Step R6 sends a final response to the LDA, which can be used to produce a receipt. This has not been made explicit in the given transaction protocol, as it is not interesting from the point of view of the security.

4.1 Analysis of the Transaction Structure

The transaction shown in Fig. 2 contains all of the aspects identified by the structuring tools of Sect. 3:

- Operational aspects:
  - Checking operational conditions at A2, A3 and A4.
  - Performing operations at A4 and A5.
- Security aspects:
  - Generation of signatures at A3, A4 and A5.
  - Verification of signatures at A4, A5 and A6.
- Protocol aspects:
  - Driving the transaction at C2 ... C6.
  - Generation of command messages at C2 ... C6 and the generation of result messages at R2 ... R6.
  - Decoding of command messages at A2 ... A6 and the decoding of result messages at C3, C4, C5 and C6.

The presentation of Fig. 2 focuses on the protocol aspects, as the messages are clearly identified. The other elements of the structure are not identified. We will first study some of the details of one of the actions in the protocol. Thereafter we present a description of the load transaction from the standard with all its structure clearly identified.

4.2 Details of Debit PPSAM

The analysis of the transaction applies to the level of detail as shown in Fig. 2. This ignores a number of important elements, such as which cryptographic system is used, and whether or not operations are logged. Such elements must play a role in a comprehensive analysis of the protocol. To illustrate these elements we will zoom in on the most interesting step A4–R4, which debits the PPSAM. This step is the only step which defines all of the operational, security and protocol aspects. Figure 3 shows step A4–R4 as it appears in the standard.

The first addition to the earlier, more abstract rendering of step A4–R4 is that we now see that a log is maintained of the actions taken by the PPSAM. The log contains relevant information necessary to trace all transactions.

	PPSAM	
A4:	verify(has got/will get conventional payment)	Operational conditions
	verify($\overline{\mathrm{FD}}_{\mathrm{IEP}}$)	.
	verify(S_1)	Protocol security
	S_2:=sign($\mathrm{PP}_{\mathrm{PPSAM}}$, IEP, $\mathrm{NT}_{\mathrm{IEP}}$, $\overline{\mathrm{M}}_{\mathrm{LDA}}$)	.
	$\mathrm{BAL}_{\mathrm{PPSAM}}$:= $\mathrm{BAL}_{\mathrm{PPSAM}} - \mathrm{M}_{\mathrm{LDA}}$	Operation
R4:	response($\overline{\mathrm{IK}}_{\mathrm{PPSAM}}$, S_2)	Protocol

Step	Action	Details	
A4:	Update the log	write(Load Log) [IEP, $\mathrm{NT}_{\mathrm{IEP}}$]	Operational
	Is the conventional payment OK?	[Outside the scope of this standard]	conditions
	Does the IEP belong to the PP?	IF $\mathrm{PP}_{\mathrm{IEP}} \notin \{\mathrm{PP}_{\mathrm{PPSAM}}\}$ THEN	.
		$\mathrm{CC}_{\mathrm{PPSAM}}$:=PP_MISMATCH	.
		write(Load Log) [$\mathrm{CC}_{\mathrm{PPSAM}}$]	.
		abort	.
	Is the IEP in the negative file?	IF ($\mathrm{PP}_{\mathrm{IEP}} \Vert \mathrm{IEP}) \in \mathbf{NF}$ THEN	.
		$\mathrm{CC}_{\mathrm{PPSAM}}$:=IEP_OPPOSED	.
		write(Load Log) [$\mathrm{CC}_{\mathrm{PPSAM}}$]	.
		abort	.
	Has the IEP expired?	IF $\mathbf{DATE}_{\mathrm{PPSAM}} > \mathrm{DEXP}_{\mathrm{IEP}}$ THEN	.
		$\mathrm{CC}_{\mathrm{PPSAM}}$:=IEP_EXPIRED	.
		write(Load Log) [$\mathrm{CC}_{\mathrm{PPSAM}}$]	.
		abort	.
	Is the algorithm and	IF $(\mathbf{ALG}_{\mathrm{IEP}}, \mathbf{VK}_{\mathrm{IEP}}) \notin$	Protocol
	key version supported?	$\{(\mathbf{ALG}_{\mathrm{PPSAM}}, \mathbf{VK}_{\mathrm{PPSAM}})\}$ THEN	security
		$\mathrm{CC}_{\mathrm{PPSAM}}$:=ALG_OR_KEY_MISMATCH	.
		write(Load Log) [$\mathrm{CC}_{\mathrm{PPSAM}}$]	.
		abort	.
	Select master key using key version	$\mathbf{KML}_{\mathrm{PPSAM}}$:=select(IEP, $\mathbf{VK}_{\mathrm{IEP}}$) [$\{\mathbf{KML}_{\mathrm{PPSAM}}\}$]	.
	Compute diversified key	$\mathbf{KD}_{\mathrm{PPSAM}}$:=encipher($\mathbf{KML}_{\mathrm{PPSAM}}$) [IEP]	.
	Compute session key	$\mathbf{KSES}_{\mathrm{PPSAM}}$:=encipher($\mathbf{KD}_{\mathrm{PPSAM}}$)	.
		[$\mathrm{DEXP}_{\mathrm{IEP}} \Vert \mathrm{NT}_{\mathrm{IEP}}$]	.
	Verify IEP signature	IF $\mathrm{S}_1 \neq$ sign($\mathbf{KSES}_{\mathrm{PPSAM}}$)	.
		[$\mathrm{M}_{\mathrm{LDA}} \Vert$ $\mathbf{CURR}_{\mathrm{IEP}} \Vert$ $\mathrm{BAL}_{\mathrm{IEP}} \Vert$ PPSAM$\Vert$ R] THEN	.
		$\mathrm{CC}_{\mathrm{PPSAM}}$:=LOAD_S1_FAILED	.
		write(Load Log) [$\mathrm{CC}_{\mathrm{PPSAM}}$]	.
		abort	.
	Compute debit signature	S_2:=sign($\mathbf{KSES}_{\mathrm{PPSAM}}$) [$\mathrm{M}_{\mathrm{LDA}} \Vert$ $\mathrm{CURR}_{\mathrm{LDA}}$]	.
	Decrease PPSAM balance	$\mathrm{BAL}_{\mathrm{PPSAM}}$:=$\mathrm{BAL}_{\mathrm{PPSAM}} - \mathrm{M}_{\mathrm{LDA}}$	Operations
R4:	Transmit PPSAM parameters and debit signature	response(S_2, $\mathrm{CC}_{\mathrm{PPSAM}}$)	Protocol

Fig. 3. Two levels of detail for the 'Debit PPSAM' step using DES from the Load transaction without an LSAM.

The first four IF statements show how the operational conditions are checked. The abort primitive causes the present step to be abandoned. It also generates a result message to the LDA containing only the completion code $\mathrm{CC}_{\mathrm{PPSAM}}$.

The identifiers in bold font are identifiers that have not yet been seen. They appear only at this level of detail. For example **NF** represents the negative file. This file contains the identities of the IEP's that are no longer valid, for example, because they have been reported lost.

The fifth IF statement shows how authentication works in the cryptographic protocol being used. A session key $\mathbf{KSES}_{\mathrm{PPSAM}}$ is computed, which is then used to encipher the data contained in the signature S_1. If the enciphered data fails to match the signature, the authentication fails.

The standard is in principle neutral with respect to the chosen cryptographic system. It provides diagrams such as Fig. 3 both for DES and for RSA.

This concludes the description of the standard load protocol. The main point of the description is that the structure of the protocol is easily obscured by a wealth of detail. We have indicated the structural elements by indicating to the right of the statements in Fig. 3 which parts of the structure they represent. In the following section we will look at a conformant prototype of the protocol structure from the standard, with the aim to clearly expose the structure.

5 A Prograph Prototype of the Protocol Structure.

It is desirable that the conformant prototype follows the standard as closely as possible, and so we expect to see the different aspects of the standard reflected in separate parts of the implementation. In this section we give the reader a flavour of the prototype, without becoming too involved in the details of the Prograph language.

We begin by looking at the first part of the standard, where the components are defined. The window that does this for the example we are using is shown in Fig. 4. There are five entities defined in the Prograph, which correspond to five of the entities from the standard. These are the components *PSAM*, *IEP* and *PPSAM* and the devices *LDA* and *PDA*. At this level, we have also chosen to expose the difference between the values that are part of the permanent data of components (the *State*), and the ephemeral or session data (the *Vars*), since the classification of data is important.

Prograph is an object-oriented language, and the lines between components indicate inheritance. The PSAMs, IEPs and PPSAMs are proper components since they inherit from *COMP State*. The LDAs and PDAs are not considered (by the standard) to have the same status, and therefore they do not inherit from the class *COMP*. When running the system, it is possible to open any of these icons and examine the attributes that instances of these entities have.

Most of the standard is concerned with the actual processing and flow of messages, and so it is important that the prototype accurately reflects the view of Fig. 2, which is the heart of the design. The Prograph method (procedure) which implements this is shown in Fig. 5. The dataflow in Prograph is down the

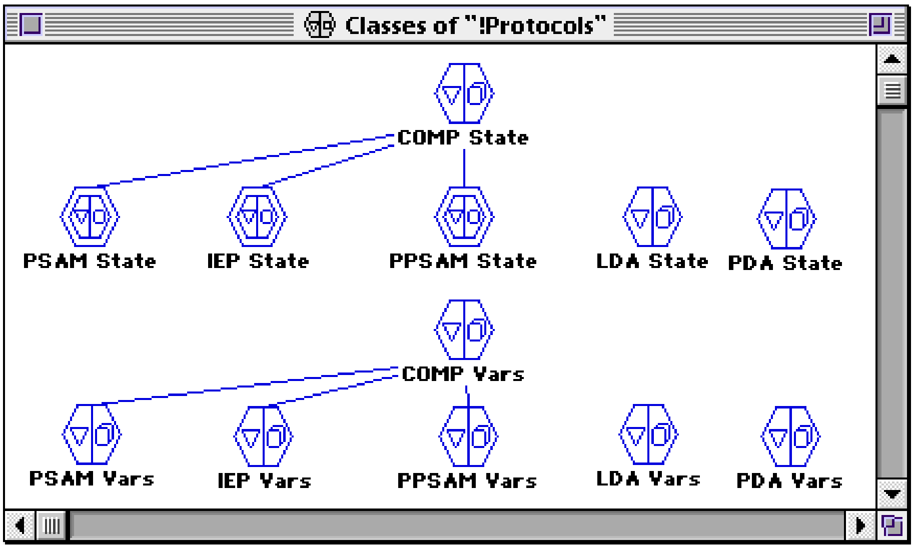

Fig. 4. The window defining the types of components of the standard.

page (data enters the top of operations and leaves from the bottom). We have placed the operations so that the whole method sends the right visual cues, with the IEP on the left, the LDA in the middle and the PPSAM on the right. This method is parameterised over the IEP and PPSAM, which come in as pairs of (State, Vars) at the input bar at the top, and return their new States via the output bar at the bottom.

Following the principle laid out in Sect. 3, we have introduced an extra level of representation here, for Fig. 2 includes more detail than Fig. 5. The latter is merely the Prograph equivalent of our general notion of a protocol as given in Fig. 1. We can now see that it is useful to be able to view the transaction without the details. Omitting the details also makes the animation of this method (when the system dynamically illustrates the operations being fired) clear.

Two aspects of the system have been made visible, which could not be seen in the standard. The first is concerned with aborting transactions. The notation in Prograph to indicate that an operation might fail, is to add an $\boxed{\text{X}}$ on the right. We can thus see that the LDA is in control of the transaction, since it is the component that can report failure (this is because it is the LDA that looks at the Completion Codes, and decides on the actions).

The second aspect that has been made visible in our rendering of the protocols, but not in the standard is the explicit threading of state from one step of the protocol to the next. The vertical lines connecting the steps to A3, from A3 to A5 and from A5 carry the permanent *State* and the ephemeral *Vars* of the IEP. Similarly for the LDA in the middle and the PPSAM on the right.

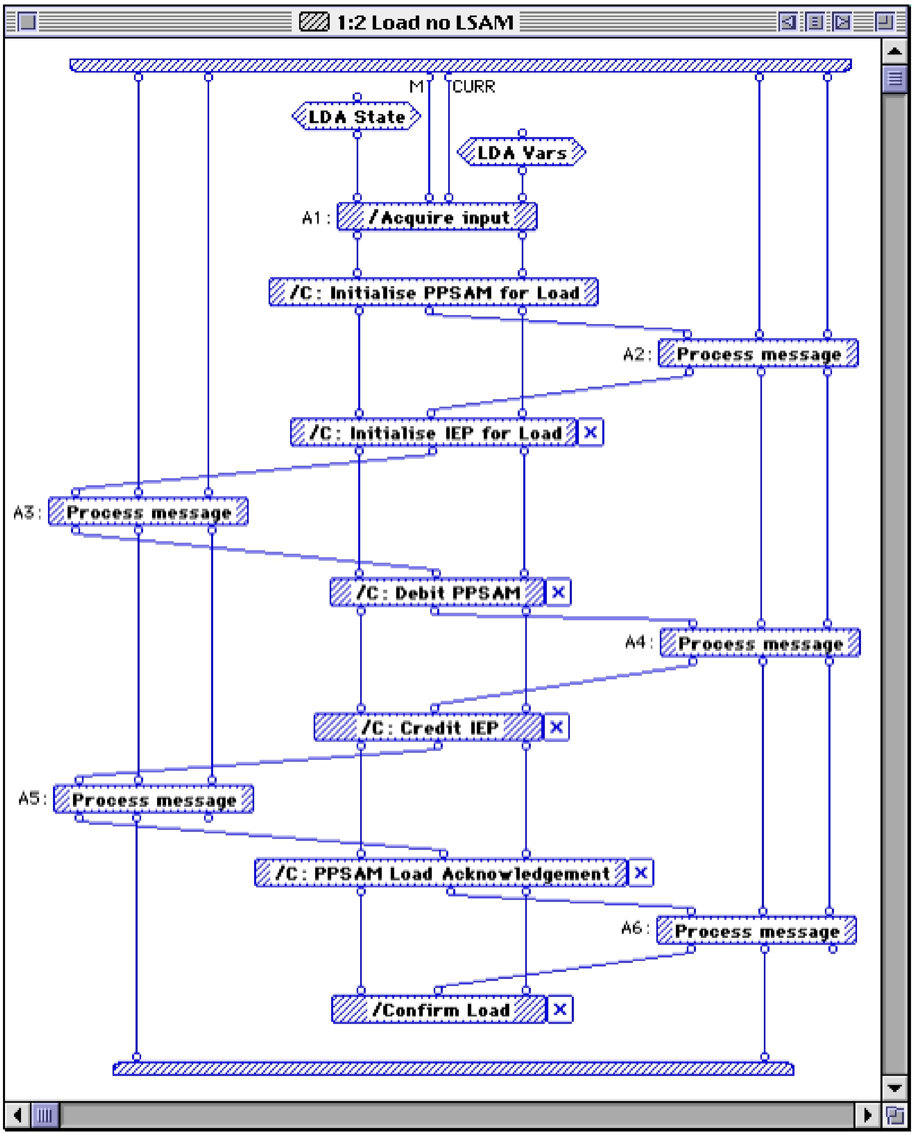

Fig. 5. The method which defines the Load no LSAM transaction protocol.

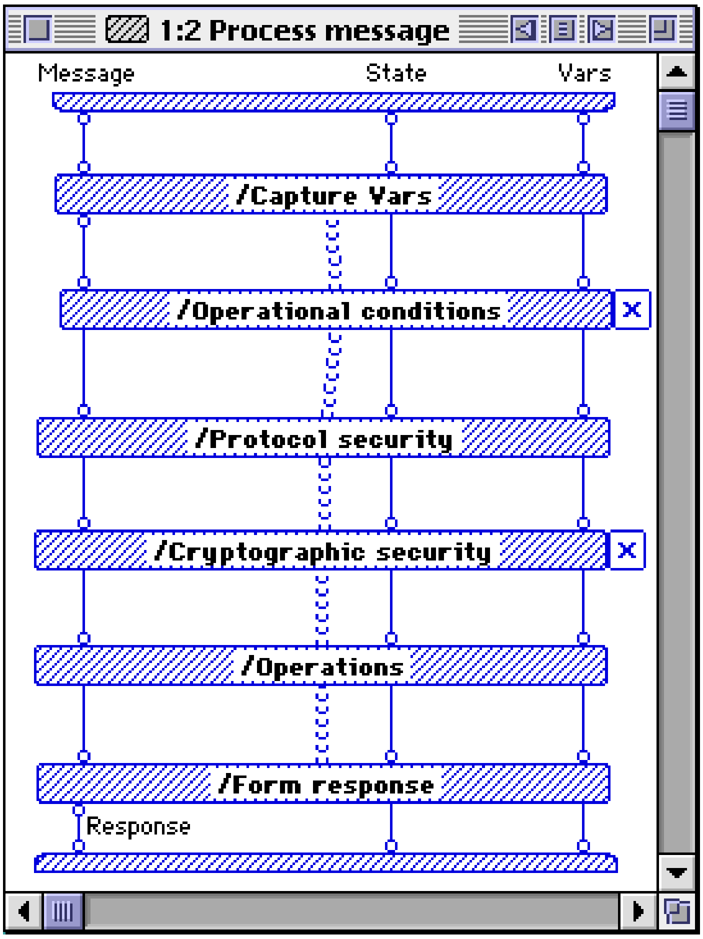

Fig. 6. Process message method conforming to the refined substep structure of each step in the protocol.

Having looked at the transaction at the level of the message and component state flow, we now wish to examine the processing of the individual messages. Looking inside the *Process message* operation of Fig. 6, we find that it follows exactly the structure of Sect. 3. We can also see clearly where the processing of a message might cause failure of the step. The data flow edges connecting the boxes indicate that in addition to the permanent *State* and ephemeral *Vars* the incoming message is also threaded from sub-step to sub-step. This is to tell each of the methods contained in the sub-steps to which message to respond.

We have used Prograph *synchros* (the connections made from the letter u)

to ensure that we know the order of operation execution, in this case so that, for example, security is checked before any operations take place.

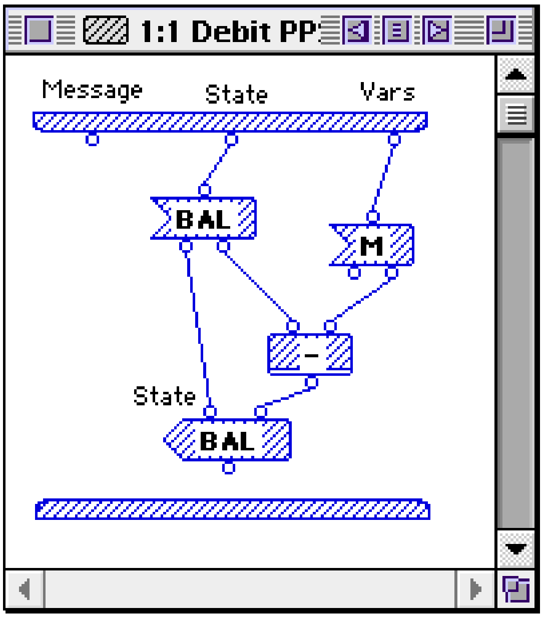

Fig. 7. The Operations method for the PPSAM in response to Debit PPSAM.

Continuing down through the structure, we now look at the details of processing a particular message, *A4: Debit PPSAM*. As a simple example, opening the *Operations* operation for this message, we see (Fig. 7) the actual operation $\mathrm{BAL}_{\mathrm{PPSAM}}{:=}\mathrm{BAL}_{\mathrm{PPSAM}} - \mathrm{M}_{\mathrm{LDA}}$ from the one page transaction overview of Fig. 2. The specially-shaped operations are Prographs way of accessing components of structures. The two at the top of the method access the $\mathrm{BAL}_{\mathrm{PPSAM}}$ and $\mathrm{M}_{\mathrm{LDA}}$ components (which leave on the right hand outputs) and the one at the bottom sets the $\mathrm{BAL}_{\mathrm{PPSAM}}$ attribute to the new value. The rectangular box labelled with the $-$ sign performs the subtraction.

As another example of looking deeper into the structure, we show the *Cryptographic Security* operation (again of the PPSAM in response to Debit PPSAM). This operation deals with two signatures: it checks S_1 and creates S_2. The dataflow view on this is provided in Fig. 8. This shows that both operations need access to the ephemeral and permanent states, but not to the message. At this level we do not see exactly what data elements are required, which would be the purpose of the next level down, that is the methods *Check S1* and *Create S2*. Checking signature S_1 may fail. The creation of S_2 cannot fail, but instead its

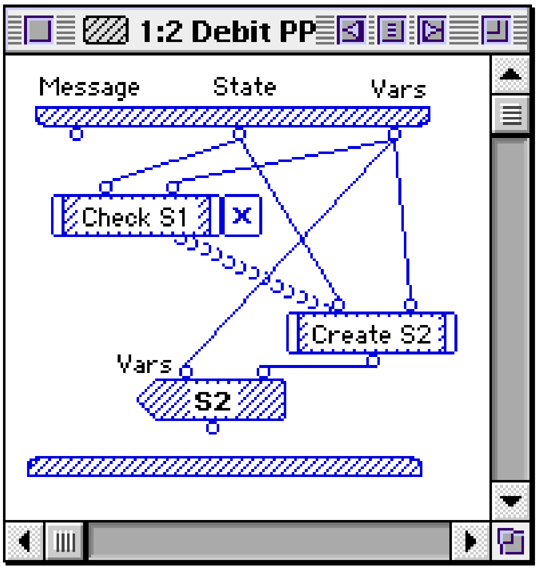

Fig. 8. The Cryptographic Security method for the PPSAM in response to Debit PPSAM.

value must be recorded in the ephemeral state. Because of the use of locally defined operations (with white bars at the sides) in the dynamic system, it is easy to name the more complex operations. We can also see that it is independent of the cryptographic system being used.

Looking inside the *Create S2* local shown in Fig. 9, we find that it has been possible to separate the signature processing into two steps. The first is to gather all the values to be put into the signature, as well as any keys and algorithms from the component (these are in $\overline{\text{IK}}_{\text{PPSAM}}$). The second is then to apply the cryptographic routines of the system being used. In this way we have provided cryptographic neutrality by encapsulating the cryptographic details as tightly as possible.

The final method to shows the start of the cryptographic system specific part, which is where methods for dealing with each of the signatures must be provided for each cryptographic system. Thus Fig. 10 shows the *Make S2* details for DES. It is interesting to note that it is here that the decisions of the cryptographer are documented, which are frequently used to optimise some calculation. In this case there are two such decisions, which our Prograph rendering of the protocol brings to the fore.

The first is the choice to create the session key ($\mathbf{KSES}_{\text{PPSAM}}$) once and to use it twice: to check S_1 and to create S_2. Figure 10 shows that the session key $\mathbf{KSES}_{\text{PPSAM}}$ is simply picked up from the ephemeral state. The code for *Check*

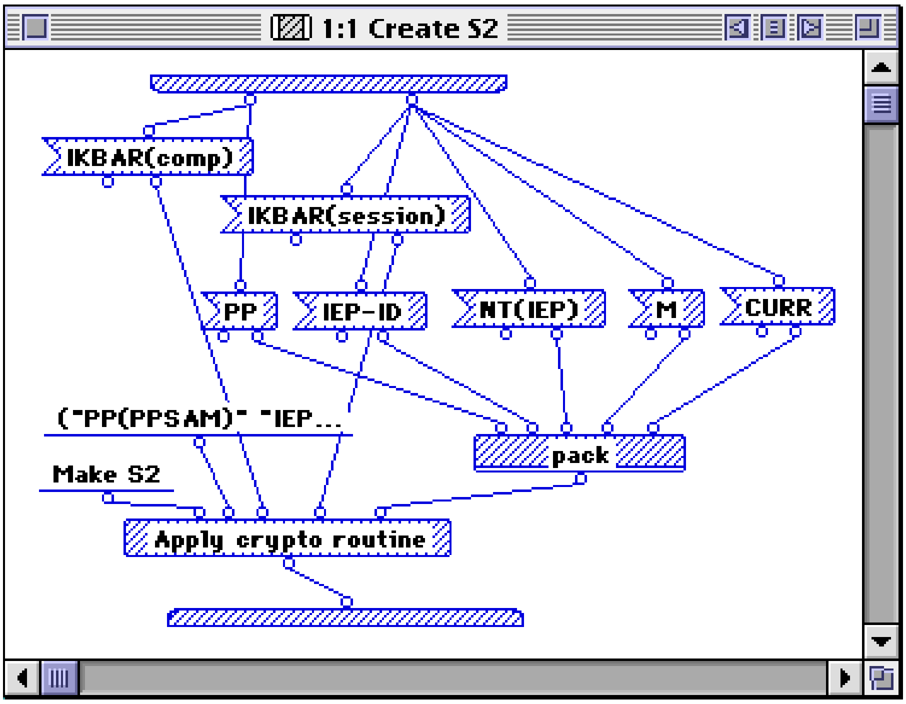

Fig. 9. The Create S2 local from the Cryptographic Security method for the PPSAM in response to Debit PPSAM.

S1 (not shown) puts the session key into the ephemeral state.

The second decision is to sign only $\mathrm{M}_{\mathrm{LDA}}$ and $\mathrm{CURR}_{\mathrm{LDA}}$, instead of the data $\mathrm{PP}_{\mathrm{PPSAM}}$, IEP, $\mathrm{NT}_{\mathrm{IEP}}$ and $\overline{\mathrm{M}}_{\mathrm{LDA}}$, as suggested at step A4 in Fig. 2. (Remember that the notation $\overline{\mathrm{M}}_{\mathrm{LDA}}$ stands for the pair $\mathrm{M}_{\mathrm{LDA}}$ and $\mathrm{CURR}_{\mathrm{LDA}}$). Omitting the data is permitted, as they have been used in computing the session key. One would have to study the details of Fig. 3 quite carefully to see what is happening. In the Program code omitting the first three elements of the list is shown as an explicit discarding using *split-nth*.

The final point about building a prototype in an object oriented system is that certain mistakes are easily found. Consider the code labelled 'Verify IEP signature' in Fig. 3. We have shown in bold face the data that did not appear in the one page overview. As could be expected, all data to do with DES, such as $\mathbf{KSES}_{\mathrm{PPSAM}}$ appear in bold face. Such data could not appear in the one page overview of Fig. 2, as that would have made crytographic neutrality impossible. The appearance of $\mathbf{CURR}_{\mathrm{IEP}}$ is surprising, for it is a data element belonging to the IEP component. The code labelled A4 is part of the PPSAM. There are at least two possibilities. Firstly it could be that the standard contains a

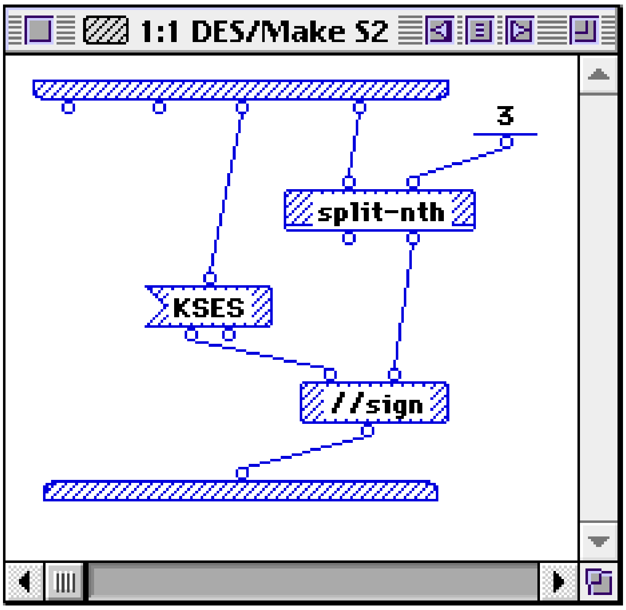

Fig. 10. The Make S2 method from DES Encryption system.

mistake. Secondly this could be an optimisation relying on the fact that at step A4, CURR_{LDA} and $\textbf{CURR}_{\text{IEP}}$ represent the same value. The construction of the conformant prototype of the standard has revealed a number of such issues. Their description is beyond the scope of this paper, as it would require us to present more detail of the standard.

6 Related Work.

Protocol structure is generally discussed in terms of the language used to express the protocols. At one end of the spectrum we find general purpose programming languages. They are often used to specify and or implement protocols. This provides little opportunity for appropriately structuring the protocols. At the other end of the spectrum one finds model-based protocol specifications [2, 4]. Widely known formal languages such as LOTOS and Estelle provide a high level abstract view of protocols and impose structure on the protocols. It is interesting to note that development of both LOTOS and Estelle were part of the standardisation efforts of OSI.

Abbot and Peterson [1] argue the necessity to avoid both extremes. This has some elements in common with our work. Their object oriented language Morpheus is designed for protocol specification. The key idea of Morpheus is to provide a fixed number of so-called shapes, as building blocks for protocols.

The shapes are similar to our components. A shape is a sub class of an abstract super class. Code and data reuse of the abstract class and of the shape make it easy to create specific instances of shapes. These instances are then used as building blocks for a protocol. Morpheus allows only a number of predetermined shapes (a multiplexor, a worker and a router). Abbott and Peterson argue that this provides their Morpheus compiler with more scope for optimisation. Our conformant prototyping system does not need to deliver high performance and offers more flexibility by using a general purpose programming language. Our protocols are represented using only a fixed set of classes, which are used in a disciplined fashion. Abbott and Peterson note that protocols from standards are generally difficult to represent because of their concern for efficiency. In fact they represent the functionality of a standard protocol without necessarily adhering to its exact syntax. Our rendering of the transaction protocols from the standard is exact in that sense. We have prototyped every detail of the load purse transaction as specified in the standard. Admittedly, we are not aiming to specify arbitrary protocols, but a more restricted family of protocols.

Many tools have been developed to study complex protocol behaviour. The Interrogator is an automatic protocol security analysis tool [10]; the language Argos [8] supports automatic protocol validation using a mix of depth-first and breadth-first search for particular scenarios; the Protean tool [3] is designed to detect deadlock in complicated protocols specified as petri nets. The emphasis in all these systems is on understanding and reasoning about the dynamics of complex protocols. The understanding is achieved by animations and simulations of the protocols. The Proteon system, for example, produces animated displays of the history of a protocol, offering the user the opportunity to view a message flowing on an arc, and even to change the message. Such facilities are also offered by our system, because they are provided by Prograph. Our prototype thus offers sophisticated functionality at no extra effort.

Many authors who work with protocols note that the lack of structure in the protocols causes them problems. Billington et al [3] and Bochmann et al [13] discuss the possibility of separating normal operations from exceptional conditions, and thus enriching the structure of protocols. Tel [12], in the context of proving properties of distributed algorithms in general and protocols in particular, suggests a number of simple rules that enhance the structure of a protocol. Tel also suggests that properties of protocol skeletons may carry over to fully-populated protocols, provided certain precautions are taken.

Our restricted setting of the authentication protocols has made it possible to separate out exceptional from normal operations, and also to separate out the security operations. This gives an extra level of structure in our protocols. To our knowledge this has not been achieved elsewhere.

7 Conclusions.

The standard way of visualising protocols using pictures with boxes (representing operations) and arrows (representing messages) is appropriate for a global study of a protocol. The standard visualisation method is insufficient to study the protocols in detail, the problem being that the structuring of the protocols relies on elements not explicit in the standard visual rendering. Such elements include the security and the state of the interacting components. To render the structure properly one should visualise not only the operations and the messages but also the state and the security.

Using the data flow model makes it possible to explicitly render the state manipulations implied by the protocols. Using an object oriented design methodology makes it possible to be selective about the state elements that are accessed and updated at various points in the protocols. A system combining both models is thus appropriate to deal with the manipulation of state in the protocols.

Security in an abstract sense relies on cryptographic systems. In a concrete sense it is the encapsulation of the cryptographic data and operations that determines whether the system is robust or fragile. We have shown how the separation of steps in the protocol into a number of judiciously chosen sub-steps separates out the important protocol, security and operational aspects. The design method that we have used supports this subdivision of larger into smaller steps. It has pointed out where the standard may be unclear. This indicates that building a highly structured prototype and using an object oriented data flow model which supports the encapsulation of data and operations gives a high level of confidence in the robustness of the system.

The combination of data flow, object orientedness and visual programming is provided by the Prograph language that we have been using to build a prototype of some of the protocols in the load purse transaction of the CEN Inter-sector electronic purse draft standard. The Prograph prototype makes it possible to animate the protocols and to perform simulations and experiments with the protocols. This feedback of the system to the designer is a powerful design tool.

8 Acknowledgements.

We thank Hugh McEvoy and the referees for their comments on a draft version of the paper. Jon Hallet provided assistance with the rendering of the screen shots.

References

1. M. B. Abbott and L. L. Peterson. A language-based approach to protocol implementation. In *Communications architectures & Protocols (SIGCOMM)*, pages 27–38, Baltimore, Maryland, Oct 1992. ACM Computer communication review, 22(4).
2. M. S. Atkins. Experiments in SR with different upcall program structures. *ACM transactions on computer systems*, 6(4):365–392, Nov 1988.
3. J. Billington, G. R. Wheeler, and M. C. Wilbur-Ham. PROTEAN: A High-Level petri net tool for the specification and verification of communication protocols. *IEEE transactions on software engineering*, SE-14(3):301–316, Mar 1988.

4. A. Dupuy, J. Schwartz, Y. Yemini, and D. Bacon. BEST: A network simulation and prototyping testbed. *CACM*, 33(10):64–74, Oct 1990.
5. CEN European Committee for Standardization. Identification card systems – inter–sector electronic purse. Draft standard prEN 1546, European Committee for Standardization, Brussels, Nov 1995.
6. F. R. Giles, P. T. Cox, and T. Pietrzykowski. Prograph: A step towards liberating programming from textual conditioning. In *Workshop on Visual Languages*, pages 150–156, Rome, Italy, Oct 1989. IEEE Computer society press, Washington.
7. P. H. Hartel and E. K. de Jong Frz. Towards testability in smart card operating system design. In V. Cordonnier and J-J. Quisquater, editors, *1st Smart card research and advanced application conference (CARDIS 94)*, pages 73–88, Lille France, Oct 1994. Univ. de Lille, France.
8. G. J. Holzmann. Automated protocol verification in *argos* : Assertion proving and scatter searching. *IEEE transactions on software engineering*, SE-13(6):683–696, Jun 1987.
9. J. R. McGraw. The VAL language: Description and analysis. *ACM transactions on programming languages and systems*, 4(1):44–82, Jan 1982.
10. J. K. Millen, S. C. Clark, and S. B. Freeman. The Interrogator: Protocol security analysis. *IEEE transactions on software engineering*, SE-13(2):274–288, Feb 1987.
11. S. Stepney. *High integrity compilation: A case study.* Prentice Hall, Hemel Hempstead, England, 1993.
12. G. Tel. *Topics in distributed algorithms.* Cambridge Univ. Press, Cambridge, England, 1991.
13. G. v. Bochmann and J. P. Verjus. Some comments on "Transition-Oriented" versus "structured' specification of distributed algorithms and protocols. *IEEE transactions on software engineering*, SE-13(4):501–505, Apr 1987.
14. A. H. Veen. Dataflow machine architecture. *ACM computing surveys*, 18(4):365–396, Dec 1986.

This article was processed using the LaTeX macro package with LLNCS style