FORMAL DESIGN OF EFFICIENT AUTHENTICATION AND KEY AGREEMENT PROTOCOLS

Gunnar Jacobson
Siemens AG Corporate Technology, D-81730 Munich, GERMANY
gunnar.jacobson@mchp.siemens.de

Abstract We present a new method for the formal requirements specification and the design of Authentication and Key-Agreement protocols. The method "SDL combined with inverse BAN logic", SDL/*i*BAN is based on the inverse application of the BAN logic of Burrows, Abadi and Needham and the integration with the Specification and Description Language SDL. The exemplary formal design of Kerberos demonstrates the applicability and reliability of the method. We classify cryptosystems and protocol runs and provide a generic design approach, which is on an idealised layer independent from the cryptosystem to be used. We show, how concrete specifications of new protocols may be derived and propose the integration of our method with existing specification methods and software development tools.

1. BACKGROUND AND MOTIVATION

Communication over public networks must be protected by providing the required security services like authentication, confidentiality and non-repudiation. Such security services are implemented by integrating cryptographic mechanisms into the communication protocols. For an authenticated communication, a protocol entity has to prove its identity to its partners. In a mutually authenticated session, all parties are convinced of the partner's identity and the integrity and timeliness of exchanged messages. For a confidential communication, unauthorised read access to the protocol messages must be prevented by encrypting them, so encryption and

B. Preneel (ed.), *Secure Information Networks*

decryption keys have to be agreed upon. For a non-repudiable communication, the origin and destination of the protocol messages must be provable, so signature and validation keys have to be agreed upon. The problem of entity authentication and key agreement grows with the square of the number of involved entities, so efficient and reliable mechanisms are required for this purpose.

Diffie and Hellman introduced the concept of public key cryptography for solving the key distribution problem in large, public networks in 1976 [7]. Needham and Schroeder made proposals for authentication and key agreement (AKA) protocols using conventional, shared key and public key cryptography with the help of a trusted third party [16]. These and further publications of AKA protocols were the subject of informal analysis and discussions and many protocols have shown to be flawed. The detection of flaws often takes place years or even decades after the initial protocol publication. Denning and Sacco showed a weakness in the Needham-Schroeder shared key protocol in 1981 and Lowe did this for the public key protocol in 1996 [6][13].

Flaws in AKA protocols bury a high risk, because it is mostly much easier for an attacker to break the protocol, than to break a cryptographic algorithm. In order to analyse AKA protocol specifications in an efficient and reliable manner, formal analysis methods were developed [14][11][9][3]. A well-known approach is the so called BAN logic of Burrows, Abadi and Needham [4]. Work on AKA protocol design is even now mostly limited to the definition of informal design principles, which can not be considered as an adequate mechanism for the design of reliable, efficient AKA protocols [1][2]. Meadows demands formal design methods for this purpose [15]. Buttyan et al. present a formal logical design method for AKA protocols, which uses channels, that are a generalisation of communication links [5]. In the following, we present a new method for the formal requirements specification and design of AKA protocols and show how it fits into a layered development process for AKA protocols.

2. A LAYERED DEVELOPMENT APPROACH

We divide a development process into four phases and corresponding layers. Each layer uses the results of the above layer as the starting point:

1) Requirements Specification
2) Design
3) Specification

4) Implementation

1) The development process begins with the specification of requirements for the AKA protocol. They are basically derived from the security requirements of the communication service to be protected, the chosen trust-model and assumptions made on the infrastructure. The requirements are described using the formal language of layer 2).
2) The main purpose of this layer is the design of a reliable and secure protocol from a logical point of view. The required messages, their ordering and their security mechanisms are derived from the requirements specification and represented in a new form, which may be easily transformed into the layer 3) representation.
3) The specification layer delivers an abstract specification of the protocol, from which interoperable implementations may be derived. We propose the use of combined SDL/ASN.1 specifications in this layer [10]. With this method, data structures and protocol behaviour can be represented in a specification diagram. The transformation to layer 4) can be done automatically using existing tools.
4) The lowest layer, the implementation layer, may consist of source-code modules, libraries, compilers and debuggers for a high-level language like "C". The result of this layer is executable object code of the AKA protocol for a specific computer and operating system.

Our new method for the implementation of the layers 1) and 2) is introduced in the following. For layers 3) and 4), the reader is encouraged to consult the existing literature.

3. A FORMAL DESIGN METHOD

The original purpose of the BAN logic is the analysis of formal AKA protocol descriptions which were derived from an informal protocol description through an idealisation process. The analysis starts with the initial assumptions of the protocol. Logical formulas, assertions about the state of the system, before and after each protocol step are derived by applying the inference rules of the logic. The assertions after the last protocol step contain the conclusions of the protocol - formulas representing the belief in keys or secrets.

The basic approach of our new design method is the inversion of the original BAN analysis process. The starting point for the protocol design process is the definition of the final conclusions Z and initial assumptions Y from the requirements specification which are described using the BAN

syntax (see App. A). We start at the end of the protocol with the final conclusions and apply the inverted rules of the logic backwards in an iterative process until we reach the initial assumptions. In this way, we get the required logical send statements, representing the messages of the idealised protocol. The design may be divided into two sub-layers. The logical layer on top defines the process of exchanging formulas about the entity's state of belief. The idealised layer below defines security mechanisms for the protection of the exchanged formulas. We call this approach the "inverse BAN logic", *i*BAN.

The BAN logic does not consider time in a protocol run, it makes only a distinction between past and present. Moreover, the direction of transmitted messages may not be expressed using BAN. In order to enhance these features, we propose the integration of BAN with SDL, see fig. 1. The SDL diagram represents the timely behaviour and message flow of the protocol. The SDL symbols contain BAN expressions. The SDL "send" and "receive" symbols represent the protocol messages in a new form, corresponding to the "says" and "see" expressions of the original BAN syntax. SDL decision symbols contain sending and receiving conditions for messages. The initial assumptions and final conclusions are contained in assignment blocks.

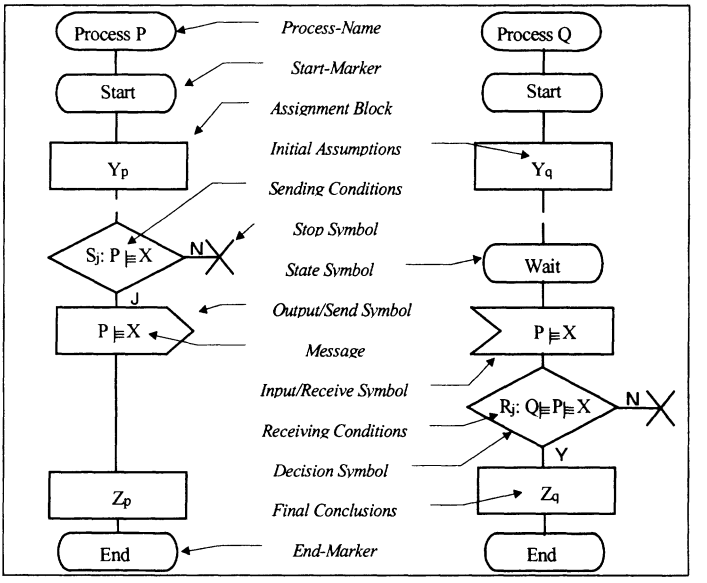


Figure 1: Elements of SDL/*i*BAN

Fig. 1 contains exemplary expressions of the logical layer. P and Q are variables on users. We insert initial and final states in the diagram for each process and assign the initial assumptions Y and final conclusions Z from

the requirements specification. A final state may only be entered, if all final conclusions hold after receiving the last message. For this, we insert an SDL decision symbol containing Z, e.g. $Q \models P \models X$, as receiving conditions R_j. We postulate, that an expression of the form $Q \models P \models X$, which is not an initial assumption, must be preceded by Q's receipt of the expression $P \models X$. As no further processes are defined in this example, P must have sent this expression to Q with message M_j. Before a process P may send statements about its current belief, these expressions must hold. For this, we insert a decision symbol with the corresponding sending condition S_j. From the sending conditions, we derive the required receiving conditions R_{j-1} of Message M_{j-1} and proceed with the iteration until we reach sending conditions S_1 that are accomplished by the initial assumptions.

Now, we transform the protocol into the idealised layer. We postulate, that private and secret keys may not be revealed to unauthorised users, i.e. $R \triangleleft P \overset{K}{\leftarrow}$ or $R \triangleleft P \overset{K}{\leftrightarrow} Q$ is not allowed if $R \neq Q \neq P$ and R is not a trusted party. An appropriate basic cryptosystem or a combination of them has to be chosen which provides the required security properties, see Appendix F. Then the corresponding BAN rules have to be applied. Fig. 2 shows an example of the idealised layer. Q's initial assumptions Y on the secret key and nonce used are shown in the block after the process start marker. Based on the receiving conditions and the initial assumptions, we derive expressions of the form $P \triangleleft X$ using the inverted message meaning rules. Then we replace the logical expressions in the diagram with the derived expressions in idealised form.

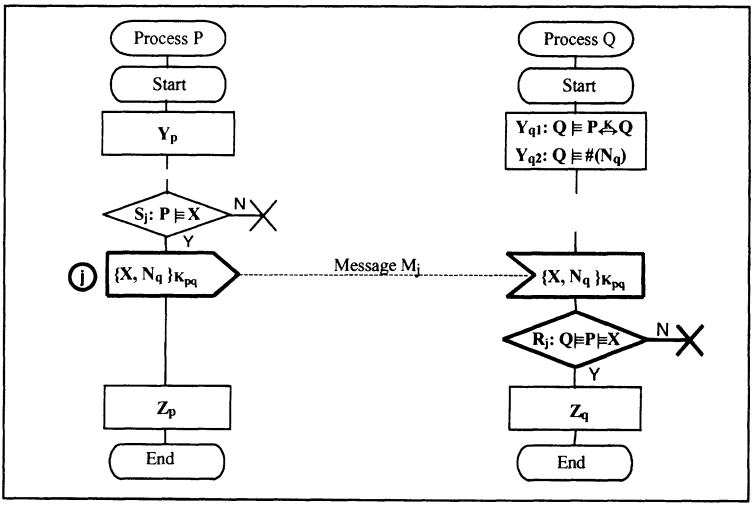


Figure 2: SDL/*i*BAN – Idealised layer

We call this method "SDL combined with inverse BAN logic", SDL/*i*BAN. The representation of the design results may facilitate the transformation into the standardised SDL/ASN.1 specification form of layer 3) [10]. In Appendix B we show how the idealised messages in SDL/*i*BAN are mapped to a conventional notation which is commonly used in the specification literature. By this, we are able to make our results comparable with existing publications. By providing strict rules for this transformation process, we aim at minimising the risk of inserting flaws at this step. The informal idealisation process of the original BAN logic, however, may lead to misinterpretations. Lowe discovered a flaw in the public key protocol in [16], which was not detected by BAN, because the protocol idealisation masked the problem [4][13].

4. APPLYING THE METHOD

The applicability of the method is demonstrated now, by deriving a popular and well understood AKA protocol, Kerberos [12]. In the requirements specification, we demand second level beliefs in a symmetric Session key $A \overset{K}{\leftrightarrow} B$ for both parties A and B as final conclusions of the protocol. A is the initiator and B is the responder in the protocol. The initial assumptions express the party's trust in keys shared with an authentication server S. The timeliness of protocol messages is guaranteed by using timestamps of synchronised clocks. We define the corresponding belief statements using BAN and insert them into the SDL/*i*BAN diagram in Appendix C. In the following, we show how to derive messages M_j, receiving conditions R_j and sending conditions S_j by applying the BAN rules inversely, according to chapter 3. The results are shown in App. C.

The iterative design process starts on the logical layer with the last message M_n, which is derived from the 2nd level belief of one party. The number of messages in the protocol is unknown at this step, later on we set n=3. We are free to either select A or B as the receiver of message M_n, and choose A[1]. The 2nd level belief of A is expressed as:

$Z_{a2})\ A \models B \models A \overset{K}{\leftrightarrow} B$

According to chapter 3, the receiving condition is set to $R_n = Z_{a2}$ and we get the expression $B \models A \overset{K}{\leftrightarrow} B$ which is sent from B to A. B may send this last message only, if the expression holds at this step, so one sending condition S_n (=$S3_1$, see App. C) is $B \models A \overset{K}{\leftrightarrow} B$. Furthermore, all the required final

[1] We will show the result for the inverse case further below.

conclusions Z_b of B must hold at this step, so we add them as sending conditions ($S3_2$). From the sending conditions, the required receiving conditions for further messages are derived. $R2_1$ is derived from $S3_1$ using the initial assumption Y_{b2}) ($A \models S \Rightarrow A \overset{K}{\leftrightarrow} B$) and the jurisdiction rule. $S3_2$ can not be inferred further, so we set $R2_2$=$S3_2$. From $R2_1$ and $R2_2$, we conclude, that B must have received the expressions $S \models A \overset{K}{\leftrightarrow} B$ and $A \models A \overset{K}{\leftrightarrow} B$ before. We are free to decide, how these expressions are transferred to B, either separately or in a combined message. In Kerberos, the second option is chosen for message M_2 from A to B. Before sending M_2, $A \models A \overset{K}{\leftrightarrow} B$ must hold and A must possess the expression $S \models A \overset{K}{\leftrightarrow} B$, so we get $S2_1$ and $S2_2$. Accordingly, we get $R1_1$ and, by applying the possession rule from [8], we get $R1_2$ ($A \lhd S \models A \overset{K}{\leftrightarrow} B$). From this, we infer M_1 ($S \models A \overset{K}{\leftrightarrow} B$ *for A*, $S \models A \overset{K}{\leftrightarrow} B$ *for B*) from S to A and $S1_1$ ($S \models A \overset{K}{\leftrightarrow} B$). At this point we have reached a sending condition, which is an initial assumption so our process of backwards inference is complete. The protocol could start with M_1, but according to the requirements specification A should be the initiator. So, an initialisation message M_a is added.

Now, we have all protocol elements on a logical layer and the required security mechanisms for all messages M_i may be derived on the idealised layer of the design process. We show as an example how to infer M_n from R_n. The only rule, which helps us to solve this expression with ***i***BAN is the nonce-verification rule, see App. A. We apply this rule inversely and get

1) $A \models \#(A \overset{K}{\leftrightarrow} B), A \models B \mid\sim (A \overset{K}{\leftrightarrow} B)$.

From the second part we derive an expression of the form $A \lhd X$, meaning "A sees X", using the appropriate message meaning rule for shared keys. The expression $A \models \#(A \overset{K}{\leftrightarrow} B)$ from 1) is not an initial assumption. So we have to extend Z_{a2} using the inverted rules for sets of statements and get

2) $A \models B \models \#(X, A \overset{K}{\leftrightarrow} B)$.

X is a variable. We apply the nonce-verification rule inversely and get

3) $A \models \#(X, A \overset{K}{\leftrightarrow} B), A \models B \mid\sim (X, A \overset{K}{\leftrightarrow} B)$.

Using the freshness rule and comparing X with the initial assumptions, we get

4) $A \models \#(T_a, A \overset{K}{\leftrightarrow} B), A \models B \mid\sim (T_a, A \overset{K}{\leftrightarrow} B)$.

The first part of 4) is accomplished using the initial assumptions. The second part is derived to

5) $A \models A \overset{K}{\leftrightarrow} B, A \lhd \{ T_a, A \overset{K}{\leftrightarrow} B \}_{K_{ab}}$.

The first part of 5) is equal to $S2_1$ and will be solved in further steps. The second part represents the idealised last message of Kerberos from B to A.

We insert it in the SDL/*i*BAN diagram of the idealised layer in Appendix C. We proceed with the derivation of all messages M_i and conditions S_i, R_i from the logical layer and add them to the diagram. The expression in R_a means, that S sees a request for a new session key which is to be shared between A and B. According to our mapping rules for secret cryptosystems from Appendix B, we transfer the expressions into the conventional representation of the specification layer:

M3: $\{ T_a, B \}_{K_{ab}}$ M2: $\{ T_s, A, K_{ab} \}_{K_{bs}}$, $\{ T_a, A \}_{K_{ab}}$

M1: $\{ T_s, B, K_{ab} \}_{K_{as}}$, $\{ T_s, A, K_{ab} \}_{K_{bs}}$ Ma: A, B

The result corresponds with the ticket granting protocol of Kerberos version V. The redundant double encryption of the server ticket, like in the repeatedly criticised Kerberos version IV, is avoided when using our method. M3 differs slightly from the original specification, where B's name is omitted. Ticket lifetimes in the original are an implementation aspect, and could be added to the mapping rules.

5. GENERIC DESIGN

In the following, we present a universal, generic design model for efficient AKA protocols and show how to infer generic design schemes using the basic method SDL/*i*BAN. Such a generic design scheme is independent from the concrete cryptosystems and freshness mechanisms to be used in a protocol. Different concrete protocol design results may be easily derived from the generic scheme. The basic idea comes from the observation, that today's cryptographic systems can be classified into a generic model. The model considers the function of a key in a protocol, which can be the provision of confidentiality by encryption and decryption, or authenticity by signing and proving, see also App. F.

In the most general case, which we consider here, we have one key for each operation at each of the communication partners. We call such a cryptosystem, according to the number of keys, CS-8. In fact, CS-8 may be implemented by a combination of available cryptosystems. For example, the El-Gamal public key encryption algorithm, providing solely confidentiality with Q's public encryption key $\overset{K^c}{\rightarrow}Q$ and P's private decryption key $P\overset{K^d}{\leftarrow}$, may be combined with NIST's Digital Signature Algorithm using P's private signature key $P\overset{K^s}{\leftarrow}$ and the public proving key $\overset{K^p}{\rightarrow}Q$ for Q's signature. By inheriting the features of encryption and signature proving keys from CS-8 we get one private key per user, $P\overset{K}{\leftarrow}$. The same is done for decryption and signature keys, where we get one user's public key, $\overset{K}{\rightarrow}P$. We call this type

CS-4. RSA could be an example implementation of CS-4 [18]. CS-8 and CS-4 are both types of public key cryptosystems. We proceed and inherit the features of the user's private key and his partner's public key and get an asymmetric key pair, $P\overset{K}{\leftarrow}(Q)$ and $Q\overset{K}{\leftarrow}(P)$, for a pair of users. Each user has to keep his key of CS-2 private. All cryptographic operations in a secure communication with a partner are done with such a multi-purpose key. The Pohlig/Hellman algorithm is an example implementation of CS-2 [17]. All of the above cryptosystems support non-repudiation with the help of a trusted third party, which either certifies a user's public key of CS-8 or CS-4 or generates and distributes keys of CS-2 and is therefore able to prove the origin of a message definitely. We get this feature by using asymmetric algorithms. Asymmetric algorithms are known to be relatively slow and therefore not well suited for data encryption. Encryption is mainly done with symmetric cryptosystems of type CS-1, where each user pair has a common secret key $P\overset{K}{\leftrightarrow}Q$ ($=Q\overset{K}{\leftrightarrow}P$), e.g. the DES algorithm. CS-1 is derived from CS-2 by inheriting the features of both user's private keys. All cryptographic operations in a bilateral communication are done with the same key, hence the proof of a message's origin cannot be definite.

Crypto system	Keys used by P				Keys used by Q			
	encrypt	prove	decrypt	sign	encrypt	prove	decrypt	sign
CS-8	$\overset{K^c}{\rightarrow}Q$	$\overset{K^p}{\rightarrow}Q$	$P\overset{K^d}{\leftarrow}$	$P\overset{K^s}{\leftarrow}$	$\overset{K^c}{\rightarrow}P$	$\overset{K^p}{\rightarrow}P$	$Q\overset{K^d}{\leftarrow}$	$Q\overset{K^s}{\leftarrow}$
CS-4	$\overset{K}{\rightarrow}Q$		$P\overset{K}{\leftarrow}$		$\overset{K}{\rightarrow}P$		$Q\overset{K}{\leftarrow}$	
CS-2	$P\overset{K}{\leftarrow}(Q)$				$Q\overset{K}{\leftarrow}(P)$			
CS-1	$P\overset{K}{\leftrightarrow}Q$							

Table 1: Generic classes of keys

The execution efforts of an AKA protocol depend mainly on the number of protocol steps, the efforts for cryptographic operations and the amount of exchanged data elements. We call an AKA protocol "efficient", if these efforts are low for a given combination of requirements for initial assumptions and final conclusions. In order to determine a minimum number of protocol steps, we analyse the exchange of logical statements: Using communicating sequential processes for protocol design, the evolution of logical beliefs must also be sequential. This means, that a 2nd level belief in a formula must be preceded by the corresponding 1st level belief. Fig. 3 shows the possible evolution in the case of 2nd level beliefs for both parties. The symbol "⇨" indicates the point of time of a message transmission. There are always two mirrored variants of a protocol, depending on who, either the initiator (P=A) or the responder (P=B), reaches the final conclusions at last.

$S \not\models X_{pq}$, $S \Rightarrow X_{pq}$ $S \not\models X_{qp}$, $S \Rightarrow X_{qp}$	⇨	$P \models X_{pq}$	⇨	$Q \models X_{qp}$ $Q \models P \models X_{pq}$	⇨	$P \models Q \models X_{qp}$

Figure 3: Logical belief evolution

We distinguish between two phases of an AKA protocol. During the agreement phase, keys or secrets are exchanged using cryptographic messages. The protocol, however starts with a synchronisation phase, where an initialisation message and data representing timeliness may be exchanged. Timeliness may be guaranteed using Nonces N_p and N_q, which are generated and relied upon by either user, or using timestamps T of synchronised clocks. In the following, we consider timestamps as a special case, where $N_p=N_q=T$. In our generic design, we use N_p and N_q.

Now, we have introduced all the building blocks for our generic design method and can start to describe the design process. For the requirements specification and the generic design of efficient AKA protocols, we use cryptosystem CS-4 in our classification, for the representation of exchanged session keys as well as for applied master keys. The reason for this is, that the original BAN logic does not cover cryptosystems of our most general type CS-8. In the requirements specification, we define the number of involved parties, the final conclusions and the initial assumptions using BAN. The assumptions on trust in the authority on keys are fundamental and lead to different generic protocol schemes with two or three parties. We use our method SDL/*i*BAN for the backward inference of messages and conditions and start from the final conclusions, defined in the requirements specification. In the idealisation layer we use the message meaning rule for public keys of CS-4. Concrete protocol designs may be derived from a generic protocol scheme by combining generic keys according to table 1. The result is a SDL/*i*BAN diagram which makes use of a particular cryptosystem on the design layer. The design result may be mapped to the specification layer using mapping rules, as in chapter 4.

6. EXAMPLES

In App. D, the idealised, generic AKA three-party protocol is shown, as an example result of applying our design method from chapter 3[2]. Master keys are marked by an asterisk. The SDL connection symbols link with the SDL/*i*BAN diagram of the chosen synchronisation phase in App. E.

[2] The detailed sending and receiving conditions, are left for clarity.

Now, we show how we could derive Kerberos from the generic protocol: We choose CS-1 for the session keys as well as for the master keys. This means, that according to Table 1, the class CS-4 keys $\overset{K}{\rightarrow}Q$, $P\overset{K}{\leftarrow}$ and $\overset{K}{\rightarrow}P$, $Q\overset{K}{\leftarrow}$ are combined to a single class CS-1 key $P\overset{K}{\leftrightarrow}Q$. The master keys $\overset{K}{\rightarrow}Q^*$ and $S\overset{K}{\leftarrow}$ are combined to $S\overset{K}{\leftrightarrow}Q$. The nonces N_q and N_p are replaced by a timestamp T, using synchronised clocks, as we proposed in chapter 5. As a consequence of combining authenticity and confidentiality using one key, we get redundant formulas after the replacement in the generic diagram. We delete those redundant parts, which do not provide the required confidentiality for the secret session key. This leads to the following protocol messages in idealised form, which we may replace for the generic messages in the SDL/*i*BAN diagram in Appendix D:

M3 $P\rightarrow Q$: $\{T\}_{K_{pq}}$

M2 $Q\rightarrow P$: $\{T, P\overset{K}{\leftrightarrow}Q\}_{K_{ps}}, \{T\}_{K_{pq}}$

M1 $S\rightarrow Q$: $\{T, P\overset{K}{\leftrightarrow}Q\}_{K_{qs}}, \{T, P\overset{K}{\leftrightarrow}Q\}_{K_{ps}}$

If we replace variable P with initiator A and variable Q with responder B, then we get the original Kerberos protocol. If we do it just the other way round and set P=B and Q=A, then we get a new, "mirrored" Kerberos protocol. This variant leads to the same protocol goals in a different order. The synchronisation phase consists of a single initialisation message M_a, see App. E. Fig. 5a) shows the mapping to the commonly used conventional representation of the specification layer[3].

Our next example is a new three-party AKA protocol using private master and session keys of CS-2 and nonces. According to our table, $\overset{K}{\rightarrow}Q$, $P\overset{K}{\leftarrow}$ are combined to a private session key $P\overset{K}{\leftarrow}(Q)$, also represented as K_{pq}^{-1} for cryptograms. The master keys $\overset{K}{\rightarrow}Q^*$, $S\overset{K}{\leftarrow}$ are combined to $S\overset{K}{\leftarrow}(Q^*)$, also represented as K_{sq}^{-1}. Accordingly, we get the inverse private session key $Q\overset{K}{\leftarrow}(P)$ and $S\overset{K}{\leftarrow}(P^*)$, also represented as $K_{qp}^{-1} = (K_{pq}^{-1})^{-1}$ respectively K_{sp}^{-1}. After replacing the expressions in the generic diagram and deleting the redundant formulas, we get the idealised messages of the new protocol:

M3 $P\rightarrow Q$: $\{N_q, P\overset{K}{\leftarrow}(Q)\}_{K_{pq}^{-1}}$

M2 $Q\rightarrow P$: $\{N_p, P\overset{K}{\leftarrow}(Q)\}_{K_{sp}^{-1}}, \{N_p, Q\overset{K}{\leftarrow}(P)\}_{K_{qp}^{-1}}$

M1 $S\rightarrow Q$: $\{N_q, Q\overset{K}{\leftarrow}(P)\}_{K_{sq}^{-1}}, \{N_p, P\overset{K}{\leftarrow}(Q)\}_{K_{sp}^{-1}}$

This protocol provides privacy and, with the help of S, non-repudiation with a single cryptographic transformation using a combined encryption and signature algorithm, e.g. [17]. Synchronisation is done using nonces, see

[3] We provide this representation only for the purpose of being comparable with other publications.

App. E. We map the idealised form into the conventional representation of the specification layer, see Fig. 5b).

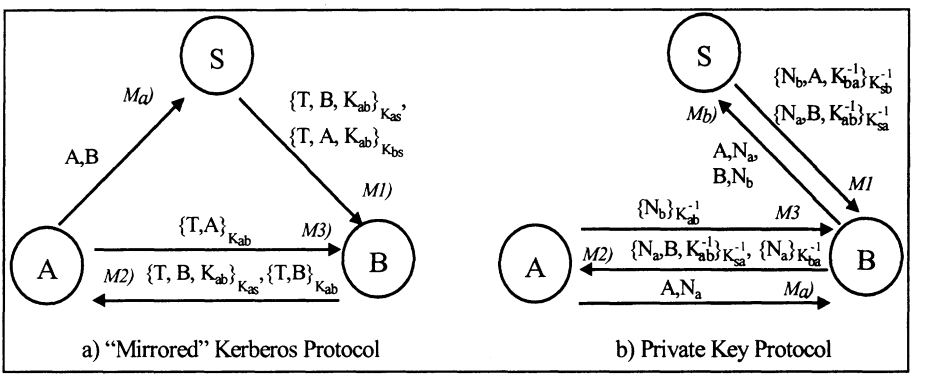


Figure 5: Protocol examples

7. CONCLUSIONS

The method, which we presented here, shall enable the development of AKA protocols in a reliable, efficient and intelligible way. The results may be directly used as design tools by a protocol designer. We offer two options for this: The basic method SDL/*i*BAN may be used for a creative, logical design process. Alternatively, concrete protocols may be derived from generic design schemes without in-depth knowledge of the logic by choosing the relevant parameters.

It was shown, how protocol specifications may be derived from generic design templates. The process, which was demonstrated here for three-party protocols, is also applicable for two-party protocols. We presented mappings to a conventional specification representation in order to compare the results with existing publications. Furthermore, for the proposed general development approach mappings to SDL/ASN.1 specifications are required. This, along with the integration into reliable software generating systems could be subject of future work. The ultimate goal would be the automatic generation of secure and robust code implementing AKA protocols which depends only on the well defined result from the requirements specification.

REFERENCES

[1] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. *Digital Systems Research Center Research Report 125*, 1994.

[2] R. Anderson and R. Needham. Robustness Principles for Public Key Protocols. In *Advances in Cryptology CRYPTO '95 Proceedings, Springer Verlag*, 1995, pp.236-247.

[3] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Advances in Cryptology CRYPTO '93 Proceedings, Springer Verlag*, 1993, pp. 232-249.

[4] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *Digital Systems Research Center Research Report 39*, 1989.

[5] L. Buttyán, S. Staamann, and U. Wilhelm. A Simple Logic for Authentication Protocol Design. In *11th IEEE Computer Security Foundations Workshop*, 1998, pp. 153-162.

[6] D. Denning and G. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM, 24 (2)*, 1981, pp. 58-68.

[7] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on InformationTheory Vol. IT-22 No.6*, 1976, pp. 644-654.

[8] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. *Proc. of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 1990, pp. 234-248.

[9] S. Gritzalis, D. Spinellis, and P. Georgiadis. Security Protocols over open networks and distributed systems: *Formal Methods for their Analysis, Design, and Verification. Computer Communications Journal, Elsevier*, 22(8), 1999, pp. 695-707.

[10] International Telecommunications Union ITU."SDL Combined with ASN.1 (SDL/ASN.1)". *ITU Recommendation Z.105*, 1995.

[11] R. Kemmerer, C. Meadows, and J. Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology, 7(4)*; 1994, pp. 79-130.

[12] J. Kohl. The Evolution of the Kerberos Network Authentication Service. In *EurOpen Conference Proceedings*, 1991, pp. 295-313.

[13] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. Tools and Algorithms for the Construction and Analysis of Systems. In *Lecture Notes in Computer Science, Springer Verlag*,1996, pp. 147-166.

[14] C.A. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security, 1(1)*, 1992, pp. 5-35.

[15] C.A. Meadows. Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology – ASIACRYPT'94 Proceedings, Springer Verlag*, 1994, pp. 133-150.

[16] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in large Networks of Computers. *Communications of the ACM, 21 (12)*, 1978, pp. 993-999.

[17] S. Pohlig and M. Hellman. An improved Algorithm for computing Logarithms in GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory, 24(1)*, 1978, pp. 106-111.

[18] R. Rivest, A. Shamir, and L. Adleman. A Method for obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM, 21(2)*, 1978, pp. 120-126.

APPENDIX A: USED EXPRESSIONS AND RULES OF THE BAN LOGIC

A, B, S	Protocol entities with specific roles: A=Initiator; B=Responder; S=Key Server
P, Q, R	Variables for protocol entities without roles
K_{ab}	Secret key, shared between A and B
K_a	A's public key for communication with anyone
K_a^{-1}	A's private key for communication with anyone
K_{ab}^{-1}	A's private key for communication with B **
N_a	Nonce, generated by A.
X,Y	Variables for statements
K	Variable for keys
,	Conjunction
$P \models X$	P believes in X
$P \lhd X$	P sees X
$P \mid\sim X$	P says (sends) X
$P \ni X$	P possesses X[4]
$P \Rightarrow X$	P has authority on X
#(X)	X is fresh (e.g. a Nonce)
$P \overset{K}{\leftrightarrow} Q$	P and Q share a common secret key K
$\overset{K}{\rightarrow} P$	K is P's public key. This implies, that P owns the inverse private Key K^{-1}
$P \overset{K}{\leftarrow}$	K^{-1} is P's private key. This implies, that an inverse public key K exists [4]
$P \overset{K}{\leftarrow} (Q)$	K_{pq}^{-1} is P's private key, used exclusively for communicating with Q [4]
$\{X\}_K$	X was encrypted using key K

Message meaning rules

$$\frac{P \models P \overset{K}{\leftrightarrow} Q,\ P \lhd \{X\}_K}{P \models Q \mid\sim X} \qquad \frac{P \models \overset{K}{\rightarrow} Q,\ P \lhd \{X\}_{K^{-1}}}{P \models Q \mid\sim X}$$

Nonce-verification rule

$$\frac{P \models \#(X),\ P \models Q \mid\sim X}{P \models Q \models X}$$

Jurisdiction rule

$$\frac{P \models Q \Rightarrow X,\ P \models Q \models X}{P \models X}$$

Freshness rule

$$\frac{P \models \#(X)}{P \models \#(X, Y)}$$

Rules for sets of statements

$$\frac{P \models (X, Y)}{P \models X} \qquad \frac{P \models Q \models (X, Y)}{P \models Q \models X}$$

[4] These expressions are not contained in the original BAN logic.

APPENDIX B: MAPPING RULES

Basic Rules for Public Key Cryptosystems CS-4:

1) $R \vdash \{X, R \overset{K}{\leftarrow} \}_{K_r^{-1}} \Rightarrow \{X\}_{K_r^{-1}}$

This expression means, that R shows, that he owns and is able to apply his private Key K_r^{-1}. Everyone who believes in the public key K_r of R can definitely prove this. On a logical level, the private key is part of the signed expression and represents its meaning. The specification level provides the signed data structure. Data representing the key itself must not occur.

2) $R \vdash \{X, \overset{K}{\rightarrow} Q \}_{K_r^{-1}} \Rightarrow \{X, Q, K_q\}_{K_r^{-1}}$

Here R says, that K_q is Q's public key. On a logical level, the link between Q and its key is explicitly expressed in the formula. On the specification level, the link is provided by naming Q and its key explicitly in the signed data structure.

3) $R \vdash \{Y, \{X, P \overset{K}{\leftarrow} \}_{K_r^{-1}} \}_{K_p^*} \Rightarrow \{Y, \{X, K_p^{-1}\}_{K_r^{-1}} \}_{K_p^*}$

With this message R says, that K_p^{-1} is P's private key. Private keys must always be encrypted for transmission. Only P has the decryption key $(K_p^*)^{-1}$. So, the intended recipient is implicitly defined by using P's encryption key. Therefore, on the specification level, the name of the recipient must not be contained in the data structure.

Derived Rules for Private Key Cryptosystems CS-2:

In CS-2 the signature and encryption keys of CS-4 are combined to a single private key for each of the partners. So we get:

1_2) $R \vdash \{X, R \overset{K}{\leftarrow} (Q) \}_{K_{rq}^{-1}} \Rightarrow \{X\}_{K_{rq}^{-1}}$

This signature can definitely be proved only by the owner Q of the inverse private key K_{qr}^{-1}.

2_2) $R \vdash \{Y, \{ X, P \overset{K}{\leftarrow} (Q) \}_{K_{rp}^{-1}} \Rightarrow \{Y, \{X, Q, K_{pq}^{-1}\}_{K_{rp}^{-1}}$

With this message R says, that K_{pq}^{-1} is P's private key for communicating with Q. The link between the private key and P is done implicitly, as above by encrypting the message with a key K_{rp}^{-1}, so that only P may decrypt it. The link between Q and P's private key for Q is explicitly provided in the data structure containing Q's name.

Derived Rules for Secret Key Cryptosystems CS-1:

In CS-1 we have one common secret key used by both partners. We get:

1_1) $R \vdash \{X, R \overset{K}{\leftrightarrow} Q \}_{K_{rq}} \Rightarrow \{X, R\}_{K_{rq}}$

Here, the "signature" is ambiguous. Both, R or Q could have produced and submitted the message. This may lead to protocol weaknesses, because message originator and recipient are not explicit. So, we add the originator's name to the data structure in the specification.

2_1) $R \vdash \{Y, P \overset{K}{\leftrightarrow} Q \}_{K_{rp}} \Rightarrow \{Y, Q, K_{pq}\}_{K_{rp}}$

Like in CS-2 we have to provide the link between the secret key and the partner entity by indicating its name explicitly.

Common Rules:

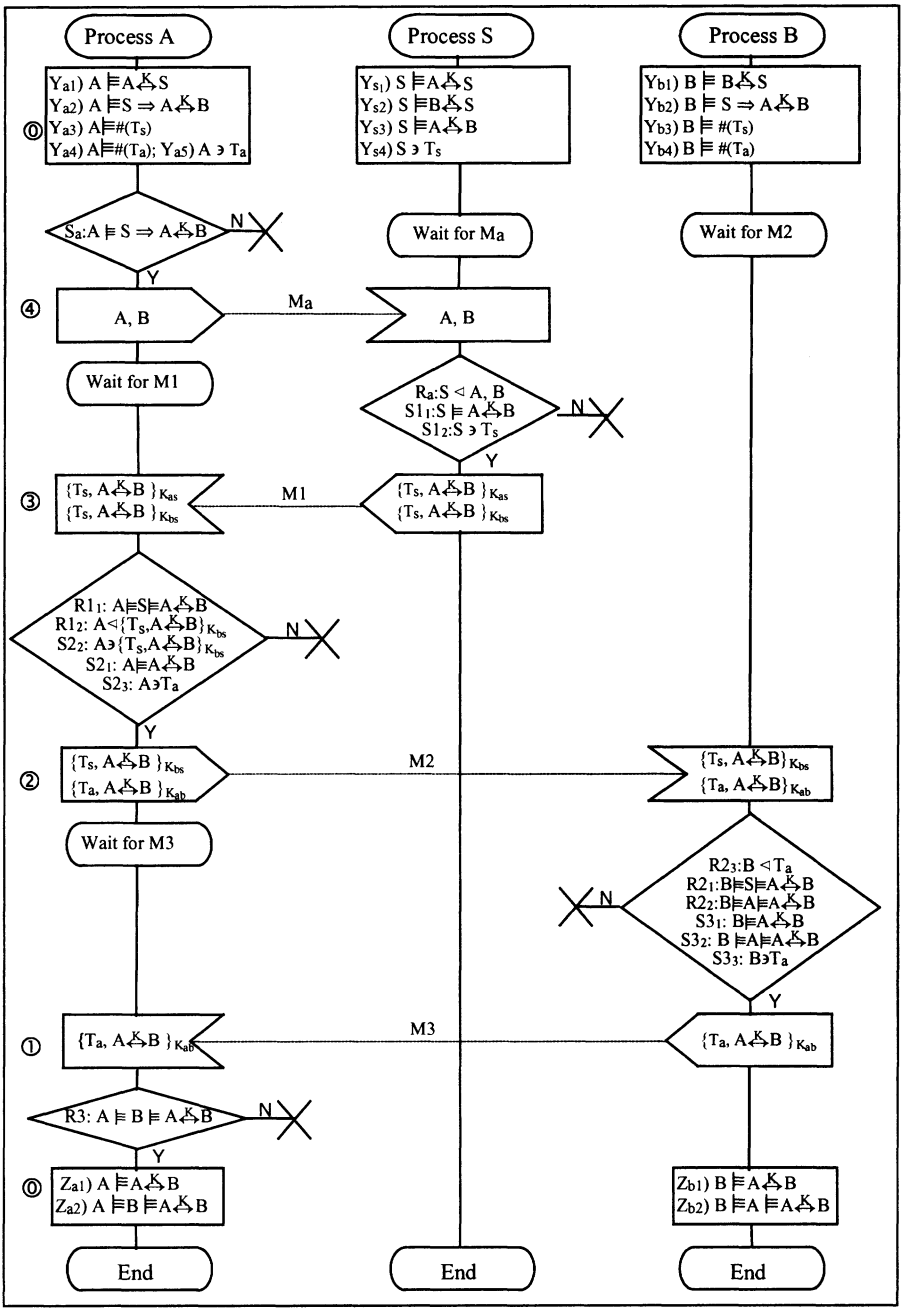
4) $A \Rightarrow A$

Cleartext expressions remain unchanged.

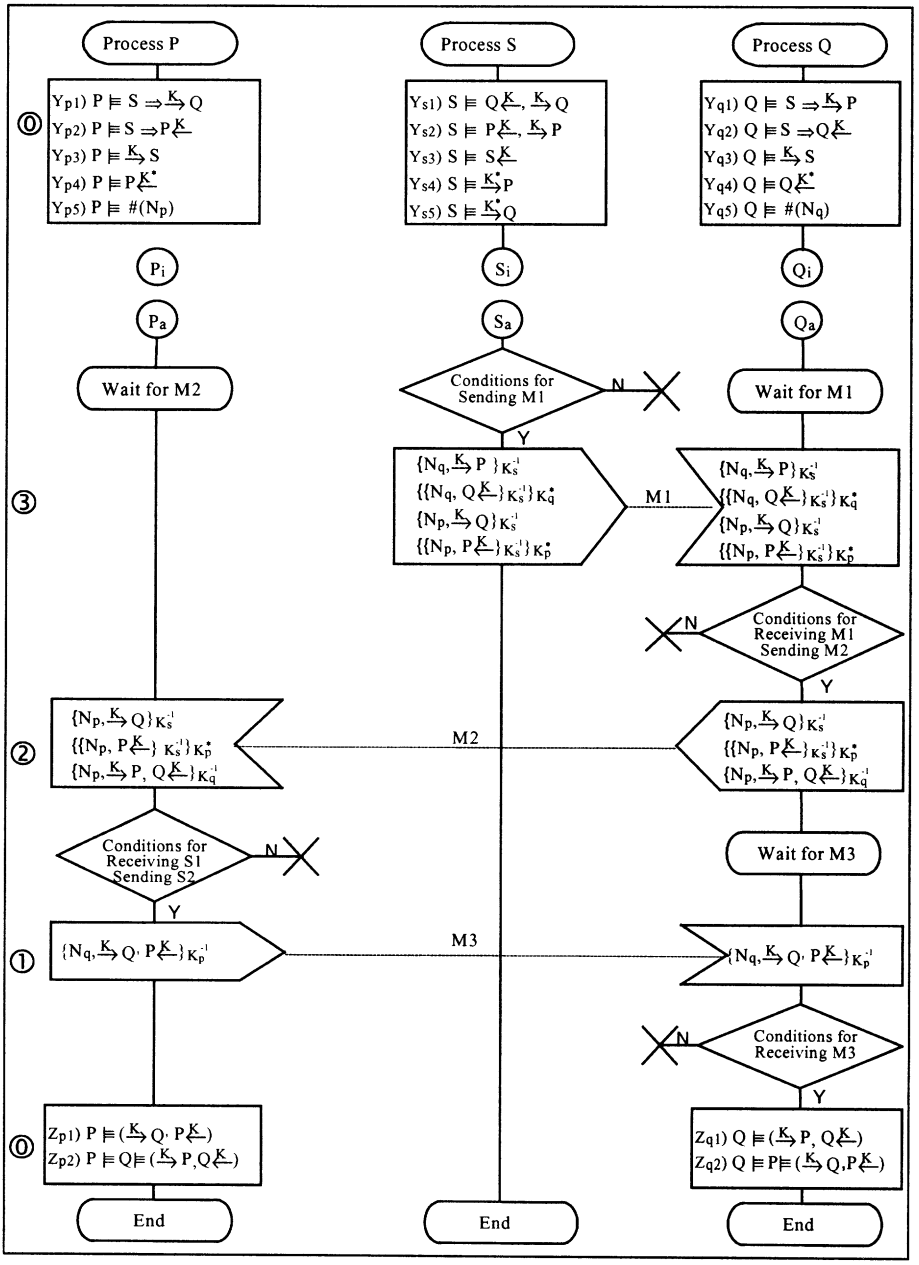
5) $\{T\}_K \Rightarrow \{T\}_K$

Encrypted expressions containing plain data like timestamps or nonces remain unchanged.

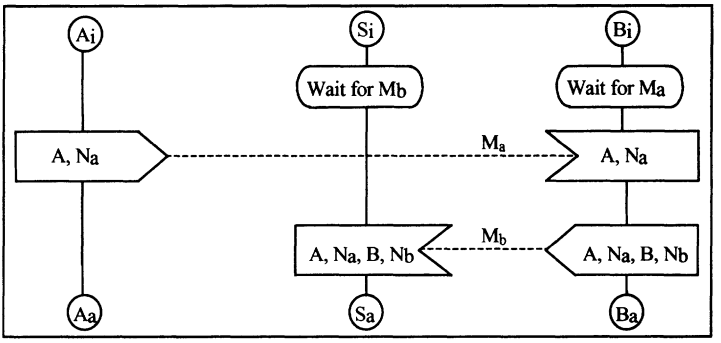
APPENDIX C: IDEALISED KERBEROS PROTOCOL



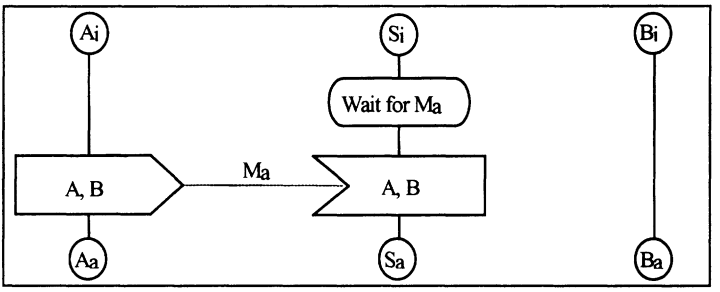
APPENDIX D: GENERIC, IDEALISED PROTOCOL FOR THREE PARTIES



APPENDIX E: SYNCHRONISATION PHASES FOR THREE PARTIES



Synchronisation using nonces



Synchronisation using timestamps

APPENDIX F: PROPERTIES OF BASIC CRYPTO-SYSTEMS

Crypto-system	Type of key used by Originator P	Type of key used by Recipient Q	Cryptogram	Proof of origin	Proof of original destination	Confi-dentiality
Symmetric Cryptosystem	Secret key	Secret key	$\{M\}_{K_{pq}}$	ambiguous	ambiguous	yes
Public Signature System	Private signature key	Public proving key	$\{M, Q\}_{K_p^{-1}}$	definite	definite[5]	no
Public Encryption System	Public encryption key	Private decryption key	$\{M\}_{K_q}$	no	no	yes
Private Cryptosystem	Private Key	(Inverse) Private Key	$\{M\}_{K_{pq}^{-1}}$	definite	definite	yes
One-way Function	Shared Secret	Shared Secret	$M,H(M,N_{pq})$	ambiguous	ambiguous	no

[5] For this, Q's name has to be provided explicitly in the cryptogram.