Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Verification of the Dynamic Authorization Protocol

Felipe Rodopoulos de Oliveira

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientadora
Prof.a Dr.a Cláudia Nalon

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Verification of the Dynamic Authorization Protocol

Felipe Rodopoulos de Oliveira

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Brasília, 26 de Julho de 2019

Dedicatória

Dedico este trabalho ao meu avô, Carlos Alberto, pois foi ele que, durante toda a minha infância, me instigou a explorar os mistérios e maravilhas do mundo e da natureza, através da ciência. Fui cativado pelas suas milhares de histórias e anedotas e, por isso, acredito que todo o fascínio que tenho pela beleza da ciência veio destes nostálgicos anos ao seu lado.

Também dedico este trabalho à todos aqueles que se aventuram e dão o sangue pela ciência no Brasil, pois em um país onde a ciência anda tão desacreditada, nos restam os ombros uns dos outros para que continuemos essa árdua caminhada.

Agradecimentos

Primeiramente, agradeço a minha família por todo o apoio e companheirismo no decorrer deste trabalho. Agradeço especialmente à minha mãe: uma mulher que, com muita perseverança e dedicação, me mostrou que todo o trabalho duro se paga. Obrigado por ser a mãe que você é.

Agradeço à minha orientadora, Cláudia Nalon, pela enorme ajuda durante todo este processo. Por vezes, com as palavras certas e o toque humano inato à sua pessoa, me instigou a continuar de cabeça erguida e não desistir nos momentos onde eu achava que toda essa experiência não era para mim. Agradeço o apoio nas conferências, palestras e apresentações e espero fazer por alguém o mesmo que você fez por mim.

Dedico um agradecimento especial aos meus colegas durante este mestrado, Daniella e Lucas. A partilha dos problemas e das inseguranças tende a unir as pessoas e deste apoio mútuo, surgiu uma amizade que eu cultivo com muito afeto dentro de mim. Que nossas anedotas continuem e a vida sempre sorria para vocês.

Agradeço a Annelise, quem me acompanhou por grande parte deste processo. O ombro amigo, a parceria e todo o carinho neste período foram únicos e muito importantes para mim.

Por fim, agradeço a todos os meus amigos e demais familiares, por manter minha sanidade e alegria de viver. Sem isso, nenhum trabalho valeria à pena e nenhum sucesso teria o mesmo sabor, pois não poderia ser compartilhado com vocês.

Resumo

Questões em segurança compõem grande parte dos desafios das soluções de *internet banking*. Diferentes protocolos de segurança foram projetados visando prover confiabilidade para transações bancárias *online*. O Protocolo de Autorização Dinâmica segue um esquema onde uma chave compartilhada entre o banco e o usuário é estabelecida para uso em futuras transações bancárias online. Em cada transação, o usuário é desafiado a recuperar e apresentar uma chave de transação única ao servidor, usando um *smartphone* como uma entidade externa de validação que escaneia um QR-code para testes de integridade. Dessa forma o usuário consegue realizar operações em um computador inseguro e validá-las em um canal *offline*. Entretanto, o protocolo não é formalmente verificado. Neste trabalho, utilizamos a teoria do Método Indutivo para formalizar e verificar as propriedades do protocolo. Uma extensão desta teoria foi necessário para modelar o funcionamento de *smartphones* e sua interação com os usuários. Parte das propriedades do protocolo foram validadas, enquanto alguns problemas relativos a privacidade e clareza nas mensagens foram identificados.

Palavras-chave: internet banking, protocolos de segurança, formalização de protocolos, Método Indutivo, prova de teoremas

Abstract

Security issues are one of the biggest internet banking challenges, as such systems are highly targeted by ciber criminals. Distinct security protocols were designed in order to provide reliability to online banking transactions. The Dynamic Authorization Protocol is an interesting approach, where a shared key is defined between the User and the Bank for use in future online banking transactions. For each transaction, the User is challenged to retrieve a unique transaction code, using a smartphone as a third-party validation entity, scanning a QR-code for integrity checks, for being able to carry operations in a untrusted computer. However, the protocol is not formally checked. In this work, we used the Inductive Method theory for formalizing and verifying the protocol and its properties. An extension to this theory was necessary in order to reason about the smartphones and their interaction with users. Some protocol properties were confirmed, although some issues with privacy and message clearness were identified.

Keywords: internet banking, security protocols, protocol formalization, Inductive Method, theorem proving

Contents

1 Introduction **1**

1.1 Motivation . . . 2

1.2 Results . . . 2

1.3 Content Outline . . . 3

2 Security Protocols **4**

2.1 Basic Notions . . . 4

2.2 Attacker Assumptions . . . 5

2.3 Cryptography and Key Management . . . 5

2.4 Notation . . . 6

2.5 Confidentiality . . . 7

2.6 Authentication . . . 8

2.7 Other Goals . . . 11

2.8 Attack Approaches . . . 12

3 The Inductive Method **14**

3.1 Formal Verification . . . 14

3.2 Method Introduction . . . 15

3.2.1 Isabelle . . . 16

3.2.2 Agents . . . 16

3.2.3 Cryptographic Keys . . . 17

3.2.4 Messages . . . 17

3.2.5 Events . . . 18

3.2.6 Threat Model . . . 19

3.2.7 Operators . . . 21

3.2.8 Protocol Model . . . 24

3.3 Goals Verification . . . 24

3.3.1 Reliability . . . 25

3.3.2 Regularity . . . 26

3.3.3 Confidentiality 26
3.3.4 Unicity 26
3.3.5 Authenticity 27
3.3.6 Authentication 27
3.3.7 Goal Availability 28

4 DAP: Dynamic Authentication Protocol **30**
4.1 Internet Banking 31
4.1.1 Security Issues 31
4.2 Motivation 32
4.3 Definitions 33
4.3.1 Initial Assumptions 33
4.3.2 General Model 34
4.4 Key Generation 34
4.4.1 Symmetric Scheme 35
4.4.2 Asymmetric Scheme 35
4.5 Message Transactions 37
4.6 Security Discussion 38

5 Offline Channel Formalization **40**
5.1 DAP Analysis 40
5.1.1 Smartphone Scope 41
5.1.2 DAP Assumptions *vs.* Real World Scenarios 42
5.2 Smartphone Formalization 43
5.2.1 Vulnerabilities 43
5.2.2 Keys 44
5.2.3 Usability 45
5.2.4 Events 46
5.2.5 Agents' Knowledge 47
5.3 Threat Model 49
5.4 Protocol Model 49
5.5 Further Discussion 50

6 Protocol Verification **51**
6.1 Message Transaction 51
6.1.1 Threats 55
6.2 Model Reliability 57
6.2.1 Server Guarantees 57

6.2.2 Smartphone Use 59
6.2.3 Smartphone Outputs 59
6.2.4 Smartphone Inputs 61
6.2.5 Regularity 62
6.3 Model Properties 63
6.3.1 Confidentiality and Privacy 63
6.3.2 Authenticity 63
6.3.3 Unicity 65
6.4 Discussion 65

7 Discussion and Conclusion **67**
7.1 Results 67
7.2 Challenges 68
7.3 Future Work 69

References **70**

Appendix **73**

A Smartphone Model Proof Script **74**
A.1 Smartphone Usage 74
A.2 Agents' Initial State 74
A.3 Shared Key Properties 75
A.4 Function *knows* 77
A.5 Nonce Lemmas 77
A.6 Tactics for possibility theorems 78

B Smartphone Events Proof Script **81**
B.1 Function *knows* 85
B.2 Function *used* 92

C Secure DAP Proof Script **94**
C.1 Protocol Model 94
C.2 Message Reception Lemmas 97
C.3 Smartphone Device Lemmas 98
C.4 Reliability Lemmas 99
C.4.1 Server Guarantees 99
C.4.2 Smartphone Usability Guarantees 101
C.4.3 Protocol termination 103

C.4.4 Scans Event Guarantees 103
C.4.5 Shows Event Guarantees 104
C.5 Spy Counterguarantees 107
C.6 Regularity Lemmas 109
C.7 Propeties Lemmas 109

List of Figures

2.1 Example protocol for notation understanding . . . 7

2.2 Naive session key establishment . . . 7

2.3 Session key establishment with proper confidentiality . . . 8

2.4 First attack possibility against the protocol described in Figure 2.3 . . . 9

2.5 Second attack possibility against the protocol in Figure 2.6 . . . 10

2.6 Session key establishment with proper confidentiality and authenticity . . . 10

2.7 A replay attack on the protocol described on Figure 2.6. Key K'_{AB} is obtained by the Spy from by eavesdropping previous protocol runs . . . 12

2.8 Session key establishment with proper confidentiality, authenticity and freshness . . . 12

3.1 Formal model for the protocol displayed in Figure 2.1 . . . 25

4.1 Diagram of a symmetric method for generation of key K . . . 36

4.2 Diagram of a message transaction in DAP. The protocol steps are referenced in the diagram . . . 39

6.1 Secure DAP message transaction base case and reception . . . 52

6.2 Secure DAP rule 1 and 2 formalizations . . . 52

6.3 Secure DAP rule 3 formalization . . . 53

6.4 Secure DAP rule 4 formalization . . . 54

6.5 Secure DAP rule 5 formalization . . . 54

6.6 Secure DAP rule 6 formalization . . . 55

6.7 Secure DAP rule 7 formalization . . . 55

6.8 Secure DAP rule 8 formalization . . . 56

6.9 Spy general behavior rule for Secure DAP . . . 56

6.10 The Spy exploit for rule *DT4* . . . 57

6.11 The Spy exploit for rule *DT6* . . . 57

List of Tables

2.1 Common types of protocols attacks 13

Chapter 1

Introduction

If computers were a broadly accepted tool for improving work and life, then computer networks were a turning point for human society. Finances, administrative and bureaucratic processes, commercial transactions, social interactions, everything seems to be slowly migrating to a computer system environment, not to mention that almost everything nowadays can be network connected: mobile phones, televisions and even home appliances, like a refrigerator. At a certain point, sensible data and critical information was being handled and, quickly, a new necessity emerged: security.

Internet banking is one of these areas, where sensible data, like monetary transactions and users credentials, is a desired target by malicious peers, who try to steal, manipulate or sniff on these data, potentially leading to great monetary or corporations' image damage. As a counter measure, researchers and corporate organizations try to develop ways for providing security for users' property and information through security protocols.

The *Dynamic Authentication Protocol* (DAP) [1] is a security protocol, developed in 2012, aiming to provide a trustful authorization method for each banking transaction on an out-of-band secure access model. The author claims that the protocol provides minimization of success to a vast field of known attacks and approaches. However, the cited document does not provide a formal and incontestable proof of the protocol correctness, only providing to the reader a study case of a real world situation.

Formal verification is the field of Computer Science where mathematical tools are used to proving or disproving the correctness of algorithms, software or hardware. In essence, this requires the translation of those systems and their properties into a formal language for further verification by formal methods. In contrast to informal methods, which tend to be less arduous, formal verification underlies in mathematical principles and provides reproducible proofs or counter-examples of a software correctness.

Like a regular computer program, security protocols are software, often involving concurrency, where general scenarios gather multiple peers communicating with each other.

Numerous techniques were developed for formally verify security protocols. The *Inductive Method*, first proposed by L. Paulson [2] and further developed by G. Bella [3] is an interesting system, which abstracts many details of a security protocol context in a manner where it is possible to reason about past executions in a formal and logical way.

In this work, the Inductive Method was explored and subject to test on DAP, a real world and challenging protocol, where the chosen area of interest shows significant relevance. Based in techniques from previous works [4, 5, 2], the protocol was formally specified and verified.

1.1 Motivation

Authentication is the biggest concern in security for internet banking [6]. Allied to confidentiality, these properties provides a reliable environment for users to perform monetary transactions without worrying with passive network eavesdroppers or active attackers trying to modify these transactions.

However, despite the efforts to provide desired security qualities for internet banking systems, criminals are still able to perform successful attacks on those infrastructures [7].

Promising works for ensuring security have appeared over years using OTP schemes [8, 9], the basis for DAP. Still, no formal verification was done in any of these approaches. Therefore, there is no incontestable guarantee that they will not fail. Allied with a set of automated tools, formal verification can provide mathematical proofs that security protocols are correct or not. For that reason, we argue that formally verifying such critical systems is not only a valid motivation, but an obligation.

1.2 Results

The work produced three main results. First, a reliable formalization of smartphones and its related events regarding data input and output. Using such model, the formalization of the DAP was possible. The formalization considered the scenario where devices are secured and could not be exploited. The obtained model was proved sound against the protocol, containing proofs about the operation of the bank server, peers and their smartphones operation.

Additionally, some of the protocol claimed properties were explored and further certified through formal reasoning. It was found that the protocol had issues with the banking transaction privacy. Also, from the user's viewpoint, some security properties could not be verified, due to encrypted components in messages that could not be verified by the User's personal computer.

1.3 Content Outline

This document structure is divided in the following way:

1. **Chapter 2** presents the general theory in security protocols, discussing its properties and goals;
2. **Chapter 3** begins with a brief introduction on formal verification and then describes our selected approach, the Inductive Method;
3. **Chapter 4** outlines internet banking systems and its challenges for presenting our targeted protocol, DAP, detailing its models, assumptions and operation;
4. **Chapter 5** presents the formal model built for reasoning about smartphones and its operation. Also, it discusses other extensions done in the Inductive Method in order to provide a reliable model for the protocol formalization itself;
5. **Chapter 6** focus on the formalization and verification of the DAP itself. The security premises presented in the original specification of the DAP are used, following a more cautious approach. Many results are presented in this chapter;
6. **Chapter 7** concludes this document, reviewing the results and discussing challenges and future work

Chapter 2

Security Protocols

This chapter surveys the main aspects of security protocols, explaining its common properties, goals, and frequent attack methods. A basic notation, which will be used along all this document, will be presented as well.

2.1 Basic Notions

Security protocols are handy tools for providing protection among communication protocols and systems. They can be informally defined as a set of steps executed between multiple entities, aiming to ensure a reliable and secure communication between them. Current protocol specifications vary in quality, purpose and syntax, although they are the base documents for formally reasoning about these protocols [10]. More definitions on security protocols can be obtained at [11] and [12].

As a communication agreement, security protocols are inherently concurrent, leading to a large spectrum of possibilities among its steps, peers behavior and system states. When considering the attacker factor — an agent who can both act as a legal participant or produce and inject fake information to exploit the protocol — the difficulty in designing a reliable solution can grow dramatically. For the very same reason, it is difficult to design a good security protocol and easy to develop a defective one [3].

Protocol goals depend on the environment, resources and the protocol architecture itself and each situation asks for stronger guarantees in some properties than others. Regarding this, some of those goals may have varying definitions in the literature, like authentication [10]. Nevertheless, some properties and concepts can be universal among security protocols, as well as some goals are a consensus in the literature [12].

2.2 Attacker Assumptions

A basic assumption for every entity engaged in a security protocol session is that the knowledge about the environment is uncertain. In other words, agents will interact with other participants, which can be hostile or not, and the communication channel is generally untrusted. Moreover, it is wise to expect the worse, since security properties are relative to the resource of attackers [11]. It is common to provide guarantees assuming that attackers could perform unlikely achievements, like obtaining session keys or acting as a legal peer. This can improve the system resilience.

A suitable approach for attacker modeling is the classic work of Dolev-Yao [13]. In this model, the attacker is described in a general way, giving her full control of the network and protocol operation. Therefore, it is interesting to model her capabilities in the following assumptions, based on [11]:

1. The adversary is able to eavesdrop on all messages sent on the network;
2. The adversary is able to alter any message captured during protocols sessions, using any information available. Also, it can re-route to other peers and create new messages at any time;
3. The adversary may be a legal participant, an outsider or both;
4. The adversary can decipher or obtain information from a current session combining pieces of data from previous sessions.

2.3 Cryptography and Key Management

Despite being a crucial basis on the success of a reliable protocol, cryptography will be treated in an abstract way here, since what matters are its concepts and its impact on the protocol behavior. Cryptographic keys are the basis for generating secure communication, where it is used for enciphering all data transmitted along the channels. However, creating a valid new session key demands a previous secured channel. Indeed, the establishment of such key may occur in one of these three contexts:

1. The peers have a pre-shared key, already created;
2. An off-line public infrastructure may be used, where the peers hold certified public keys;
3. An on-line public infrastructure may be used, where the peers share a key with a trusted third-party entity.

Concerning the user entity and based on these concepts, the generation of session keys can be based on a *key transport* or *key agreement* protocol. The first is associated with protocols where participants use on-line servers, which generates keys and transfers them to users. The second is often associated with off-line infrastructures, which generates the key based on inputs provided by the peers. Hybrid protocols for key generation based on these two approaches also exist [11].

2.4 Notation

Through this document, a consistent notation for expressing protocol actions and symbols will be used. A system based on the one presented in [3] will also be used, which brings aspects from the seminal work of [14], since it is our main reference method for verification in this work.

Primitive data instances, such messages, timestamps and *nonces* can be represented as simple mnemonic letters and the encryption of such entities are represented with subscripts, for example, a message M using a key K is represented by M_K. When giving authorship to a given resource, we also use subscripts, such a key from agent A resulting in the symbol K_A. Accordingly, the session key established between agents A and B is represented by the symbol K_{AB}. On following examples and proofs, the definitions will be straightforward.

Concerning the key scope, a distinction is also required. For that, private keys are preceded by the indicator s, meaning signature, and public keys are preceded by p, meaning public. Hence, the private and public keys of peer A would be represented as sK_A and pK_A, respectively.

Besides, the use of fat brackets ($\{\!|$ and $|\!\}$) has two purposes: it can distinguish protocol messages from sets and represents concatenation. Therefore, a concatenation of messages m and n is represented as $\{\!|m, n|\!\}$ and its encryption under key K is written as $\{\!|m, n|\!\}_K$.

Finally, we need to represent message transportation. The syntax follows a simple structure, where each protocol step is represented in one line, following the order: step number, sender, recipient and finally the message contents, which contains at least one symbol. For instance, we use the hypothetical protocol provided in Figure 2.1, where the host A sends its identity and a nonce N_A to a host B, who replies with the nonce and a fresh session key K_{AB}, both encrypted under its private signature key sK_B.

As a background example intended to provide better understanding of the next concepts, the problems within some protocol designs and the syntax presented in this section, we describe a common situation, where two agents, A and B (commonly called Alice and Bob), want to communicate securely. For that, they generate a session key K_{AB}, aided

$$\begin{array}{lll} 1. & A \longrightarrow B & : \; A, N_A \\ 2. & B \longrightarrow A & : \; \{\!|N_A, K_{AB}|\!\}_{K_B} \end{array}$$

Figure 2.1: Example protocol for notation understanding

by a trusted third-party peer, the Server S. Such key will be used for ciphering future messages, so it must be newly generated and only known to these three entities.

In Figure 2.2, a protocol is designed as a first and naive attempt to represent this situation. At first, Alice sends to the server her and Bob identities, who intend to communicate with each other. The Server replies to her the session key K_{AB}, which Alice readily forwards to Bob, along with her identity. Over the next sections, we will identify problems and redesign the protocol.

$$\begin{array}{lll} 1. & A \longrightarrow S & : \; A, B \\ 2. & S \longrightarrow A & : \; K_{AB} \\ 3. & A \longrightarrow B & : \; K_{AB}, A \end{array}$$

Figure 2.2: Naive session key establishment

2.5 Confidentiality

Confidentiality (or secrecy) is a required aspect when systems should not leak information to untrusted peers, guaranteeing access only for the trusted ones. Also, it is one of the most desirable properties among security protocols. As seen previously, cryptography is a suitable method for generating keys used for securing messages. We define a encryption scheme consisting of a key set $\mathcal{K}$, a message set $\mathcal{M}$ and a cyphertext set $\mathcal{C}$ and three main algorithms:

- **Key Generation:** outputting an encryption key $K \in \mathcal{K}$ and decryption key $K^{-1} \in \mathcal{K}$;
- **Encryption:** taking a message $m \in \mathcal{M}$ and encryption key $K \in \mathcal{K}$, it outputs the cyphertext $c \in \mathcal{C}$, which is defined as $c = m_K$;
- **Decryption:** taking a cipher-text $c \in \mathcal{C}$ and decryption key $K^{-1} \in \mathcal{K}$, it outputs the message $m \in \mathcal{M}$, which is defined as $m = D_{K^{-1}}(c)$.

Considering key properties, when a given encryption key K is the same as the decryption key K^{-1}, that is $K = K^{-1}$, it is stated that they are *symmetric* keys. Otherwise, they are *non-symmetric* keys, which are normally engaged in a process of public key encryption, commonly used nowadays [12].

Secrecy consistency can be discussed with respect to some degree of flexibility. An active spy should not be able to learn something on generated cyphertext. The property of *semantic security* states that if an attacker can compute something from a cipher-text, it must also be able to compute it from its equivalent message. Stronger than that, *non-malleability* makes impossible to transform a given cyphertext in a related one without knowing the input plain text. A detailed discussion on this topic is provided in [10].

Considering our previously presented example given in Figure 2.2, it is easy to show that confidentiality is compromised in that protocol. The session key K_{AB} is transmitted in clear text over the network, so an active spy could simply extract it from the channel and use it for decipher subsequent messages exchanged between A and B.

We redesign our first attempt of securing the protocol, given in Figure 2.3. Here, we define keys K_{AS} and K_{BS} as previously shared keys known by the server and agents A and B, respectively. Alice will repeat the first step of the former protocol, but now the server will reply with a tuple, composed by the session key K_{AB} encrypted with shared keys K_{AS} and K_{BS}. Finally, Alice sends the session key, encrypted with Bob shared key, and her identity to Bob.

$$\begin{array}{lrcl}
1. & A \longrightarrow S & : & A, B \\
2. & S \longrightarrow A & : & \{\!|K_{AB}|\!\}_{K_{AS}}, \{\!|K_{AB}|\!\}_{K_{BS}} \\
3. & A \longrightarrow B & : & \{\!|K_{AB}|\!\}_{K_{BS}}, A
\end{array}$$

Figure 2.3: Session key establishment with proper confidentiality

If the agents are not compromised, then the spy cannot retrieve the session key K_{AB}, since she does not know the shared keys from A and B. Consequently, the communication between Alice and Bob is protected.

2.6 Authentication

Authentication is a vastly studied topic among security researchers, since its definition is not really well-established, causing problems for its correct implementation of this property among protocols. Such topic is discussed at length in [15].

Here, we recall some definitions provided in [16] and [12]. Authentication can be defined as the intention to initiate a communication session, where peers can be assured about each others identity. Similarly, the establishment of trust in a peer identity, can be related to the generation of a session key for further communication among the peers involved in the session, validating each message.

Concerning the authentication of origin, confirming the authorship of A of a given message received by B, in a given protocol session, gives some guarantees. The first one is that A is alive and she must have sent the message. Further, a stronger property claims that she truly intended to communicate with B, agreeing in following the protocol directives.

Also, guaranteeing the authorship of a message is guaranteeing its integrity. For this reason, some authors [3] considers authentication and integrity can be considered equivalent. In order to ensure message integrity, one can use manipulation detection codes (MDC) or message authentication codes (MAC) [17].

Other stronger properties may be needed. One is freshness, which is the necessity of a received message to be the first one, avoiding the acceptance of repeated messages. Concerning the protocol which will be reviewed in Chapter 4, this notion is crucial. A good discussion on this is also given in [15].

Now we focus on our working example. Figures 2.4 and 2.5 presents a diagram where agents are represented as nodes and messages are described above edged, preceded by they order. As stated, the attacker can intercept and modify messages, since she has full control of the network.

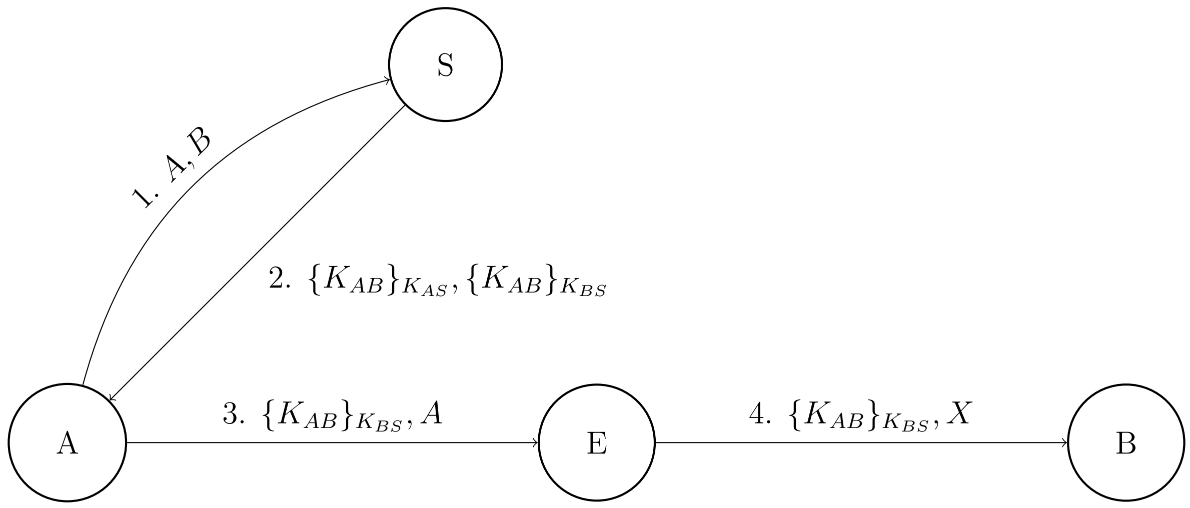
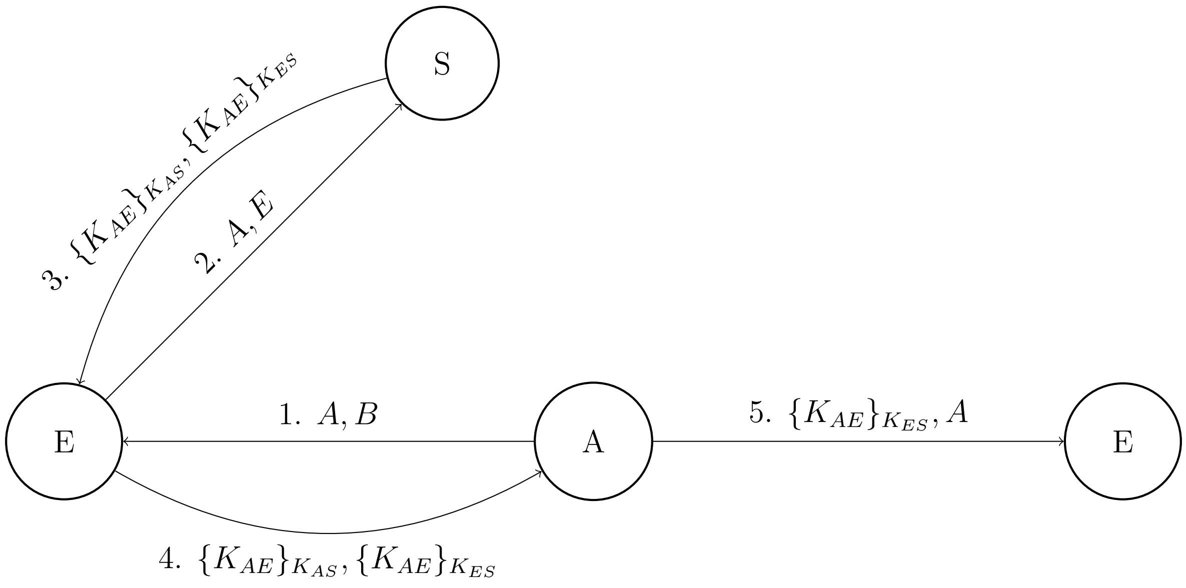

S
1. A,B
2. $\{K_{AB}\}_{K_{AS}}, \{K_{AB}\}_{K_{BS}}$
A
3. $\{K_{AB}\}_{K_{BS}}, A$
E
4. $\{K_{AB}\}_{K_{BS}}, X$
B

Figure 2.4: First attack possibility against the protocol described in Figure 2.3

In the first case, the Spy intercepts the message containing the session key and A's identity, replacing A's identity for another identity X, and sending it to B. Here, X could be any agent identity, including the spy itself, and the security failure consists exactly in the fact that the guarantee of knowing who is sending a message does not hold, even though the spy does not know the session key K_{AB}.

The second case presents a more serious flaw. Here, the Spy intercepts the first message from A, replacing B's identity by its own, and sends it to the server. Therefore, the server will provide a session key K_{AE}, destined for communication between A and E. Additionally, the spy will also impersonate B, receiving the session key K_{AE} and A's identity, establishing a legal and ciphered communication channel with A. The problem here relies in the fact that A supposes that such communication is being held with B, which is a false statement. So, not only the authentication of origin, but the session key is also compromised.



S
3. $\{K_{AE}\}_{K_{AS}}, \{K_{AE}\}_{K_{ES}}$
2. A, E
E
1. A, B
A
5. $\{K_{AE}\}_{K_{ES}}, A$
E
4. $\{K_{AE}\}_{K_{AS}}, \{K_{AE}\}_{K_{ES}}$

Figure 2.5: Second attack possibility against the protocol in Figure 2.6

In order to deal with the aforementioned problems, we review our approach, redefining the protocol, which is given in Figure 2.6. Now, the session key is linked with the identity of its owners, since the server replies Alice with two tuples: the session key and the identity of the participants, encrypted with their shared key. At the end, Alice forwards her identity, together with K_{AB}, to B. Note that now, the spy cannot replace any identity, since it is also encrypted and thus, authentication is preserved.

$$\begin{array}{lll}
1. & A \longrightarrow S & : \; A, B \\
2. & S \longrightarrow A & : \; \{\!|K_{AB}, B|\!\}_{K_{AS}}, \{\!|K_{AB}, A|\!\}_{K_{BS}} \\
3. & A \longrightarrow B & : \; \{\!|K_{AB}, A|\!\}_{K_{BS}}
\end{array}$$

Figure 2.6: Session key establishment with proper confidentiality and authenticity

2.7 Other Goals

In the previous section we have presented the two most desirable goals in security protocols, but it is important to mention other properties that are also cited in literature. In the following, we give brief descriptions of such properties. We opted for a non-exhaustive description, as those properties are more loosely used among protocols, even though their importance is uprising.

Non-repudiation is the ability of providing evidence on peers actions. Such property differs from authentication, where the former tries to provide an identity assurance to the system and the latter produces a formal and enduring proof of authorship on actions and messages, which can be checked by participants and third-parties. Consequently, the author cannot try to deny such actions. Notice that non-repudiation acts like a defense against not only outsiders, but from legitimate users as well.

Unicity or **freshness** is a protection against replay attacks, by providing a way to state that a message or key is new and not a replayed one from previous sessions. It is a common protocol requirement, since attacks that take advantage of the knowledge on old messages are increasing. At the end of this section, we will extend our working example for providing freshness of the keys.

Even if it is not the focus of this work, **availability** is considered as a desired goal in protocols, since it is crucial for systems properly maintain ongoing sessions. Not only keeping a peer available is important, but guaranteeing a session key or message reception is also a concern. Thus, modeling the attacker ability of killing protocol messages and reliable defenses against denial of service attacks is a valid concern among protocol designers. At the same time, the notion of shared resources between protocol sessions can be considered, but it will not be explored in this work.

Based on common errors found in published protocols, the work in [18] presents interesting guidelines for designing reliable security protocols. Although these guidelines are not sufficient for guaranteeing correctness, the document offers examples of protocols that do not follow the displayed principles and fails to fulfill its goals, arguing that such principles are indeed a convenient starting point for formulating protocols.

We now do a final analysis of the working example, considering the freshness property. A possible attack, extracted from [11], is represented in Figure 2.7, where the Spy impersonates the server and follow the protocol structure, but instead of using a new session key K_{AB}, she uses an old K'_{AB} key, used in past protocol runs. Even if the spy does not know the value of K'_{AB}, she can replay previous messages encrypted with K'_{AB} and gather more data for post cryptanalysis.

An attempt for fixing such flaw is the classical protocol of Needham-Schroeder [19], which uses nonces, ciphered with shared keys, for guaranteeing freshness of the session

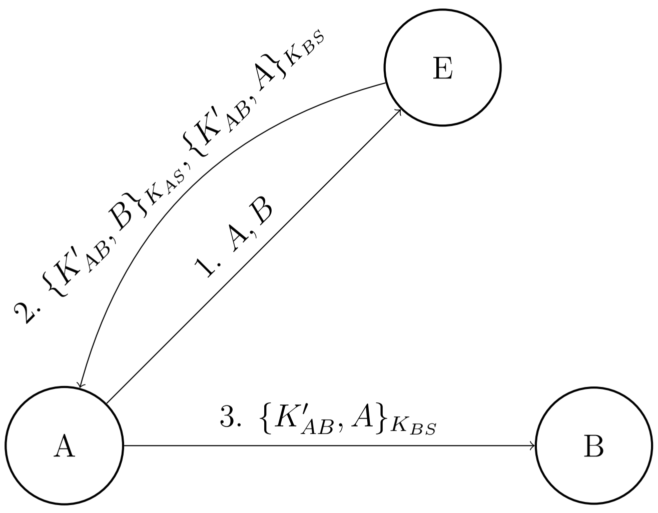

E
2. $\{K'_{AB}, B\}_{K_{AS}}, \{K'_{AB}, A\}_{K_{BS}}$
1. A, B
A
3. $\{K'_{AB}, A\}_{K_{BS}}$
B

Figure 2.7: A replay attack on the protocol described on Figure 2.6. Key K'_{AB} is obtained by the Spy from by eavesdropping previous protocol runs

key. However, this approach was proven to contain failures [20]. That said, we will focus on a reliable solution, also described in [11] and illustrated in Figure 2.8.

$$\begin{array}{lrcl}
1. & B \longrightarrow A & : & B, N_B \\
2. & A \longrightarrow S & : & A, B, N_A, N_B \\
3. & S \longrightarrow A & : & \{\!|K_{AB}, B, N_A|\!\}_{K_{AS}}, \{\!|K_{AB}, A, N_B|\!\}_{K_{BS}} \\
4. & A \longrightarrow B & : & \{\!|K_{AB}, A, N_B|\!\}_{K_{BS}}
\end{array}$$

Figure 2.8: Session key establishment with proper confidentiality, authenticity and freshness

The final solution relies in two fresh nonces, N_A and N_B. Now, B will start the protocol, sending her identity and nonce to A. A sends hers and B's identities and nonces to the Server, who responds A with K_{AB}, B identity and N_A encrypted with A's shared key with the Server and an equivalent instance, encrypted with B's shared key K_{BS}. Finally, Alice sends B's instance to her, properly distributing the session key K_{AB}. The nonces allows the agents to check freshness of messages and its contents, the keys protect the integrity of data exchanged among peers, and the protocol is secure.

2.8 Attack Approaches

Complementing the notes on security protocols, it is interesting to provide a section about the classical and most common attack approaches studied in the literature, aiming

Table 2.1: Common types of protocols attacks

Eavesdropping	The adversary capture the data sent in the protocol
Modification	The adversary alters the data sent in the protocol
Replay	The adversary record data seen in previous protocol sessions and reuses it in a different, usually later, session for exploitation
Preplay	The adversary engages in a run of the procotol prior to a run by the legitimate principals
Reflection	The adversary sends protocol messages back to the principal who sent them
Denial of Service	The adversary prevents or hinders legitimate messages or principals from completing the protocol
Typing Attacks	The adversary replaces a protocol message fields of on type with a message field of another type
Cryptanalysis	The adversary gains some useful leverage from the protocol to help in cryptanalysis
Certificate Modification	The adversary chooses or modifies certificate data to attack one or more protocol runs
Protocol Interaction	The adversary chooses a new protocol to interact with a known protocol

in understanding how such threats scaled up. A non-exhaustive list, adapted from [11], is presented in Table 2.1, introducing its names and a brief explanation of its strategy.

It is essential to note that this list is not a complete guide, but an attempt to picture the mainstream knowledge on attack methods. Many attacks against systems and protocols involve combinations of several of these approaches and not all of them are applicable to all protocols. Different protocols have different purposes, therefore, the strategies for attacking them may vary.

Chapter 3

The Inductive Method

This chapter shows a brief introduction on formal verification and a more detailed explanation on the selected approach to be used in this work: the Inductive Method. The concepts will be restricted to the necessary elements needed for comprehension of the proposed problem.

3.1 Formal Verification

Using the previous definitions for security protocols, we note that one can reason about protocol rules, *verifying* if they are correct with respect to the protocol's goals.

Informal reasoning on verifying protocols was the first attempt to guarantee their correctness, succeeding in recognizing some flaws and weaknesses quickly and providing a good understanding about protocols design [3]. However, not finding an error does not mean that a certain model does not hold one, since informal methods may fail to identify critical flaws in major security protocols.

A classical example is the Needham-Schröeder protocol [19]. Despite the authors efforts on its specification and informal verification, security flaws were found using formal techniques based on Lowe's work [20]. This was an optimal context for the rise of formal methods for verifying security protocols [21, 22, 23, 14], providing a mathematical approach for reasoning about abstract protocols models.

Formal verification requires the definition of such protocols in a proper language, suitable for the application of specific methods and, hopefully, the use of automated reasoning tools, improving proof methodology. Also, formal methods can be divided in two major categories, according to [11]:

- **Model checking**: protocol behavior can be modeled as a finite set of states. Such proposal is suitable for checking if a given configuration satisfy a set of correctness conditions and analyze a certain past configuration, searching for attacks attempts. This approach is more appropriate for finding inconsistencies and attacks in the protocol rather than proving some of its correctness properties;

- **Theorem proving**: the protocol can described as a model, where *all* possible behavior are considered. Thus, any wanted property must be described as a theorems and proved using the axioms stated by the model. This method is more suitable for proving the correctness and properties of the protocol rather than finding possible attacks on them.

3.2 Method Introduction

The Inductive Method is a proof theory for modeling and reasoning about security protocol, which aims for both the formalization and the verification of such protocols, using model checking and structural induction. First, the intended protocol $\mathcal{P}$ must be defined as an inductively constructed model P, composed by rules describing the real world steps of $\mathcal{P}$.

An infinitely countable set of agents is considered, who can fire events accordingly with the model P, in any desired order or regularity. Agents can interleave between protocol sessions with their actions indefinitely. Among the agents, we stress a special one, the Spy, who has control over the network and compromised agents, it is used to model the threat scenario.

At a second stage, the model is verified towards proposed properties, reasoning along technical and theoretical traits. Here, structural induction takes a crucial role. If a security property must hold for the protocol, it must hold for all possible traces derived from the formal model P, hence induction over the set of possible network traces is the main proof tool used in this phase.

Induction has been previously used for the formalization of protocol, for example in the NRL Protocol Analyzar by C. Meadows [21], and the Inductive Method has similarities with other model checking theories, like CSP [12]. It comprehends a strong threat model and concurrency among peers, however it does not check liveness properties. Moreover, the framework on which it is built provides easy ways for extension, allowing steady formalizations of atypical systems, a feature which interests us.

The method was first proposed by L. Paulson [2] and extended by G. Bella [3]. It has been used for the verification of many deployed and significant protocols [5, 4, 24].

3.2.1 Isabelle

Given the considerably great number of possible agents and possible traces which can be derived from a model, correctness proofs can take a substantial amount of work. As a result, the use of an automatic theorem prover is desirable. Isabelle [25] is one of such systems, where the theory framework is built on, providing machine verifiable and maintainable proofs scripts.

Isabelle is a generic theorem prover, which can reason over several formal systems, in an interactive way. Hence, proofs are not entirely automatic, requiring certain user guidance. It combines high order logic, typed formalism and quantifiers for functions, predicates and sets. Also, the system has a good range of proof tools: simplifiers, induction-oriented commands and access to some automatic provers and state-of-the-art theories.

Lemmas and theorems are stated as goals, to which the user must apply proof tactics, being able to use previously proved lemmas. Such process may derive subgoals, which also need to be dealt with. Hence, the proof process is arduous and must be done with care. If an user fails to find a proof for a certain property, this may not be a certification of the nonexistence of a proof. At the same time, the proof or some of its aspects may be built on top of wrong formalizations, meaning the user is not skilled enough.

Each formalized theory in Isabelle is contained in a proof script. Also, each Isabelle distribution comes with a library of such formalizations. The Inductive Method framework is defined in the *Auth* library [26], comprised in three files: `Message.thy`, `Event.thy` and `Public.thy`. The content of those files will be depicted over the rest of this chapter, where some of Isabelle syntax will be presented, when convenient.

3.2.2 Agents

Agents are described in the `Message.thy` file, being the basic type for specifying participants in a protocol session. Three main entities are defined as free types: legal agents (Friends), the Spy and the Server. In the definition below, the ≜ symbol reads as a definition equality operator for a keyword and the | symbol is the disjunction operator, separating the possible types for the `agent` datatype.

```
datatype agent ≜ Server | Spy | Friend nat
```

The set of legal agents has a correspondence with the set of natural numbers. Thus, we have both an easy mapping for each participant of a protocol session and removal of limitations of its population size, providing the ability to reason over protocols with an indefinite and infinite number of peers.

The nullary constructor `Server` defines the trusted third part entity, presented in many protocols. It is considered uncompromisable and holds the long-term secrets (keys) of all agents. The Spy is the malicious agent, but who can also act as a legal one. She has access to the secrets of all compromised agents and her own secrets.

3.2.3 Cryptographic Keys

Also defined in `Message.thy`, cryptographic keys are bounded to the natural numbers, but constrained within a proper set. Later, each type of key is normally defined as a relation between the sets of agents and keys. For instance, the habitual specification of shared keys is defined as follows, where the $\longrightarrow$ symbol defines the relation between the type sets.

$$shrK\colon\ \texttt{agent} \longrightarrow \texttt{key}$$

The description of public and private keys structures uses a similar construction. The set `symKeys` helps in the distinction between of long-term keys and session keys. Both types are considered shared keys, but the latter is commonly a fresh entity generated at protocol runtime and distributed among peers. Below, `range` is a function which gives us the image of a given function.

$$K \in \texttt{symKeys} \text{ and } K \notin \texttt{range}\ shrK$$

Finally, the function *invKey* receives a key and returns its inverse key, which can decipher any cipher created by the former. If it is applied to a symmetric key, it return the same key, while for asymmetric keys, the respective half is returned. Any other kind of keys not mentioned here must be manually defined.

3.2.4 Messages

A message is any kind of information that can be transmitted during the execution of a protocol. This includes agents names, keys, nonces, timestamps, and so on. Therefore, its constructor accepts many kinds of formats, as stated below:

$$\begin{aligned} \texttt{datatype} \triangleq\ & \textbf{Agent}\ \texttt{agent}| \\ & \textbf{Nonce}\ \texttt{nat}| \\ & \textbf{Key}\ \texttt{key}| \\ & \textbf{Mpair}\ \texttt{msg msg}| \\ & \textbf{Hash}\ \texttt{msg}| \\ & \textbf{Crypt}\ \texttt{key msg} \end{aligned}$$

Some of the accepted datatypes are constructors and others are simply natural numbers. If any other datatype is needed, the definition of this scope must be extended.

Another important concept is the `Crypt` directive. It receives a message M and a key K and produces the corresponding cipher, i.e. `Crypt` M K. In this model, encryption is perfect and collision-free, so a message is only accessible within a cipher by a peer if she has the proper key.

Some protocol formalizations demand new types of message, since security protocols often resort on numeric entities for soundness of some properties. Significant ones are described below:

Nonces are big random natural numbers which are unguessable to any agent, including the Spy.

Guessable numbers are natural numbers, often used to model message option fields, time properties and similar concepts. As a result, they are always known by the Spy.

Timestamps are defined on top of the length of a trace, instead of relying on classical time units. Its definition and properties will be further explored in the sections below.

3.2.5 Events

Events, defined in the theory `Event.thy`, are the basic units that compose network traces. There are three main events, which are suitable for the analyzed protocol in this work, although there are more available for others protocols:

```
datatype ≜ Says agent agent msg
           Notes agent msg
           Gets agent msg
```

The action of `Says` is straightforward and it follows easily from its syntax: the first agent is the sender, the second is the intended receiver and the third parameter is the message which will be sent.

Both events `Notes` and `Gets` are related to information acquisition by an agent. The former is the literal action of an agent obtaining and storing a given message, enabling them to explore messages contents, deriving new information, and the Spy on collecting data from the network. The latter event denotes message reception by an agent, but without enrichment of agent's knowledge. It was a late extension of the Inductive Method [3, Ch. 8], in order to explore an agents' knowledge set of messages. Without this, agents' knowledge would be constrained to past events only. Additionally, the creation of such event, introduces the concept of *reception invariant*, demanding that `Gets` events are preceded by a `Says` event. This guarantee must be enforced by the protocol model.

A *trace* can now be defined: a list of network events occurring while an unbounded population of agents are running the protocol. Specifically, the trace is a list of such events, disposed in reverse chronological order, since events are added at the list head. Hence, traces may have many configurations, but must stay faithful to the protocol model. The set of possible traces of a given protocol characterizes its formal model.

We can now recall some aspects concerning timestamps. In the Inductive Method, a timestamp describes the moment where an event happened in a trace, using the trace length as the measure. Therefore, for generating a timestamp, the following function is used:

$$\texttt{CT:}\ \ \texttt{event list} \longrightarrow \texttt{nat}$$

Thus, the function receives a trace and returns a natural number, which will be the length of that trace, leading to the following definition, where `length` trivially denotes the function that returns the length of a trace.

$$\texttt{CT}\ evs \triangleq \texttt{length}\ evs$$

Precisely, the timestamp for a trace which has n events will be n and, consequently, an event happening at that moment will receive a timestamp value of n. Note that such definition does not allow that two distinct events have the same timestamp, eliminating concurrency among events in the same trace. Further, this concept is properly guaranteed by the existence of two distinct traces, where such two events happens at switched positions, defining two corresponding sequential approximations.

3.2.6 Threat Model

The standard thread model used in the Inductive Method is based upon the Dolev-Yao [13] approach, introducing three main characteristics concerning the Spy:

1. *The Spy is a legitimate agent:* since she can act as a legal agent, she has their same features, such as shared keys with the Server, public asymmetric keys, etc.;

2. *The Spy controls the network traffic:* this copes with the capability of the Spy on monitoring and obtaining messages sent on network channel and preventing the delivery or redirecting messages;

3. *The Spy can perform any message operation, except cryptanalysis:* with this abilities, the Spy can break, compose and modify messages on-the-fly, being able to alter legal messages or create fake ones. There is only one exception, she must hold the correspondent key to a cipher in order to obtain the plain text, which guarantees that any encryption is perfect.

The Inductive Method implements all of these three aspects. The first one is already assured by the definitions of agents and cryptographic keys. Moreover, the protocols models should enable the Spy to participate as a legal action along their runs.

The set `bad` contains all agents which are compromised by the Spy. These agents disclose all their secrets to the malicious peer, both prior and during the protocol run. The Spy herself is also included in this set. Secrets are contained in the agent's knowledge set, which is a crucial concept for meeting the second requirement of the threat model.

Function *initState* describes the peer knowledge set prior to any protocol interaction, defined as follows.

$$\mathit{initState}\colon\ \texttt{agent} \longrightarrow \texttt{msg set}$$

1. The Server initially knows all shared and public keys and its own private keys;

$$\begin{aligned}\mathit{initState}\ \texttt{Server} \triangleq{} & (\texttt{Key range}\ \mathit{shrK}) \cup (\texttt{Key range}\ \mathit{pubK}) \cup \\ & \{\texttt{Key}\ (\mathit{priK}\ \texttt{Server})\}\end{aligned}$$

2. Each legitimate agent initially knows its own shared and private keys and all public keys;

$$\begin{aligned}\mathit{initState}(\texttt{Friend}\ i) \triangleq{} & \{\texttt{Key}\ (\mathit{shrK}\ (\texttt{Friend}\ i))\} \cup \\ & \{\texttt{Key}\ (\mathit{priK}\ (\texttt{Friend}\ i))\} \cup \\ & (\texttt{Key range}\ \mathit{pubK})\end{aligned}$$

3. The Spy knows all secrets from a compromised agent, which includes herself. Additionally, she also knows all public keys.

$$\mathit{initState}\ \texttt{Spy} \triangleq (\texttt{Key}\ \mathit{shrK}\ \texttt{bad}) \cup (\texttt{Key}\ \mathit{priK}\ \texttt{bad}) \cup (\texttt{Key range}\ \mathit{pubK})$$

Function *knows* defined how the agents' dynamic knowledge is built during the protocol run. Its definition is presented below, where the symbol # denotes the concatenation of an element at the head of a list.

$$\textit{knows}\colon [\texttt{agent, event list}] \longrightarrow \texttt{msg set}$$

1. An agent knows her initial state

$$\textit{knows } A \; [] \triangleq \textit{initState } A$$

2. An agent knows what she sends to anyone in a trace and the Spy knows all messages ever sent over it.

$$\textit{knows } A \; ((\texttt{Says } A' \; B \; X) \; \# \; \textit{evs}) \triangleq \begin{cases} X \cup \textit{knows } A \; \textit{evs}, & \text{if } A \in \texttt{bad} \\ \textit{knows } A \; \textit{evs}, & \text{otherwise} \end{cases}$$

3. An agent knows what she notes in a trace, while the Spy also knows the compromised agents' notes

$$\textit{knows } A \; ((\texttt{Notes } A' \; X) \; \# \; \textit{evs}) \triangleq \begin{cases} X \cup \textit{knows } A \; \textit{evs}, & \text{if } A = A' \text{ or} \\ & (A = \texttt{Spy} \text{ and } A' \in \texttt{bad}) \\ \textit{knows } A \; \textit{evs}, & \text{otherwise} \end{cases}$$

4. An agent, except the Spy, knows what she receives in a trace. The Spy knowledge is not enriched here since she already knows by case 2 and by the reception invariant.

$$\textit{knows } A \; ((\texttt{Gets } A' \; X) \; \# \; \textit{evs}) \triangleq \begin{cases} X \cup \textit{knows } A \; \textit{evs}, & \text{if } A = A' \text{ and } A \neq \texttt{Spy} \\ \textit{knows } A \; \textit{evs}, & \text{otherwise} \end{cases}$$

It is important to note that the expression *knows* `Spy` *evs* expresses the entire network traffic occurred to the point registered in *evs*. With these formalizations, the attacker's omnipotence requirement for the threat model is fulfilled. For the third one, the definition of message operations will be necessary, as given in the next section.

3.2.7 Operators

Operators are functions for reasoning about message sets. They translate the act of exploring and producing messages, being defined as follows.

$$\texttt{analz, synth, parts:} \;\; \texttt{msg set} \longrightarrow \texttt{msg set}$$

In this section, some new symbols are introduced. The symbol $\Longrightarrow$ denotes a sequent, where $\Gamma \Longrightarrow \Delta$ reads as *if* Γ *holds, then so does* Δ. Also, the $\{\!|$ and $|\!\}$ symbols are a shorthand syntax for the `Mpair` constructor, denoting concatenation of messages, while $[\![$ and $]\!]$ are concatenation of clauses in a list.

The `analz` operator denotes the act of inspecting the components of a message, breaking it up. Therefore, considering the message set H, the set `analz` H uses the following rules below.

1. Any element of a message set H can be analyzed from it;

$$X \in H \Longrightarrow X \in \texttt{analz}\ H$$

2. Any element in a given concatenation of messages, that could be analyzed from its message set, can also be analyzed from it;

$$\{\!|X, Y|\!\} \in \texttt{analz}H \Longrightarrow X \in \texttt{analz}\ H$$
$$\{\!|X, Y|\!\} \in \texttt{analz}H \Longrightarrow Y \in \texttt{analz}\ H$$

3. The contents of an analyzed cyphertext can only properly be analyzed and further retrieved in possession of its decryption key.

$$[\![\texttt{Crypt}\ K\ X \in \texttt{analz}\ H; \texttt{Key}\ (\mathit{invKey}\ K) \in \texttt{analz}\ H]\!] \Longrightarrow X \in \texttt{analz}\ H$$

The set `analz (spies)` ***evs*** holds all data that the Spy can gather from the network in the trace *evs*, either by decomposition or decryption of ciphers. Thus, saying a set of message X does not belong to this set is equivalent to state the confidentiality of X in such trace.

The second operator is the `synth`, which basically denotes the action of composing messages from given components. Its rules are the following:

1. Any element or agent name can be synthesized from a message set;

$$X \in H \Longrightarrow X \in \texttt{synth}\ H$$
$$\texttt{Agent}\ A \in \texttt{synth}\ H$$

2. If a message can be synthesized from a message set, then so it can its hash;

$$X \in \texttt{synth}\ H \Longrightarrow \texttt{Hash}\ X \in \texttt{synth}\ H$$

3. If two messages can be synthesized from a message set, then so it can its concatenation;

$$[\![X \in \texttt{synth}\ H; Y \in \texttt{synth}\ H]\!] \Longrightarrow \{\!|X, Y|\!\} \in \texttt{synth}\ H$$

4. If a key belongs to a message set and a message can be synthesized from it as well, then so the encryption of the message with the given key.

$$[\![\texttt{Key}\ K \in H; X \in \texttt{synth}\ \ H]\!] \Longrightarrow \texttt{Crypt}\ K\ X \in \texttt{synth}\ H$$

Finally, the last operator `parts` is very similar to the `analz` operator, hence its rules are similar to the latter, except the one concerning encryption:

4. The contents of any cyphertext can be obtained.

$$\texttt{Crypt}\ K\ X \in \texttt{parts}\ H \Longrightarrow X \in \texttt{parts}\ H$$

Thereby, the extraction of messages from ciphers does not require a key in this operator, simulating unbounded computational power. This operator is mainly used in analysis of agents' knowledge sets along proofs and not used to give an agent unlimited power over sets of messages. Additionally, the following relation can be derived concerning the last two operators:

$$\texttt{analz} \subseteq \texttt{parts}$$

Finally, *freshness* can be defined. Normally, nonces and session keys demand to be freshly generated in order to increase security. To analyze this properties for a given message the following function is defined, which receives a trace and return the messages used in that trace:

$$\mathit{used}\colon \texttt{event list} \longrightarrow \texttt{msg set}$$

1. All components of any agent's initial state are used in a trace, including the empty one

$$\mathit{used}\ [] \triangleq \bigcup A.\ \texttt{parts}(\mathit{initState}\ A)$$

2. All components of messages sent in a trace are used in that trace

$$\mathit{used}\ A\ ((\texttt{Says}\ A\ B\ X)\ \#\ \mathit{evs}) \triangleq \texttt{parts}\ \{X\} \cup \mathit{used}\ \mathit{evs}$$

3. All components of messages noted in a trace are used in that trace

$$\mathit{used}\ A\ ((\texttt{Notes}\ A\ X)\ \#\ \mathit{evs}) \triangleq \texttt{parts}\ \{X\} \cup \mathit{used}\ \mathit{evs}$$

4. All components of messages received in a trace do not count as used, due to reception invariant and hence, they are already considered as used.

$$\mathit{used}\ A\ ((\texttt{Gets}\ A\ X)\ \#\ \mathit{evs}) \triangleq \texttt{parts}\ \{X\} \cup \mathit{used}\ \mathit{evs}$$

Hence, it is said to a component to be fresh in a given protocol execution when such component was not used prior to that execution.

3.2.8 Protocol Model

Once all elements are properly defined, we are able to formally define the protocol model. As previously stated, the model will be structured as a set of unlimited traces, representing all possible network histories induced by the protocol. Here, we use the protocol in Figure 2.1 as the basis for our example. The corresponding model is presented in Figure 3.1.

The model rules are composed by two parts. The first part are the preconditions, a list of clauses that need to be true in order to fire the correspondent event. The latter are the postconditions, which states which events will be added to the head of the trace, composing the network history.

Rule `Nil` is the base case, where the empty trace is a valid trace in a protocol session. Rules `DSP1` and `DSP2` emulate the protocol steps, hence if a protocol has n parts, the model should contain at least n rules. A regular precondition among rules are that the trace must be faithful to the model, thus, considering a trace evs, then evs should belong to the model. In rule `DP2`, we see that the preconditions contains a `Gets` event resembling rule `DP1`, showing a common formalization of how rules demand previous ones.

The rule `Fake` aides the formalization of the threat context, where the Spy can fake a message X once she has the necessary knowledge, i.e. message X belongs to the set of data which can be constructed based on her knowledge set. The rule `Oops` abstracts the loss of a session key to the Spy by accident. This local security breach is used to model specific situations where a lost session key could affect future protocol sessions. Thus, this rule only concerns protocol involving this kind of keys.

Finally, the rule `Rcpt` is meant to fulfill the *reception invariant*, i.e. a message can only be received if it was previously sent. Note that it does not guarantee that sent messages will ever be received, but only enforces that traces with `Gets` events must have a matching `Says` event.

3.3 Goals Verification

Security protocols have technical properties that are inherent to its system, outlining how peers should communicate and the protocol proceed. Meanwhile, other important security aspects, introduced in Chapter 2, are another kind of features that should be part of such specifications, being them another crucial role.

When the protocol formal model is defined, its underlying characteristics are not explicit. We need to present them, precisely stating its form, and proving that they are valid in the model, i.e. they hold in all conceivable trace. This is the *goal verification* stage, where each desired protocol feature is modeled as a goal and formal proofs build

```
Nil:
[] ∈ dsp

Fake:
⟦ evsF ∈ dsp; X ∈ synth(analz(spies evsF)) ⟧
⟹ Says Spy B X # evsF ∈ dsp

DSP1:
⟦ evs1 ∈ dsp; Nonce Na ∉ used evs1 ⟧
⟹ Says A B {|Agent A, Nonce Na|} # evs1 ∈ dsp

DSP2:
⟦ evs2 ∈ dsp; Key K ∉ used evs2; K ∈ symKeys;
  Says A' B {|Agent A, Nonce Na|} ∈ set evs2 ⟧
⟹ Says B A (Crypt (priSK B) {|Nonce Na, Key K|}) # evs2 ∈ dsp

Oops:
⟦ evsO ∈ dsp;
  Says B A (Crypt (priSK B) {|Nonce Na, Key K|}) ∈ set evsO ⟧
⟹ Notes Spy {|Nonce Na, Key K|} # evsO ∈ dsp
```

Figure 3.1: Formal model for the protocol displayed in Figure 2.1

their validation to the model. Goals are divided in types, concerning the protocol property scope.

At the end, we also try to show that *goal availability* is present in the model. This property provides formal guarantees to the agents that a given goal is present in that model, even under a threat model. Hence, these agents are assured that the protocol is reliably faithful to what it intends to offer.

3.3.1 Reliability

Reliability relates to how close a model is to the system it is representing. Note that such crucial property may not be really related to the security protocol goals but to the system goals itself. As a result, the number of reliability theorems may vary from a protocol to another.

However, some basic properties, which are common for many protocols, are already defined and likely to be used in new formalizations. Some of them may seem obvious, but are important for formal systems. As examples, we can cite the idempotence of `analz` operator, fresh keys and session keys proper distinction, and the certainty of messages composition correctness.

The theorems of this class are easy to prove. They mainly use induction and simplification as its main proof strategy. Providing such goals gives us assurance that our model is reliable to the real world.

3.3.2 Regularity

The regularity lemmas are facts that can be proved from any message that appears in the traffic. Such properties may be applied to specific protocol goals, but some other broader properties can be guaranteed using them.

For example, regularity lemmas hold for long-term keys which can never be sent through the channel, since they are never meant to be used over the network, just in a agent local context. Therefore, if such agent key is seen in the traffic it is easy to derive that this agent is compromised, as stated below.

$$\texttt{Key}\ (\mathit{shrK}\ A) \in \texttt{parts}\ (\mathit{knows}\ \texttt{Spy}\ \mathit{evs}) \text{ if and only if } A \in \mathit{bad}$$

3.3.3 Confidentiality

Holding confidentiality in a protocol can be simply translated as to preventing the disclosure of certain messages to the Spy, that is, a message X cannot belong to the Spy knowledge set. Since messages are usually protected by encryption, this property is highly related to the use of cryptographic keys.

Precisely, the confidentiality of such keys is a major issue for the protocol confidentiality, specially session keys, because if the Spy obtains them, all messages encrypted under these keys could be easily acquired and altered by her. Hence, given a key K, we have to be certain that at the end of all traces the key does not belong to the Spy knowledge set, as defined below.

$$\texttt{Key}\ K \notin \texttt{analz}\ (\mathit{knows}\ \texttt{Spy}\ \mathit{evs})$$

Note that if K is encrypted with some other private key of an agent, the latter cannot be part of the compromised agents set. Otherwise, the Spy could easily retrieve the key from the compromised agent, obtaining the session key and further messages. Confidentiality is also interesting for nonces, which are commonly used for assuring authenticity, computing checksum and other operations.

3.3.4 Unicity

The creation of fresh components in protocol is vastly used. It is seen in the production of session keys, nonces and other entities and they are mostly used for a single session, identifying it. Therefore, providing freshness in a protocol resembles unicity, a concept that establishes bounds between a message and its fresh components.

More precisely, if two events contain the same fresh message component, then they must be identical. Further, events containing fresh message components cannot occur more than once, otherwise they violate the unicity concept. Such lemmas are deeply explored when protocols apply the use of nonces and timestamps.

A formalization for this definition prompted the creation of a new predicate in Isabelle/HOL *Auth* library, the `Unique` predicate. It takes an event and a trace as parameters, holding if the given event is unique in the given trace. Such formalization is crucial for detecting replay attacks over traces.

3.3.5 Authenticity

This property can also be read as legitimacy. In the method, the guarantee of a message's authorship is presented as synonym of integrity, since if the message is unaltered (integrity), then the authorship must be preserved. Even if the Spy intercepts the message and them relays it to the recipient, if integrity is preserved, then legitimacy is still preserved, since the Spy acted as a channel relay.

As a result, it is important that the message's author does not belong to the compromised agent set. Otherwise, any messages sent by him would be compromised as well and authenticity would not hold. Such concept may seem obvious for integrity matters, but it is important to enforce the authorship interest. Therefore, both properties are attached during our verification.

3.3.6 Authentication

As discussed in Section 2.6, authentication may assume many properties. Suppose an initiator A, who completes a protocol session with a responder B. In this run, authentication may be translated as:

1. **Aliveness of B**, meaning that B has been running the protocol;
2. **Weak agreement of B with A**, meaning that B has been running the protocol with A;
3. **Non-injective agreement of B with A on H**, meaning a weak agreement of B with A, considering the set H of message components;
4. **Injective agreement of B with A on H**, meaning the non-injective agreement of B with A, using the set H of message components, where B did not respond more than once on each session with A.

The Inductive Method does not provide formalisms to reason about the fourth point. Although, it is claimed on [3] that such formalization can be easily constructed by simple verification of repetition of a given event *ev* in the analyzed trace, restraining the agents to single responses. However, the method mainly tries to provide models that are more permissive as possible, stating that any message could be repeated over traces with no harm to model.

Specifically, non-injective agreements have a wider focus. Regarding key distribution protocols, such property applied to session keys establishes a trust relation between the two agents, with a given key as a validation of such relationship, since both agents are uncompromised.

Eventually, **key distribution** becomes a major goal to be checked. Since this concept is related to the agreement of two agents in a mutual secret, it is stated in [27] that this property is, indeed, stronger than authentication and thus, authentication itself relies on key distribution. Additionally, the authors of the Inductive Method prove that if authentication holds, so does key distribution, specifically non-injective agreement on a session key.

3.3.7 Goal Availability

The concept of goal availability is one of the main contributions in the work of G. Bella [3]. In summary, it establishes a formal guarantee that the protocol model could meet a given goal and, once this is true, the validity of such goal can be applied to all peers within their assumptions. If such property holds in the model, but cannot be checked by the peers, it is argued that this goal is not available and thus, not fully assured.

Also, goal availability is closely related to the threat model considered by the protocol and its peers minimal trust. Considering the former, our adopted model is the Dolev-Yao, faithfully implemented by the Inductive Method using abstractions described in Sections 3.2.6 and 3.2.7. If a different threat model is considered, important aspects of the model context are also altered, which may change the validity of previous goals.

Minimal trust concerns facts that agents must assume true, due to inability to verify them, in order to check a protocol guarantee from their point of view. Precisely, agents can only verify what they send and receive from the network, having no guarantees at all about other peers. Likewise, agents cannot state if they are compromised. Therefore, they must take such facts as true, in order to establish a suitable set of clauses to attest some goal. We formally define minimal trust below.

Definition 1 Let $\mathcal{P}$ be a security protocol, P be a formal model for $\mathcal{P}$, and A be an agent's name. The *minimal trust* of A is the set of environmental facts formalized in P whose truth values A needs to know but can never verify.

Having the notion of what agents can verify, an *applicable guarantee* can be defined.

Definition 2 Let $\mathcal{P}$ be a security protocol, P be a formal model for $\mathcal{P}$, and A be an agent's name. A formal guarantee in P is *applicable* by A if it is established on the basis of assumptions that A is able to verify in P within her minimal trust.

In these previous definitions, it is clear that guarantees that are applicable by an honest agent A are not necessarily applicable by another honest agent B. Hence, this strengthens the argument that only proved assumptions concerning protocol messages, specially the ones related to the agent, are the valid formal guarantees for a given protocol goal.

Finally, the definition of available goal can be made. The principle should be used to guide the formal proofs about goals. As a result, whenever a goal is not available to some peer, it is not guaranteed at that model and so, it does not hold for the given protocol.

Definition 3 Let $\mathcal{P}$ be a security protocol, P be a formal model for $\mathcal{P}$, g be a goal for $\mathcal{P}$, and A be an agent's name. The goal g is *available* to A in P if there exists a formal guarantee in P that confirms g and that is applicable by A in P.

Chapter 4

DAP: Dynamic Authentication Protocol

In this chapter, our case study, the Dynamic Authentication Protocol (DAP), is presented. The protocol is designed for providing an extra reliability layer for the online banking application, creating an encrypted authorization scheme for each banking transaction over a previously authenticated session between the User and the Bank.

Prior to running the protocol, a shared key must be established between the User and the Bank, which will be stored both at the User's uncompromisable smartphone and the Bank Server. There are two different ways to generate such a key, a symmetric and an asymmetric method, where such operation is performed only once for all protocol history.

Then, the User can issue transactions at the bank server, where each operation generates a Transaction Authorization Nonce (TAN) which is sent together with the prompted job as reply. At the User side, the message is checked and a challenge is presented, as a QR-Code message. The User scans it with her smartphone, which interprets, checks and displays the operation, asking for its authorization. If it is correct, the User replies to the Server, sending the generated TAN to the Server. The latter compares it with its own copy and if it matches, the operation is confirmed. All message exchanges are encrypted and protected, using parts of the previously shared key.

In order to understand the protocol context, we first present a brief introduction to internet banking systems, indicating its challenges. Then, we detail the protocol entities, phases and strategies. At then end, we conclude with a short discussion about security claims assumed by this protocol.

4.1 Internet Banking

Internet banking is an online environment which provides a communication channel between a bank and a client, where the latter can perform financial transactions and other banking operations. It is aimed to fulfill the customer demand, reduce costs and increase efficiency at the bank side [6].

As claimed by DAP authors [1, p.8] and further supported by literature review [6], there are few formal definitions which applies exclusively to internet banking systems. Those systems are usually constructed within simple architectures and scale up to large environments, with a series of interconnected devices. However, the model can be simplified by focusing on three entities — the user, the bank and the Internet — resembling a client-server model architecture. This pattern is commonly found in the literature [6].

At the client side, many access clients can be used, where the personal computer (PC) is the classical and most common case. As new technologies arise, smartphones are also a widely accepted option. In [1], a large private base of 1.3 million clients was analyzed to attest this claim.

4.1.1 Security Issues

Since internet banking deals with sensible and financial data, those systems urges for strict security assurances. Such pledges bring reliability to the system and for its client base.

Specifically, authenticating clients and further operations carried by them are the key point for providing a reliable internet banking framework. Further, authorization, integrity and confidentiality are commonly cited qualities and finally, non-repudiation, availability and auditability are also desired properties for those systems [6].

At the same time, such systems are a preferential target among hackers and other cyber-criminal [28, 6]. Several techniques for credential theft and transaction manipulation for internet banking systems have been developed over the years, with formal classification studies made [28, 7]. It is interesting to cite some common attack patterns:

- **Channel Breaking:** intercepting the communication between client and bank, and impersonating any of the entities in order to fake a legal identity and trying to perform actions as a legal user or as the server;
- **Phishing Attacks:** the attacker acts as the bank, publishing fake content and data collector scheme, in order to trick the user to handle in her credentials, and eventually using those credentials to perform further transactions;

- **Device Control:** the attacker combines several exploiting techniques in order to have full control of the user device, being able to steal credentials, perform fraudulent actions, spy on the user, and other malicious actions.

Credential Thief is a broader technique category, which involves the theft of user credentials in any viable way. Therefore, phishing attacks also fall into this category.

Accordingly, many security schemes were developed in order to oppose against such attacking strategies. Also in [28, 7], the authors provide a listing of current common security models found among banks.

Still in [28, 7], the authors also cite how these models fail to provide reliability to the system, describing possible flaws which are passive of exploitation and strategies used by attackers. In short, such proposals are not enough for providing security for banking transactions.

One promising method, highlighted in [1, p.61], is the *One Time Password* (OTP) model combined with an external authentication channel. Particularly, in [8], a smartphone is used as the external entity to validate messages exchanged between the user PC and the server, requiring that the smartphone could communicate with the server. Further, in [9], the QR-Code technology is added to the scheme in order to provide a more reliable approach.

However, it is claimed that both schemes focus on transaction authentication, which may fail when a message is not associated to a session key. Therefore it may be a more secure approach to authorize transactions with OTP models, under a valid authenticated session. Additionally, the necessity of connectivity between the smartphone and server is a limitation that should be removed.

4.2 Motivation

The DAP was the first attempt to provide a secure internet banking environment for the client base of a notorious bank in Brazil. The protocol is composed by a set of cryptographic protocols, orchestrated to offer integrity, authenticity and secrecy over banking transactions among any valid channel defined by the bank.

Also, bandwidth and computational power limits are established. The whole system must be able to run in devices with low computational power, requiring minimum user interaction, aiming a reduction in the volume of data exchanged between client and server. Therefore, this limitation creates a restriction for some of the protocol primitives and complexity.

It is important to emphasize that the protocol identifies a unique transaction on each run. The user must be already authenticated before prompting a transaction authorization and private authentication keys must be previously exchanged between the server and the client. In a protocol run, the user receives a series of challenges, in order to verify the operation, which contain pieces of information about the given transaction.

One of these tasks involves the use of a third-party device equipped with a camera and QR-code scanning technology, usually a smartphone. This phase works as a final hash verification for each operation, where the user should scan a QR-code containing the transaction details, check it and send a 6-digit PIN that corresponds to the transaction authorization identifier, resulting in the final transaction authorization.

The protocol authors claim that this last step produce an extra security layer that could only be exploited through social engineering tactics, hardening the attack against a series of known strategies. In summary, an attacker can fake an operation but cannot authorize it, since the authorization procedure was removed from the session context and transferred to the user responsibility over an offline channel.

4.3 Definitions

First, we define the types of agents available in the system. The protocol informal specification defines two main entities, which are the Server and the User. The former relates to the Bank itself, where it may be representing the Internet Banking application or the system mainframe. The latter is always the protocol initiator and interacts with the Server using three different communication channels: a personal computer, a mobile smartphone or an ATM machine.

Each device has its proper role. Both smartphone and, mainly, the personal computer, act as the protocol initiator, for generating the shared key between User and Bank. At the User side, this key will be stored at the smartphone only. The ATM machine acts as a middle communicator, intended to give a part of the shared key to the User. In special, the personal computer is the only communication channel able to issue banking transactions to the Server.

Also, in the symmetric key generation phase, the User must use a personal smartcard for authentication. Such concept is vastly used among literature [29] for providing an extra authentication layer for protocols.

4.3.1 Initial Assumptions

Based in assumptions that resemble the Dolev-Yao approach, the authors state some basic premises for constructing the model, as follows:

1. The Spy has full control of the network, being able to intercept, block, produce or inject any kind of message, by any agent using the channel;
2. The User personal computer belongs to the set of compromised agents, i.e. the Spy has access to any information stored, produced or received by the User's computer and can produce and send messages from it, impersonating the User;
3. ATM machines and the smartphone do not belong to the compromised agents set. Even though, the hypothesis assume that the Spy can access any data directly displayed to the User via terminal access in an ATM machine. This premise does not hold for smartphones;
4. The Spy does not have access to any personal smartcard.

Finally, the problem statement may be placed: the User wants to send a message m to the Server, which contains a certain operation to be authorized. Thus, the proposed security protocol DAP will be used. Finally, an active Spy will try to exploit such system, trying to perform an illegal action.

4.3.2 General Model

The system has two major parts: the generation of shared key K, between the User and the Server, and the message exchange operations, which will carry the banking transactions requested by the User.

The production of the shared key K is done only once, therefore the same key is used along all later messages, for any banking transaction. Also, this phase may be done by two ways, symmetric and asymmetric, depending on the channel the User chooses.

4.4 Key Generation

The key generation focuses on the establishment of K: a shared key between a given user and the bank, used for further message transactions. At the end of generation process, the key will be securely stored at the User's smartphone and the Bank mainframe environment.

Each user will have its own shared key with the Bank, meaning that the number of keys stored at the bank side is identical to the number of clients which are employing the DAP system. Therefore, requesting such a key means that the client is interested in using the extra protection layer. Hence, key generation is always started by the User.

Following the protocol initial assumptions, both the Server and User's smartphone are uncompromisable. That said, we emphasize that K is assumed not compromised as well, since it is stored only at entities that are not compromised by the Spy and does not appear in clear text over the network channel.

4.4.1 Symmetric Scheme

The symmetric model of key generation is prompted by the User at the Internet Banking application over a personal computer. It is divided in two phases, where the first is centered in the smartphone and the second is aided by an ATM machine. The general scheme is illustrated in Figure 4.1.

First, the User requires to ingress the two-factor authentication scheme and for that, she must be previously authenticated in the Internet Banking system, on her personal computer.

The Server receives the request and generates three entities:

- The cipher N', whose generation is based in the User's ID and the operation request timestamp;
- A randomly generated fresh nonce N_S;
- A cipher N_A, which is obtained by the binary addition of two fresh nonces, namely $N_A = N' \oplus N_K$.

Next, the Server responds the initial request with nonce N_S, which is noted by the user over the personal computer channel, presented in the QR-Code form. The User must use her smartphone to scan the code, storing the nonce within the device. Now the User is prompted to access an ATM machine, in order to collect the second part of the key.

At the ATM machine, the user must present her credentials, through a personal smart-card and password. Proceeding the operation, the terminal requests the second part of the key to the Server, N_A, which delivers it through a secure channel. Hence, the ATM displays N_A as a QR-code, which is scanned by the User's smartphone.

Now, the smartphone computes key K, using N_S and N_A in a binary addition operation, discarding them afterwards. Since the Server generated both of the inputs, it can also computes the key K and the key is properly distributed between the Server and the User.

4.4.2 Asymmetric Scheme

The asymmetric scheme resembles the public key RSA algorithm [30]. The User must issue the generation of key K at her smartphone. Although, a pair of keys are generated, pK_U and sK_U, the public and private keys respectively. Then, it sends the public key to the Server.

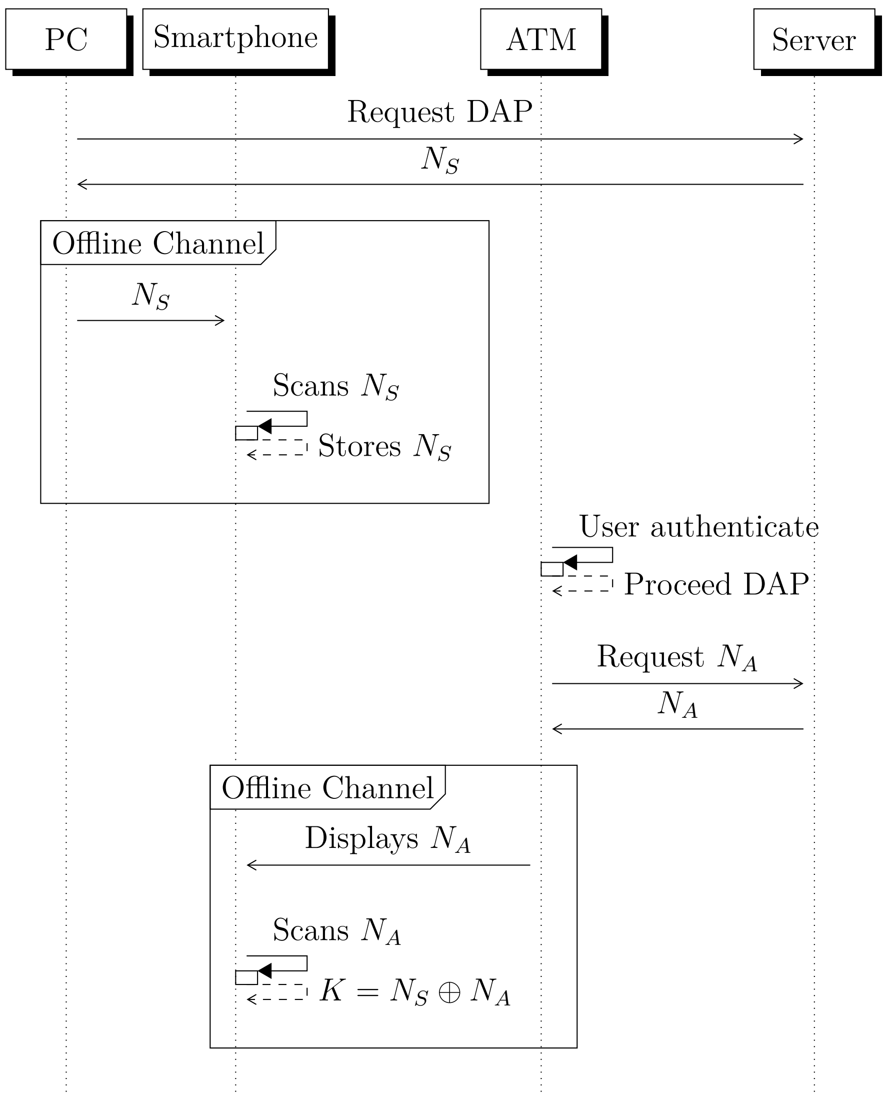


Figure 4.1: Diagram of a symmetric method for generation of key K

As soon as the Server acknowledges the User public key, she signs it with her private key sK_S and sends her public key to the User. When the latter notes the Server's public key, she stores it, encrypting it with her private key.

This process, described in [1, p.78], seems to have some inconsistencies. In private conversation with the protocol authors, we identified that the document needs reviews in this part. The text claims to describe a model based in public key infrastructure and RSA algorithm, but the presented scheme does not match the assertion. Thus, some naming conventions in the specification are not clear enough.

Therefore, we decided to provide the described methodology in [1], but noting that it will be revised and further altered for consistency.

4.5 Message Transactions

Once the shared key K is available for both the User and the Server, they can perform secure banking transactions, with atomic authorization for each operation. Initially, the User must be authenticated in the Internet Banking system, for requesting transactions. Note that this security layer is not enough for guaranteeing reliability, since the Spy has full control of the User's personal computer.

Additionally, the key K is divided into two others: k_1 and k_2. The former is used for message integrity, as an input for HMAC-SHA1 protocol [31], providing consistency checks among messages exchanged between the Server and the User. The latter is used for message secrecy, as an input for AES protocol ciphers [32], protecting messages or its components over clear channels.

Figure 4.2 explains the whole message transaction process, which is based in the following steps:

1. The User prompts the banking transaction m through the Internet Banking application, at her personal computer. She must be previously authenticated. The transaction is sent in clear text to the Server;

2. Once the Server receives m, it generates the **Transaction Authorization Nonce** (TAN), which we will name r_S. This nonce will uniquely identify the transaction m, acting as a reference for further authorization. Also, it produces cipher r', encrypting nonce r_S using key k_2 and an AES encryption scheme, and the checksum hash h_S, applying the HMAC-SHA1 protocol over the concatenation of transaction m and cipher r', using key k_1. Finally, entities m, r' and h_S are grouped and sent to the User;

3. At the User side, the personal computer receives message $\{\!|m, r', h_S|\!\}$ and visually displays it to the User as a QR-code cipher. The User may now use her smartphone to scan the code, which will input m, r' and h_S to the device;

4. By decomposition, the Smartphone retrieves m and r'. It now calculates the cipher h_U, using its own k_1 key with HMAC-SHA1 protocol, and performs the checksum between h_U and h_S. If the ciphers do not match, the Smartphone announces the error, halting the protocol. Otherwise, it considers the received transaction valid and displays it on its screen, presenting the challenge to the user;

5. The User is challenged to verify the transaction, being prompted to confirm its legitimacy. If the operation is declined, the protocol halts. Otherwise, the User confirms the message, sending a confirmation acknowledgment to the Smartphone;

6. Having the transaction checked and approved, the Smartphone decipher r', using key k_2, obtaining its own version of the TAN, namely r_U, displaying it on its screen;
7. The User retrieves nonce r_U from her Smartphone screen, sending it to the Server through its personal computer;
8. The server receives r_U and compares it with r_S. If they do not correspond, the protocol halts. Otherwise, the Server authorizes the transaction. Finally, it sends an acknowledgment to the User, confirming the authorization.

4.6 Security Discussion

Finally, we discuss the correctness arguments concerning security on DAP, postulated as hypothesis, constrained to a defined model of the system. This model restricts the entities which are vulnerable to the Spy, postulating the following:

- The User personal computer and its channel with the Server are compromised;
- The ATM machine and its channel with the Server are not compromised;
- The User smartphone is not compromised;
- The Server is not compromised.

Given that, the key K, shared between user Smartphone and Bank for further message transaction validation, is stored in uncompromisable devices only. Additionally, the smartphone never sends any data over the network, so there is no data leak at this device and the key K — and its derived keys k_1 and k_2 — are secured.

Regarding the message transaction phase, the integrity of the transaction is assured by the hash h_S, where such cipher can only be produced by the ones possessing key k_1 derived from K, and further comparison with the hash produced at the User smartphone. Besides, the TAN r is stated as a randomly fresh identifier for the transaction, delivering unicity to the protocol session.

Moreover, even if the Spy intercepts message m, alters its structure and relays it to the intended peer, still, the User will be prompted to answer the security challenge where she must verify and confirm the issued transaction, presented in clear text. If she suspects anything, she can simply drop the operation, halting the protocol.

However, none of these claims are formally proved. The authors have simply based their model in a hypothesis in a closed model. Therefore, we identify that no formal guarantees are presented at all.

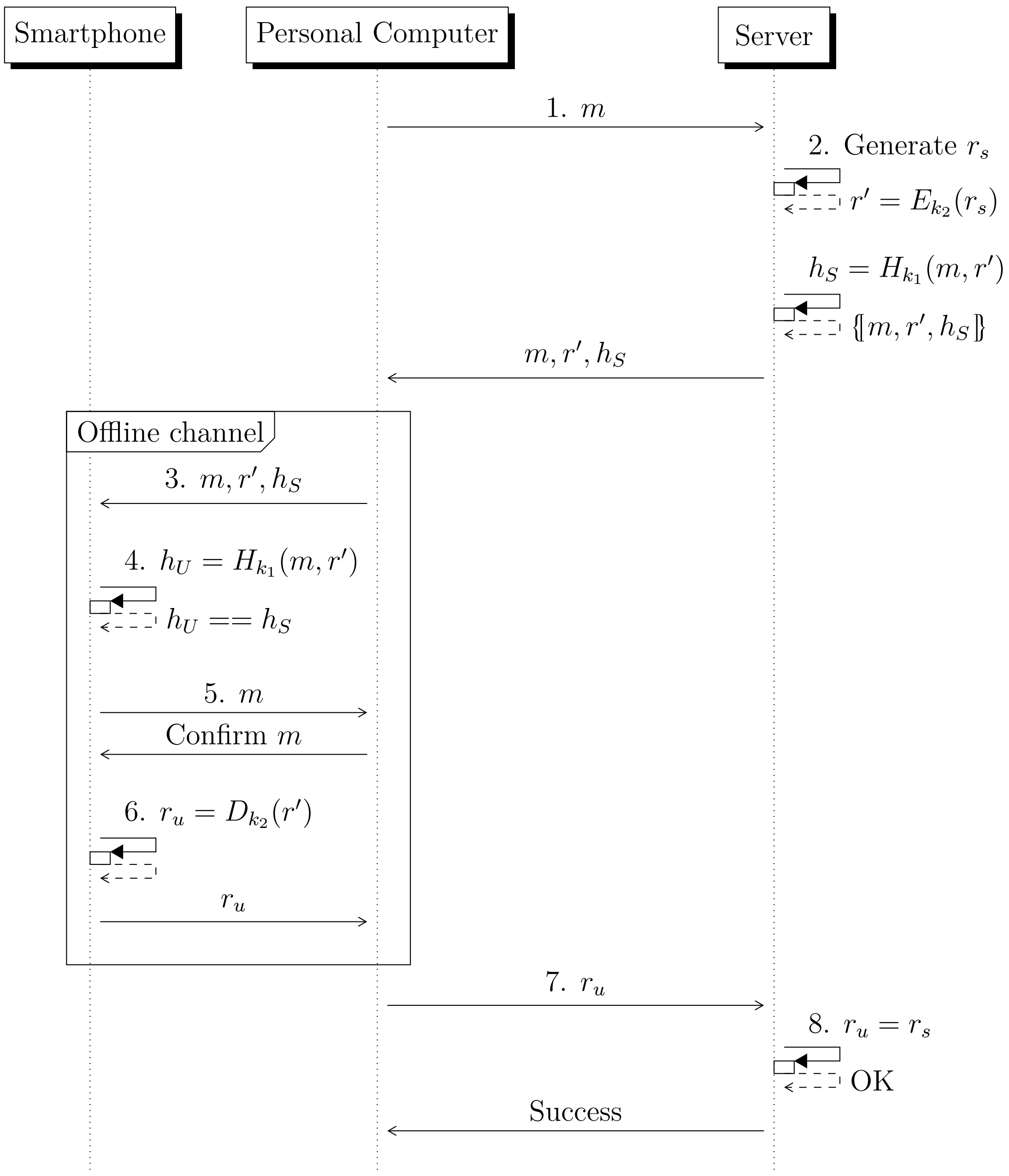


Figure 4.2: Diagram of a message transaction in DAP. The protocol steps are referenced in the diagram

Chapter 5

Offline Channel Formalization

As seen in Chapter 4, the DAP main strategy for securely authorizing transactions is providing a verification phase between the human User and her smartphone device, where the transaction will be revised and further confirmed by the User. All these operations take place in an offline and out-of-band channel, which is supposed to be inviolable by the Spy. Besides the inviolability of the channel, the DAP specification makes other strong assumptions about the smartphone security model, which will be further discussed later in this chapter.

As stated in Chapter 3, the Inductive Method has proper formalizations for the network channel and regular agents behavior, but we stress that it does not take into account the offline channel and the smartphone peculiarities. In order to verify the protocol, we need to provide a reliable and suitable model for this context. In this chapter, we construct this formal model, considering all the real world details surrounding these entities, adapting the protocol assumptions into a conceivable scenario.

5.1 DAP Analysis

The DAP specification describes the User Smartphone as a key component for the protocol operation and, further, vital for its security. In the following, based on the protocol operational steps, we identify and list the required set of actions for a device rightly fulfill the protocol's goals:

Cryptographic Calculus: the device must be able to securely generate and perform cryptographic keys, hashes, calculations and checks. Such operations are constantly performed by the smartphone on the hash verification phase and decryption of the TAN;

2D code reading capability: the ability of scanning and decoding any presented QR-Code through a physical optical reader is what gives the device means of communication with other devices. Recalling the protocol specification, the Smartphone uses its camera for scanning QR-Codes along other agents' screens, obtaining the two pieces of the symmetric key K and, further, banking transactions details;

On-screen data output: in order to show any received message and further operations to the User, the Smartphone must provide a visual physical channel to display such data;

Physical input: at last, the User must be able to accept or decline messages presented on the device screen. Therefore, the smartphone must provide a way for the User to input these actions on the phone.

The above described properties can be embedded on a dedicated hardware, other than smartphones, which could be issued by the bank to the protocol users. Each client would have their own device, already packed with any required encryption key, establishing the proper link between the Bank and the User. Similar schemes have been already implemented by other online banking systems, for instance, One Time Password (OTP) proprietary tokens are delivered to bank clients, in order to compose two-factor authentication and transaction authorization systems for online banking protocols [33, 34].

However, the protocol authors enjoyed the convenience of the smartphone. Not only the device appropriately fits the requirements for the protocol mandatory hardware but also has a strong and known acceptance among users around the world [35]. Therefore, the device is a reasonable choice.

5.1.1 Smartphone Scope

Smartphones are portable devices that have an operational system, which provides the capacity to perform some actions like a personal computer. They can process data, store information and communicate through a network channel.

Recalling the DAP specification [1, Ch. 4], the establishment of shared key K acts as a bound between the User and the Bank. It is securely stored both at the the Bank and the User side, where the latter uses its Smartphone as the entrusted entity for this task. Not only a link between agents is established, but the authentication process is also reinforced, given that only these two entities must know such secret.

Plus, the specification also states that any communication between the smartphone and other devices must be done by visual means: either it outputs data through a screen or it obtains information by the smartphone camera.

These strategies provide more security based on two main facts, according to the specification: the long-term keys are never disclosed to the User's computer, which is considered insecure, and any necessary communication between the phone and another device is restricted to a non-interposable and reliable channel [1].

However, it is important to note that the smartphone performs more actions than those required by the protocol. Due to its set of capabilities, the smartphone can act like a personal computer, which in a precise sense makes those devices as unreliable as any other online device. Hence, we have possible attack vectors that should be discussed.

5.1.2 DAP Assumptions *vs.* Real World Scenarios

Before going further, it is important to discuss the environment described in the DAP specification and contrast it with the real world. The protocol specification makes strong assumptions about the User smartphone, stating that it cannot be directly accessed or operated by the Spy.

We argue that such claim is idealistic, since recent studies have shown critical security flaws and attacks in the major mobile phones operating systems [36, 37], enabling remote operations by an attacker, for instance, which goes directly against the security requirement dictated by the protocol.

Moreover, the situation of a smartphone robbery must also be taken in account. Here, the Spy could not only operate the smartphone but, if she is skilled enough, she could also exploit the device, obtaining the same privileges as if the smartphone is compromised. This scenario is effectively contemplated in Inductive Method extended theory for smartcards [3, Ch. 10].

Therefore, in contrast with the DAP specification, but in accordance with the formal method for verifying a protocol [3, Ch. 11], we should consider a scenario where the Spy can exploit the User smartphone. Since it is feasible, the theft scenario will be considered in all cases. This strategy will give rise to two possible formalizations for the protocol and introduce new forms of exploiting the protocol.

Given the fact that the smartphone can connect to the default network channel, it can be reached by the Spy and further compromised by her. Like an ordinary computer, the smartphone can be exploited in such manner that the attacker can disclose stored data, eavesdrop communication, fake data, and perform other malicious actions through remote control.

At the same time, such attack is only possible when the smartphone is connected to the network channel, i.e., the Internet. Thus, we must properly formalize the possible states of mobile phones connectivity with the cited channel. Also, note that the offline channel remains secure and what is compromised it is only the User device.

5.2 Smartphone Formalization

As a first step towards its formalization, we properly define smartphones, as an entity linked to one, and only one, agent:

Definition 4 Smartphones are defined as a bijective relation between the agent set and a free type set, denoting the set of available smartphones.

$$\texttt{Smartphone} : \ \texttt{agent} \longrightarrow \texttt{smartphone}$$

Note that such formalization bounds the ownership of a smartphone to one User only, establishing a permanent link between both entities. In the DAP specification, such bound is described in terms of the key K, shared between the User's smartphone and the Bank, which identifies the User towards the Bank. This entity is defined in a software context, in the smartphone memory, thereby it may be exploited. Furthermore, a bound between the User and the Smartphone can also be established by hardware means, using the SIM card, present in any smartphone in order to the device receive a proper phone number.

However, these details are confined to the protocol implementation in the underlying software executing it. In short, the formalization of the smartphone entity cannot fully understand how the application will treat the link between a User and a mobile phone, combining software and hardware aspects. Accordingly, we preserve the theoretical ownership aspect, using a conservative approach, stating that our model establishes the link between the two entities and keep it until the end of all trace.

5.2.1 Vulnerabilities

An evaluation of realistic vulnerabilities for smartphones is presented, in order to fully understand how such aspects must be formalized in our final model:

- **Theft:** mobile phones are highly susceptible to theft, considering that its reduced size also increases its risk to loss and stealing. Hence, we need to formalize devices that are stolen from their owners and used by the Spy, including them in a correspondent set `stolen`;
- **Device control:** mobile phones are reduced personal computers, which can be held by users. As noted, they can also be exploited and controlled remotely by the Spy, giving her the ability to request computational actions and access device data. Therefore, compromised smartphones could be added to the set `badp`, of compromised agents. However, such capacities are limited to the set of computational operations, which does not include physical actions with the smartphone, like

pointing its camera to something. As a result, the formalization of a compromised device must not include such actions, but only the ones achievable by computational means.

Additionally, once robbed by the Spy, a smartphone may be in possession of a skilled attacker, which now can easily compromise and exploit the device. Even if it is unlikely, this claim is not idealistic [38]. Hence, we state that if a smartphone is robbed, it is automatically compromised, leading to the relation below.

$$stolen \subseteq badp$$

Considering that the DAP make the assumption of secure devices, that is, when the Spy cannot compromise smartphone devices, we consider a special flag, defined at the protocol model level, in order to state such situation. When the *secureP* flag is used, the model consider all devices secured, otherwise the *insecureP* is used. This properties will influence in the definition of how the Spy learn messages along the protocol. The definition of one is simply the negation of the other.

5.2.2 Keys

In this model, agents' long-term keys are now stored in the Smartphone and agents do not have direct access to it anymore. Thus, such keys are left out of agents' initial state. However, key formalization is kept, since it belongs to the agent:

$$\texttt{shrK} : agent \longrightarrow key$$

How such keys are stored does not require an explicit formalization by the Smartphone model, but does require a description of how they are handled by agents and their devices. This approach flexibilizes our specification process for smartphones, leaving any specific aspects to the protocol model specification.

Several smartphone's operational systems present to their users the possibility to be operated until a passphrase is given, protecting them from any unwanted access and further illegal actions. However, such behavior is not required by the DAP specification and, given the fact that stolen devices are automatically considered compromised, such security scheme is irrelevant. Thus, we do not present a formalization of PIN-required actions on the smartphone.

Finally, we mention the QR-Code ciphering scheme that happens over some steps of DAP. The protocol standard [39] presents a matrix barcode scheme for displaying information, using four possible encoding modes: numeric, alphanumeric, binary and kanji. That said, we emphasize that no innate encryption happens in such codification and so, there is no need for a key definition here.

5.2.3 Usability

Usability concerns the characterization of smartphone operations, hence modeling this property affects both legal agents and the Spy. Legal agents only perform legal actions, while the Spy can perform both legal and illegal actions.

Smartphone actions are those used to exchange data with the agents. It can be scanning a QR-code from a screen, showing data through its screen or receiving an input. However, compromised devices can leak messages to the Spy in the first two actions. The latter is only accessible to her if the device is compromised. So, the definition of legal and illegal actions will be defined based on the state of the secured devices flag.

Since legal actions are done by legal users, it is easy to define the legal operation. A smartphone is legally used when it is in the user's possession:

$$\mathit{legalUse}(\texttt{Smartphone}\ A) \triangleq \texttt{Smartphone}\ A \notin \texttt{stolen}$$

When devices can be compromised, the Spy can obtain any scanned or showed messages by the smartphone. Additionally, if she has access to the smartphone, she can also obtain such messages:

$$\mathit{illegalUse}(\texttt{Smartphone}\ A) \triangleq \texttt{Smartphone}\ A \in \texttt{stolen} \lor \texttt{Smartphone}\ A \in \texttt{badP}$$

In the case where devices are secured, the Spy can only obtain any data from the smartphones when they are in possession of them:

$$\mathit{illegalUse}(\texttt{Smartphone}\ A) \triangleq \texttt{Smartphone}\ A \in \texttt{stolen}$$

It is important to also discuss the smartphone usability for the Spy, given that she can also perform legal actions. In that sense, we stress that she cannot use her own smartphone illegally, given that such action would not result in any more relevant information to her knowledge set. Hence:

$$\texttt{Smartphone}\ \mathit{Spy} \notin \texttt{stolen}$$

The same is assumed to the server, since it does not use any smartphone.

5.2.4 Events

Having the notions of smartphone threats and usage, we are able to go forward and define the possible event among protocol traces. We extend the `event` datatype as follows:

```
datatype event ≜ Says agent agent msg
                 Notes agent msg
                 Gets agent msg
                 Inputs agent smartphone msg
                 SGets smartphone msg
                 Outputs smartphone agent msg
                 AGets agent msg
```

The regular network events are kept as in Chapter 3, while four new events are added, which describe the agents and smartphone interactions with the offline channel. We depict each event below:

`Inputs` defines the act of an agent sending data to a smartphone. Hence, this event describes when a smartphone scans data from some agent's screen;

`SGets` defines the reception event at the smartphone side, in respect to a *Inputs* event;

`Outputs` defines the sending of data from a smartphone to an agent, using the smartphone screen. Like the *Inputs* event, it involves both entities;

`AGets` finally, defines the reception of data by an agent from the offline channel.

It is easy to note that the new events allow a proper distinction from messages received on the regular channel from the ones received in the new channel. Yet, the Spy still can forge messages on compromised devices and send them through these channels.

The creation of proper events for message reception in offline channels, `SGets` and `AGets`, provides a reliable way for smartphones correctly receive messages sent by `Inputs` events and agents receive messages sent by `Outputs` events, respectively.

We introduce the formal definition of the *used* function, describing how each new event contributes to enrich the set of past messages present in the current trace:

4. All components of a message that an agent inputs to her smartphone in a trace are used on that trace:

$$\texttt{used}((\texttt{Inputs}\ A\ P\ X)\ \#\ evs) \triangleq \texttt{parts}\ \{X\} \cup \texttt{used}\ evs$$

5. All messages that a smartphone receives as inputs from an agent in a trace do not directly extend those that are used on that trace.

$$\texttt{used}((\texttt{SGets}\ P\ X)\ \#\ \mathit{evs}) \triangleq \texttt{used}\ \mathit{evs}$$

6. All components of a message that an agent's smartphone outputs in a trace are used on that trace:

$$\texttt{used}((\texttt{Outputs}\ A\ P\ X)\ \#\ \mathit{evs}) \triangleq \texttt{parts}\ \{X\} \cup \texttt{used}\ \mathit{evs}$$

7. All messages that an agent receives as inputs from a smartphone in a trace do not directly extend those that are used on that trace.

$$\texttt{used}((\texttt{AGets}\ A\ X)\ \#\ \mathit{evs}) \triangleq \texttt{used}\ \mathit{evs}$$

We stress that Cases 5 and 7 do not extend the set of used components, because its corresponding events already include these components in such set.

5.2.5 Agents' Knowledge

With those new events, the agents' knowledge sets are now subject to changes compared to the ones defined in Chapter 3. The smartphone is a device that does not necessarily disclose its data to its agents, but it requires communication in order to enrich the agent's knowledge set. At the same time, we observe that compromised devices will disclose its secrets to the Spy.

The definitions on the agents' initial knowledge set are properly updated, following the guideline of Section 3.2.6 and taking into account the secrets held by the smartphone. We omit asymmetric long-term keys for readability:

1. The Server's initial knowledge set consists of all long-term secrets, stored both in agents and smartphones:

$$\begin{aligned}\texttt{initState Server} \triangleq &(\texttt{Key range shrK}) \cup \\ &(\texttt{Key shrK}\{A.\ \mathit{Smartphone}\ A\})\end{aligned}$$

2. The knowledge set for legitimate agents remains the same: their own secrets and all public keys. They do not know their smartphone secrets before any communication;

3. The Spy knows all public keys and all secrets from compromised agents and smartphones, which includes her own:

$$\texttt{initState}\ \mathit{Spy} \triangleq (\texttt{Key shrK bad}) \cup (\texttt{Key shrK}\ \{P.\ \mathit{Smartphone}\ P \in \mathit{badP}\})$$

The function *knows* will also be extended. The new events impacts both on agents and the Spy knowledge set, where the latter needs careful analysis, as follows. When the assumption of secured devices does not hold, we have that:

4. An agent knows what she inputs to any smartphone in a trace. If an agent is compromised, then the Spy also knows any given message;

$$\texttt{knows}\ A\ ((\texttt{Scans}\ A'\ P\ X)\ \#\ \mathit{evs}) \triangleq \begin{cases} \{X\} \cup \texttt{knows}\ A\ \mathit{evs} & \text{if } A = A' \text{ or } (A = \mathit{Spy} \text{ and } A' \in \mathit{bad}), \\ \texttt{knows}\ A\ \mathit{evs} & \text{otherwise} \end{cases}$$

5. No legal agent can extend her knowledge with any of the messages received by any smartphone in a trace, since she already knows given the case of `Inputs` event. However, if the smartphone is compromised, the Spy can learn any information it receives;

$$\texttt{knows}\ A\ ((\texttt{SGets}\ P\ X)\ \#\ \mathit{evs}) \triangleq \begin{cases} \{X\} \cup \texttt{knows}\ A\ \mathit{evs} & \text{if } A = \mathit{Spy} \text{ and } P \in \mathit{badP} \\ \texttt{knows}\ A\ \mathit{evs} & \text{otherwise} \end{cases}$$

6. An agent does not know what her smartphone outputs in a trace, since the device can be compromised. The Spy can obtain all information output by compromised smartphones;

$$\texttt{knows}\ A\ ((\texttt{Shows}\ P\ A\ X)\ \#\ \mathit{evs}) \triangleq \begin{cases} \{X\} \cup \texttt{knows}\ A\ \mathit{evs} & \text{if } A = \text{ Spy and } P \in \mathit{badP} \\ \texttt{knows}\ A\ \mathit{evs} & \text{otherwise} \end{cases}$$

7. An agent other than the Spy knows what she receives from her smartphone

$$\texttt{knows}\ A\ ((\texttt{AGets}\ A\ X)\ \#\ \mathit{evs}) \triangleq \begin{cases} \{X\} \cup \texttt{knows}\ A\ \mathit{evs} & \text{if } A = A' \text{ and } A \neq \mathit{Spy} \\ \texttt{knows}\ A\ \mathit{evs} & \text{otherwise} \end{cases}$$

When the assumption of secure devices hold, the definition of the *knows* predicate is simplified. Events `Scans` and `Outputs` do not need their correspondent reception events, since message reception is guaranteed. The `Scans` event is preserved, since it can only be exploited when the agent herself is compromised. Therefore, we only update the `Outputs` event form:

6. An agent, including the Spy, knows what her smartphone shows;

$$\texttt{knows}\ A\ ((\texttt{Shows}\ P\ A\ X)\ \#\ evs) \triangleq \begin{cases} \{X\} \cup \texttt{knows}\ A\ evs & \text{if } A = A' \\ \texttt{knows}\ A\ evs & \text{otherwise} \end{cases}$$

5.3 Threat Model

Traditional protocols could have their threat model defined by rule `Fake` and some other definitions concerning agents knowledge. The introduction of smartphone events and further modifications in knowledge sets motivate new kinds of aspects to be considered.

Smartphones can be exploited through illegal actions, as defined in Section 5.2.3. Our model does not allow the Spy to control the offline channel, but she has control over Smartphone inputs and outputs. Therefore, given an illegally usable smartphone, the Spy can send fake messages to it or can output fake messages to agents from her smartphone, pretending to be the compromised one. Such events should not be mixed with the ability of faking message in the regular network channel, in order to preserve behaviors linked to reception of messages from a specific channel.

Finally, rule `Fake` is extend in order to embrace such details, producing two more events besides the regular one. All other aspects concerning agents' knowledge sets are already described in Section 5.2.5.

5.4 Protocol Model

Once formalized, the model for smartphones guarantees the reception of messages for agents and devices, but only when the assumption of secure devices holds. For the case when it does not hold, we need two more rules that models the reception of data, similarly to the reception rule for network events, since rules are not forced to happen in the model. The `SRcpt` and `ARcpt` rules guarantee the reception of data for insecure devices. Both are defined at the protocol level.

$$\begin{aligned}&\texttt{SRcpt}:\\&[\![\mathit{evsRs} \in \mathit{protocol};\ \texttt{Scans}\ A\ (\text{Smartphone}\ B)\ X \in \mathit{set\ evsRs}]\!]\\&\Longrightarrow \texttt{SGets}\ B\ X\ \#\ \mathit{evsRs} \in \mathit{protocol}\end{aligned}$$

$$\begin{aligned}&\texttt{ARcpt}:\\&[\![\mathit{evsRa} \in \mathit{protocol};\ \texttt{Shows}\ (\text{Smartphone}\ A)\ B\ X \in \mathit{set\ evsRa}]\!]\\&\Longrightarrow \texttt{AGets}\ B\ X\ \#\ \mathit{evsRa} \in \mathit{protocol}\end{aligned}$$

5.5 Further Discussion

The model presented in this section is suitable for a reasonable analysis of the proposed protocol and its tools. The formalization introduced the mechanisms for agent-device communication, considering possible attacks from the Spy perspective.

However, such formalization has many concepts being grouped into a single model. The out-of-band entities, which are not properly embedded in the software aspects of the protocol, are modeled together with the technical details of the protocol. Even if the model can depict major parts of the whole protocol operation, some aspects may be left out, such as aspects regarding *social engineering attacks*.

The definition of a security protocol as a ceremony, involving both technical and social aspects, has already been done [40] and introductory formalization and further extensions of such aspects have been done in the Inductive Method [41]. The technique considers many out-of-band elements of the protocol, even the ones regarding human and computer interactions. The protocol is structured in layers, each representing one major aspect in the protocol. The agent is also split into different kind of layers, where interactions are held between two layers. Thus, a security property must transverse many layers until reaching the human entity to be considered as valid.

Additionally, this strategy can be extended to cover new entities, such as a new device that acts as a trusted device, but that can be exploited by a malicious peer, given this possible scenario. Hence, this approach seems to be more interesting for verifying a protocol like the DAP, given the better model granularity it provides. This task is delegated to future work.

Chapter 6

Protocol Verification

Finally, we present a partial formalization and verification of the DAP. The key generation was left out, due to time constraints and lack of explicitness in the protocol document concerning the asymmetric phase.

As discussed, it is possible to consider the situation where the user smartphone can be remotely exploited by the Spy and the one where the devices are secured. The formalization focus on the former, being faithful to the initial protocol assumptions. With this approach, any innate flaws in the protocol can be early found. Properties can be progressively tested as we build more threatening scenarios.

The proof scripts regarding the formalization are partially described, where only relevant sections are included. The complete model with its proved properties and auxiliary lemmas are attached as appendix at the end of this document. They are also available at [42].

6.1 Message Transaction

The protocol model states that no devices can be compromised, hence the `secureP` flag holds against all protocol possible traces. Most definitions concerning messages, events and the new additions related to the smartphone use are in the *Smartphone.thy* and *EventSP.thy* theories.

An observation about legal agents' initial knowledge set is necessary. In this protocol, legal agents do not own any knowledge, specially any necessary key for secure communication, since this is the base security premise of the DAP. Hence, we update the `initState` function, defined in Chapter 3, altering its definition on legal agents.

2. Legal agents do not own their private keys or any data. They initial knowledge set is empty:

$$\mathit{initState}(\text{Friend } i) \triangleq \{\}$$

$\texttt{Nil} : [\,] \in \mathit{sdaptrans}$

$\texttt{Rcpt} :$
$[\![\mathit{evs} \in \mathit{sdaptrans};\ \texttt{Says}\ A\ B\ X \in \mathit{set\ evs}]\!]$
$\Longrightarrow \texttt{Gets}\ B\ X\ \#\ \mathit{evs} \in \mathit{sdaptrans}$

Figure 6.1: Secure DAP message transaction base case and reception

The constant `sdaptrans` denotes the secured DAP model, consisting of the inductive set of protocol rules. Figure 6.1 illustrates the basic rules for the protocol operation. Rule *Nil* defines the protocol base case and rule *Rcpt* demonstrates how an agent can receive a message sent in the network.

The protocol rules are now presented. Rule *DT1* represents an agent sending the intended transaction for authorization to the Server. The Server cannot be such agent, which leads to the inability of the Server to start the protocol. Following lemmas will state that the Servers cannot use a smartphone.

It is important to stress the first adaptation of the protocol entities. A transaction is defined as the concatenation of the sender's identity and a number, representing the transaction itself. This representation is used due to its simplicity and comprehensiveness, since a banking transaction may take many forms and an arbitrary number of fields, but it is certain that a transaction will have an originator and an identification [6].

$\texttt{DT1} :$
$[\![\mathit{evs}1 \in \mathit{sdaptrans};\ A \neq \text{Server}]\!]$
$\Longrightarrow \texttt{Says}\ A\ \text{Server}\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\}\ \#\ \mathit{evs}1 \in \mathit{sdaptrans}$

$\texttt{DT2} :$
$[\![\mathit{evs}2 \in \mathit{sdaptrans};\ \text{Nonce}\ r \notin\ \mathit{used\ evs}2;$
$\texttt{Gets}\ \text{Server}\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ \mathit{evs}2$
$\Longrightarrow \texttt{Says}\ \text{Server}\ A\ \{\!|$
$\quad \{\!|\text{Agent}\ A, \text{Number}\ T|\!\},$
$\quad \text{Crypt}\ (\text{shrK}\ A)\ (\text{Nonce}\ r),$
$\quad \text{Crypt}\ (\text{shrK}\ A)\ (\{\!|\{\!|\text{Agent}\ A, \text{Number}\ T|\!\}, \text{Crypt}\ (\text{shrK}\ A)\ (\text{Nonce}\ r)|\!\})$
$|\!\}\ \#\ \mathit{evs}2 \in \mathit{sdaptrans}$

Figure 6.2: Secure DAP rule 1 and 2 formalizations

Rule *DT2* has more complex premises. In this rule, the Server instantiates the TAN, which must be fresh. Moreover, the Server must be triggered in order to respond an initiator, thus we include the reception of a transaction by the Server.

Finally, the Server uses the symmetric key both for generating the cipher with the TAN and the checksum hash for the offline phase. With the received transaction, the Server can respond the initial sender with the message m'. Note that we have another adaptation of the protocol: we do not distinguish the keys for encryption and hash creation. Since they are produced from just taking a part from the original key, the Spy could have access to both keys k_1 and k_2 if she has access to the shared key K.

Rule *DT3* concerns the phase where the agent inputs data to her smartphone using its camera. Such action is preceded by the issue of a transaction and reception of the message m'. Also, the agent must be legally capable of using her smartphone, that is, it must not be stolen.

Since the agent does not know her shared key, it cannot understand the contents of the ciphers. Therefore, such entities are hidden and are represented using the r' and h_s messages.

$$
\begin{array}{l}
\texttt{DT3}: \\
[\![evs3 \in sdaptrans;\ \text{legalUse}((\text{Smartphone } A)); \\
\quad \texttt{Says } A \text{ Server } \{\!|\text{Agent } A, \text{Number } T|\!\} \in \texttt{set } evs3; \\
\quad \texttt{Gets } A\ \{\!|\{\!|\text{Agent } A, \text{Number } T|\!\}, r', h_S|\!\} \in \texttt{set } evs3]\!] \\
\Longrightarrow \texttt{Inputs } A \text{ (Smartphone } A)\ \{\!|\{\!|\text{Agent } A, \text{Number } T|\!\}, r', h_S|\!\} \ \#\ evs3 \in sdaptrans
\end{array}
$$

Figure 6.3: Secure DAP rule 3 formalization

Rule *DT4* happens inside the smartphone. Here, the agent also must be holding her smartphone. In this stage, two important security steps are performed: the TAN decryption and the hash checksum. Both actions are formalized in the `Scans` event, where both ciphers are presented in clear, stating that the smartphone must precisely know the format of the received message. Hence, in order to present the received transaction on its screen, the smartphone must be fed with the expected input, that is, the equivalent hash representing the transaction and encrypted TAN.

Rule *DT5* models the agent confirming the visualized transaction in her smartphone screen. For this reason, the agent must have issued a transaction, received it again with its correspondent ciphers from the Server seen the same transaction, given by her in *DT3*, on the device screen. Again, we adapted the protocol: the confirmation act is

$\texttt{DT4}:$

$$\begin{array}{l}
[\![evs4 \in sdaptrans;\ \text{legalUse}((\text{Smartphone } A)); \\
\quad \texttt{Scans}\ A\ (\text{Smartphone } A)\ \{\!|, \\
\qquad \{\!|\text{Agent } A, \text{Number } T|\!\} \\
\qquad \text{Crypt (shrK } A)\ (\text{Nonce } r), \\
\qquad \text{Crypt (shrK } A)\ (\{\!|\{\!|\text{Agent } A, \text{Number } T|\!\}, \text{Crypt (shrK } A)\ (\text{Nonce } r)|\!\}) \\
\quad \in \texttt{set}\ evs4]\!] \\
\Longrightarrow \texttt{Shows}\ (\text{Smartphone } A)\ A\ \{\!|\text{Agent } A, \text{Number } T|\!\}\ \#\ evs4 \in sdaptrans
\end{array}$$

Figure 6.4: Secure DAP rule 4 formalization

represented with the `Inputs` event, using the transaction itself as an acknowledgment that the presented transaction is correct.

$\texttt{DT5}:$

$$\begin{array}{l}
[\![evs5 \in sdaptrans;\ \text{legalUse}((\text{Smartphone } A)); \\
\quad \texttt{Says}\ A\ \text{Server}\ \{\!|\text{Agent } A, \text{Number } T|\!\} \in \texttt{set}\ evs5 \\
\quad \texttt{Gets}\ A\ \{\!|\{\!|\text{Agent } A, \text{Number } T|\!\}, r', h_S|\!\} \in \texttt{set}\ evs5 \\
\quad \texttt{Scans}\ A\ (\text{Smartphone } A)\ \{\!|\{\!|\text{Agent } A, \text{Number } T|\!\}, r', h_S|\!\} \in \texttt{set}\ evs5 \\
\quad \texttt{Shows}\ (\text{Smartphone } A)\ A\ \{\!|\text{Agent } A, \text{Number } T|\!\} \in \texttt{set}\ evs5 \\
\Longrightarrow \texttt{Inputs}\ (\text{Smartphone } A)\ A\ \{\!|\text{Agent } A, \text{Number } T|\!\}\ \#\ evs5 \in sdaptrans
\end{array}$$

Figure 6.5: Secure DAP rule 5 formalization

Next, rule `DT6` models the smartphone presenting the decrypted TAN to the agent. In this case, we need that the smartphone had received the encrypted TAN and the correspondent hash and had the received transaction confirmed by the agent. Having these fulfilled, the smartphone can present the nonce.

Rule *DT7* starts the last stage of the protocol. Having issued and confirmed the transaction and received the correspondent TAN, the user can send it to the Server, in order to authorize the transaction at the bank side. Again, the agent does not know what the ciphers represent and the progress of the protocol should not rely in this fact.

In the last step, rule *DT8*, the Server acknowledges the transaction authorization if, and only if, the received TAN matches the one generated at early stages. Here, another adjustment takes place in this acknowledge message: we use the full transaction as a

$\texttt{DT6}:$

$\llbracket evs6 \in sdaptrans;\ \text{legalUse}((\text{Smartphone}\ A));$

$\quad \texttt{Scans}\ A\ (\text{Smartphone}\ A)\ \{\!|$

$\qquad \{\!|\text{Agent}\ A, \text{Number}\ T|\!\}$

$\qquad \text{Crypt}\ (\text{shrK}\ A)\ (\text{Nonce}\ r),$

$\qquad \text{Crypt}\ (\text{shrK}\ A)\ (\{\!|\{\!|\text{Agent}\ A, \text{Number}\ T|\!\}, \text{Crypt}\ (\text{shrK}\ A)\ (\text{Nonce}\ r)|\!\})$

$\quad \in \texttt{set}\ evs6$

$\quad \texttt{Shows}\ (\text{Smartphone}\ A)\ A\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ evs6$

$\quad \texttt{Inputs}\ A\ (\text{Smartphone}\ A)\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ evs6\rrbracket$

$\Longrightarrow \texttt{Shows}\ (\text{Smartphone}\ A)\ A\ \text{Nonce}\ r\ \#\ evs6 \in sdaptrans$

Figure 6.6: Secure DAP rule 6 formalization

$\texttt{DT7}:$

$\llbracket evs7 \in sdaptrans;\ \text{legalUse}((\text{Smartphone}\ A));$

$\quad \texttt{Says}\ A\ \text{Server}\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ evs7$

$\quad \texttt{Gets}\ A\ \{\!|\{\!|\text{Agent}\ A, \text{Number}\ T|\!\}, r', h_S|\!\} \in \texttt{set}\ evs7$

$\quad \texttt{Scans}\ A\ (\text{Smartphone}\ A)\ \{\!|\{\!|\text{Agent}\ A, \text{Number}\ T|\!\}, r', h_S|\!\} \in \texttt{set}\ evs7$

$\quad \texttt{Shows}\ (\text{Smartphone}\ A)\ A\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ evs7$

$\quad \texttt{Inputs}\ A\ (\text{Smartphone}\ A)\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ evs7$

$\quad \texttt{Shows}\ (\text{Smartphone}\ A)\ A\ (\text{Nonce}\ r) \in \texttt{set}\ evs7$

$\Longrightarrow \texttt{Says}\ A\ \text{Server}\ (\text{Nonce}\ r)\ \#\ evs7 \in sdaptrans$

Figure 6.7: Secure DAP rule 7 formalization

confirmation token, sent by the Server to the intended user. Finally, the protocol is concluded.

6.1.1 Threats

Three extra rules represents the threat scenario, modeling how the Spy can act in the protocol. The first one, *Fake* uses a similar structure from the one presented in Chapter 3, but also covering the smartphone case. In this rule, having all the entities to fake a message, the Spy can send it in the network or input it to another smartphone.

$$
\begin{array}{l}
\texttt{DT8}: \\
[\![evs8 \in sdaptrans; \\
\quad \texttt{Gets}\ \text{Server}\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\} \in \texttt{set}\ evs8; \\
\quad \texttt{Says}\ \text{Server}\ A\ \{\!| \\
\qquad \{\!|\text{Agent}\ A, \text{Number}\ T|\!\}, \\
\qquad \text{Crypt}\ (\text{shrK}\ A)\ (\text{Nonce}\ r), \\
\qquad \text{Crypt}\ (\text{shrK}\ A)\ (\{\!|\{\!|\text{Agent}\ A, \text{Number}\ T|\!\}, \text{Crypt}\ (\text{shrK}\ A)\ (\text{Nonce}\ r)|\!\}) \\
\qquad |\!\} \in \texttt{set}\ evs8 \\
\quad \texttt{Gets}\ \text{Server}\ (\text{Nonce}\ r)]\!] \\
\Longrightarrow \texttt{Says}\ \text{Server}\ A\ \{\!|\text{Agent}\ A, \text{Number}\ T|\!\}\ \#\ evs8 \in sdaptrans
\end{array}
$$

Figure 6.8: Secure DAP rule 8 formalization

$$
\begin{array}{l}
\texttt{Fake}: \\
[\![evsF \in sdaptrans; \text{illegalUse}(\text{Smartphone}\ A); \\
X \in \text{synth}(\text{analz}(\text{knows Spy}\ evsF)) \\
\ \Longrightarrow \texttt{Says}\ \text{Spy}\ B\ X\ \# \\
\texttt{Scans}\ Spy\ (\text{Smartphone}\ A)\ X\ \#\ evsF \in sdaptrans
\end{array}
$$

Figure 6.9: Spy general behavior rule for Secure DAP

Still, it is necessary to model how the Spy can exploit illegally usable smartphones. This behavior is modeled with rules *DT4_Fake* and *DT6_Fake*. The first covers the case where the Spy can obtain a transaction displayed by a smartphone, in order to be confirmed by an user. The `illegalUse` predicate is defined according to the `secureP` flag. Also, the Spy must have presented message m' to the smartphone, which means that she must have early access to the Smartphone, either by holding or exploiting it.

Rule *DT6_Fake* models how the smartphone can obtain the displayed TAN by a smartphone, after the user confirmation phase. As in rule *DT4_Fake*, the smartphone must be illegally used and the message m' must be given to the smartphone. Additionally, the confirmation phase must be performed and succeeded. Thereby, the Spy can obtain the deciphered than from the Smartphone.

```
DT4_Fake :
⟦evsf4 ∈ sdaptrans; illegalUse((Smartphone A));
   Scans Spy (Smartphone A) ⦃
      ⦃Agent A, Number T⦄
      Crypt (shrK A) (Nonce r),
      Crypt (shrK A) (⦃⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r)⦄)
   ∈ set evsf4⟧
⟹ Shows (Smartphone A) Spy ⦃Agent A, Number T⦄ # evsf4 ∈ sdaptrans
```

Figure 6.10: The Spy exploit for rule *DT4*

```
DT6_Fake :
⟦evs6 ∈ sdaptrans; illegalUse((Smartphone A));
   Scans Spy (Smartphone A) ⦃
      ⦃Agent A, Number T⦄
      Crypt (shrK A) (Nonce r),
      Crypt (shrK A) (⦃⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r)⦄)
   ∈ set evs6
   Shows (Smartphone A) Spy ⦃Agent A, Number T⦄ ∈ set evs6
   Inputs Spy (Smartphone A) ⦃Agent A, Number T⦄ ∈ set evs6⟧
⟹ Shows (Smartphone A) Spy Nonce r # evs6 ∈ sdaptrans
```

Figure 6.11: The Spy exploit for rule *DT6*

6.2 Model Reliability

We now discuss the general reliability properties about the protocol, concerning the Server, agents and the smartphones. The *evs* set will be considered as a valid protocol trace regarding the DAP model.

6.2.1 Server Guarantees

Theorem 6.2.1 shows that when fed with a transaction, the Server outputs the expected m' message, that is, the Server works reliably. Such guarantee is not verifiable by the initiator agent, because she cannot inspect the form of the ciphers nor can reliably inspect who sent her the message m'.

Theorem 6.2.1 (**Says_Server_DT2**). *If evs contains*

$$\begin{array}{l}\texttt{Says}\ \mathit{Server}\ A\ \{\!|\\ \quad \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\},\\ \quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})\\ \quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}, \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})|\!\})\\ |\!\}\end{array}$$

Then, evs also contains

$$\texttt{Gets}\ \mathit{Server}\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}$$

It is also guaranteed that the Server cannot initiate the protocol, which is show in Theorem 6.2.2. Hence, we can prove reliability lemmas which state that the Server never use its smartphone or anyone uses her smartphone with the Server at any protocol stage.

Theorem 6.2.2 (**Server_cannot_initiate**). *If evs contains*

$$\texttt{Says}\ A\ \mathit{Server}\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}$$

Then, $A \neq \mathit{Server}$

Finally, Theorem 6.2.3 guarantees the whole authentication process at the Server side, stating that it can only send a success message - represented by the transaction - if it received the transaction prior and the produced TAN matches the received one. This property is verifiable only at the Server side, once it is the only entity that posses the original TAN.

Theorem 6.2.3 (**Says_Server_DT8**). *If, A is not the Server and evs contains*

$$\texttt{Says}\ \mathit{Server}\ A\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}$$

Then, there is an r where evs also contains

$$\begin{array}{l}\texttt{Gets}\ \mathit{Server}\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\},\\ \texttt{Says}\ \mathit{Server}\ A\ \{\!|\\ \quad \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\},\\ \quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})\\ \quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}, \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})|\!\})\\ \texttt{Gets}\ \mathit{Server}\ \mathit{Nonce}\ r\end{array}$$

6.2.2 Smartphone Use

Lemma 6.2.1 states that if an agent other than the Spy uses a smartphone, it must be legally used and the device must be hers, guaranteeing that agent only uses theirs smartphones legally.

Lemma 6.2.1 (Scans_Shows_Smartphone). *If A is not the Spy and evs contains*

$$\texttt{Scans}\ A\ P\ X\ \textit{or}\ \texttt{Shows}\ P\ A\ X$$

Then, P = (Smartphone A) and legalUse(P)

The Spy can act in two different ways. She can only legally use her own smartphone, since it cannot be compromised, and any other smartphone that belongs to an agent is illegally usable by her. Theorem 6.2.2 confirms that.

Lemma 6.2.2 (Scans_Smartphone_Spy). *If evs contains*

$$\texttt{Scans}\ Spy\ P\ X\ \textit{or}\ \texttt{Shows}\ P\ Spy\ X$$

Then, (P = (Smartphone A) and legalUse(A)) or, for some A, (P = (Smartphone A) and illegalUse(A))

6.2.3 Smartphone Outputs

The way how a smartphone produce outputs concerns how they interact with other peers. First, the proof focuses on how smartphones depend on the correct inputs to give the expected outputs, removing any unlimited power from them, specially if it in the Spy's possession. Therefore, such lemmas concern rules where the `Shows` events are appended to the trace.

We use Rule *DT4* as an example. Lemma 6.2.4 demonstrates the conditions that must be fulfilled in order to a smartphone present the received transaction on its screen for further confirmation.

Theorem 6.2.4 (Shows_which_Smartphone_4). *If evs contains*

$$\texttt{Shows}\ (Smartphone\ A)\ A\ \{\!|Agent\ A, Number\ T|\!\}$$

Then evs also contains

$$\begin{aligned}
&\texttt{Scans}\ A\ (Smartphone\ A)\ \{\!|\\
&\quad \{\!|Agent\ A, Number\ T|\!\},\\
&\quad Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})\\
&\quad Crypt\ (shrK\ A)\ (\{\!|\{\!|Agent\ A, Number\ T|\!\}, Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})|\!\})\\
&|\!\}
\end{aligned}$$

A second version of the this lemma defines a stronger scenario, where the agent cannot be the Spy. This guarantees not only that the smartphone received the proper inputs but that a legal use of the user smartphone was performed, by its owner. Finally, the strongest version, presented in Theorem 6.2.5, consider the case for an arbitrary smartphone. In this case, it is still provable the ownership and legality on the smartphone use. Thus, it is proved that a smartphone only shows a transaction confirmation message to its owner and depends on the correct deciphering of the TAN and hash checksum.

Theorem 6.2.5 (Shows_A_Smartphone_4). *If evs contains*

$$\texttt{Shows}\ P\ A\ \{\!|Agent\ A, Number\ T|\!\}$$

Then, P is A's smartphone, it has been legally used and, for a given r, the set evs contains

$$\begin{aligned}
&\texttt{Scans}\ A\ (Smartphone\ A)\ \{\!|\\
&\quad \{\!|Agent\ A, Number\ T|\!\},\\
&\quad Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})\\
&\quad Crypt\ (shrK\ A)\ (\{\!|\{\!|Agent\ A, Number\ T|\!\}, Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})|\!\})\\
&|\!\}
\end{aligned}$$

Rule *DT6* follows the same principle. However, the strongest version of the reliability property cannot be proved for this rule. Given *DT6_Fake*, it is not possible to assure that a legal agent performed the transaction confirmation, since the Spy can steal or exploit the device. Hence, the strongest provable form of the smartphone operation property is illustrated in Theorem 6.2.6, where the smartphone ownership and smartphone reliable operation can only the guaranteed when it has been legally used and not compromised.

Theorem 6.2.6 (Shows_uncompromised_A_Smartphone_6). *If $P \notin badP$, legalUse(P) and evs contains*

$$\texttt{Shows}\ P\ A\ Nonce\ r$$

Then, P = (Smartphone A) and, for some T, evs also contains

$$\texttt{Inputs}\ A\ P\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}$$

Another type of guarantees are the ones related to the format of the messages in such events, confirming that the smartphone can reliably build the protocol messages, working correctly. The idea is that when the agent provides a specific input, the device will be able to correctly derive the message. For the `Shows` events, the proofs are pretty straightforward. The only difference between the two rules *DT4* and *DT6* lies in the number of components in the displayed messages of these two rules.

6.2.4 Smartphone Inputs

The proofs for smartphone inputs also have the two cited categories. Here, both the *Scans* and the *Inputs* events are considered. Like the weaker guarantees for Rule *DT6*, Rule *DT3* is another rule that suffers from the Spy threats. Since the smartphone can be robbed, a Spy could input a desired transaction in the stolen phone. Therefore, even if the event of a smartphone scanning the message m' is detected within a trace, it is not obvious that a legal use the smartphone was performed. Theorem 6.2.7 is the strongest guarantee for such event, where the premises include the fact that the agent performing the scanning action is not the Spy. In this case, a legal use of the agent smartphone can be assured.

Theorem 6.2.7 (Scans_A_honest_Smartphone_3). *If A is not the Spy and evs contains*

$$\texttt{Scans}\ A\ P\ \{\!|\{\!|\mathit{Agent}\ A, \mathit{Numbert}\ T|\!\}, r', h_S|\!\}$$

Then, P = (Smartphone A), legalUse(P) and evs contains

$$\texttt{Says}\ A\ \mathit{Server}\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}$$
$$\texttt{Gets}\ A\ \{\!|\{\!|\mathit{Agent}\ A, \mathit{Numbert}\ T|\!\}, r', h_S|\!\}$$

Rule *DT5* has a special meaning: it acts as the confirmation step for the protocol, since it represents the user confirmation. This message is preceded by important conditions such as the checksum verification, TAN deciphering and the smartphone displaying the received transaction to the user.

Theorem 6.2.8 (Inputs_A_Smartphone_5). *If evs contain*

$$\texttt{Inputs}\ A\ P\ \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}$$

Then, P = (Smartphone A), legalUse(A) and evs contains

$$
\begin{aligned}
&\texttt{Scans}\ A\ (\mathit{Smartphone}\ A)\ \{\!|\\
&\quad \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\},\\
&\quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})\\
&\quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}, \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})|\!\})\\
&|\!\}
\end{aligned}
$$

Theorem 6.2.8 proves that if the event of an agent confirming a transaction on a smartphone exists, then she must been acting legally and her smartphone have confirmed that the received transaction is authentic. However, it is important to note that the agent itself does not participate in any of these stages directly, since she only knows the transaction among all the previous messages, which goes according to the DAP specification.

Regarding message format guarantees, it is possible to find some issues with Rule *DT3*. Once an agent receives message m' from the Server, she cannot inspect the format of the message. Thus, the agent does not have any guarantees about that message, that is, the message is not explicit about its contents. Theorem 6.2.9 shows that the contents of the message that the agent forwards to its smartphone cannot be derived, since it is not conclusive what each message entity represents.

Theorem 6.2.9 (Scans_Smartphone_A_DT3_form_unprovable). *If evs contains*

$$\texttt{Scans}\ A\ (\mathit{Smartphone}\ A)\ \{\!|\{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}, r', h_S|\!\}$$

Then, for some nonce r', there is a nonce r that $r' \neq \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\mathit{Nonce}\ r)$

Lemmas about Rule *DT5* format are straightforward, once it is the only *Inputs* event in the protocol and its message form is the transaction itself.

6.2.5 Regularity

The DAP message transaction phase does not send any long-term keys over the network. Therefore, such keys are only present in the network if the Spy is sending then. In order to do so, the Spy must know such keys prior to the protocol execution, which means that she must have access to the legal agent smartphone, which retains the user long-term key. She cannot obtain the key from the Server, since it is a secure entity. At the same time, if the agent smartphone is compromised, the Spy can send its stored long-term key in the network. Theorem 6.2.10 is a message regularity guarantee that expresses this situation.

Theorem 6.2.10 (Spy_analz_shrK). *Trace evs is such that*

$$Key\ (shrK)\ A \in analz(knows\ Spy\ evs) \iff (SmartphoneA) \in badP$$

6.3 Model Properties

In this section, the verified model properties are described. These properties relate to the protocol guarantees and it is a partial formalization. More work on confidentiality and authorization guarantees must be made, but the results are satisfying since they explore some core features in the protocol.

6.3.1 Confidentiality and Privacy

It is observable that the DAP does not preserve the transaction privacy. Theorem 6.3.1 shows that the Spy possess the transaction number after the first step of the protocol. It is trivial to show this, since all messages that contains the transaction are sent in clear text over the network.

Theorem 6.3.1 (Spy_knows_transaction). *If evs contains*

$$\texttt{Says}\ A\ Server\ \{\!|Agent\ A, Number\ T|\!\}$$

Then, Number T ∈ analz(knows Spy evs)

Considering the best practices for a banking security framework [6], privacy is a property that is usually desired in online banking protocols. Moreover, the argument that the transaction may be protected by an higher level communication channel does not ensure the privacy property, since it is given that the user personal computer is compromised and, thus, the Spy can access the transaction again.

The confidentiality properties are bounded to the use of the pre-shared key K in the protocol. Confidentiality lemmas are related to the correct use of the given key and how it can produce correct ciphers. Therefore, the lemmas regarding the correct form of the produced messages, both by the Server and the agents, relate to confidentiality.

6.3.2 Authenticity

Authenticity lemmas cannot be verified by agents, since all ciphered entities are sealed with the long-term keys which are stored only at the user smartphone and the server. Still, it is possible for agents to indirectly obtain such guarantees from their smartphones.

Theorem 6.3.2 shows how the ciphers produced by the Server acts as an authenticity token for the user smartphone. Since they are sealed with the pre-shared key, owned by the smartphone and the server, they can only be generated by one of these entities. Once message m' travels through the network, its format can be analyzed and proved to be originated by the server.

At the smartphone side, the guarantee that message m' is authentic is confirmed by the use of the shared key K. However, it is noted that this guarantee depends on the assumption that devices cannot be remotely exploited. If the Spy could have access to the agent's smartphone, she could impersonate the server and fake the message m'.

Theorem 6.3.2 (Ciphers_authentic). *If* $(SmartphoneA) \notin badP$ *and*

$$Crypt\ (shrKA)\ (\{\!|\{\!|Agent\ A, Number\ T|\!\}, r', h_S|\!\}) \in analz(knows\ Spy\ evs)$$

Then,

$r' = Crypt\ (shrKA)\ (Nonce\ r)$

$h_S = Crypt\ (shrKA)\ (\{\!|\{\!|Agent\ A, Number\ T|\!\}, r'|\!\})$

and evs contains

$\mathtt{Says}\ Server\ A\ \{\!|$

$\quad \{\!|Agent\ A, Number\ T|\!\},$

$\quad Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})$

$\quad Crypt\ (shrK\ A)\ (\{\!|\{\!|Agent\ A, Number\ T|\!\}, Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})|\!\})$

$|\!\}$

Theorem 6.3.3 demonstrates how the server correctly links the TAN and hash h_S to the correct agent, proving that it can only authorize transactions to known peers. Thus, if the server responds to an agent, it must know the pre-shared key K belonging to such agent.

Theorem 6.3.3. *If evs contains*

$\mathtt{Says}\ Server\ A'\ \{\!|$

$\quad \{\!|Agent\ A, Number\ T|\!\},$

$\quad Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})$

$\quad Crypt\ (shrK\ A)\ (\{\!|\{\!|Agent\ A, Number\ T|\!\}, Crypt\ (shrK\ A)\ (\{\!|Nonce\ r|\!\})|\!\})$

$|\!\}$

Then, $A = A'$

6.3.3 Unicity

The TAN is the entity that uniquely identifies a transaction. Hence, it is important that such entity preserves the unicity property. The Server must guarantee that if two TAN are equal, then they are referencing the same transaction. Theorem 6.3.4 describes that if two messages in the network contain the same TAN, then they must be carrying the same transaction to the same agent. This property cannot the verified by any agent, since they cannot read the Server's message ciphered content.

Theorem 6.3.4 (Server_TAN_Unique). *If evs contains*

$$
\begin{array}{l}
\texttt{Says}\ \mathit{Server}\ A\ \{\!|\\
\quad \{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\},\\
\quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})\\
\quad \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\{\!|\mathit{Agent}\ A, \mathit{Number}\ T|\!\}, \mathit{Crypt}\ (\mathit{shrK}\ A)\ (\{\!|\mathit{Nonce}\ r|\!\})|\!\})\\
|\!\}\\
\texttt{Says}\ \mathit{Server}\ A'\ \{\!|\\
\quad \{\!|\mathit{Agent}\ A', \mathit{Number}\ T'|\!\},\\
\quad \mathit{Crypt}\ (\mathit{shrK}\ A')\ (\{\!|\mathit{Nonce}\ r|\!\})\\
\quad \mathit{Crypt}\ (\mathit{shrK}\ A')\ (\{\!|\{\!|\mathit{Agent}\ A', \mathit{Number}\ T|\!\}, \mathit{Crypt}\ (\mathit{shrK}\ A')\ (\{\!|\mathit{Nonce}\ r|\!\})|\!\})\\
|\!\}
\end{array}
$$

Then, $A = A'$ and $T = T$'

6.4 Discussion

As seen in this chapter, the smartphone model provided a base for reasoning about the devices operation in a protocol that included offline communication between agents and their devices. The model allowed the formalization of the protocol, considering the threat scenarios of device theft.

It is important to notice that the smartphone provides more functionalities than necessary, introducing the concern of possible attacks on the device, breaking the security premises of the protocol. in this first version, the formalization did not consider such context, reasoning how the protocol could provide its promised properties. In the end, the protocol proved to be consistent in many of its proposed characteristics.

However, it is important to notice that some guarantees are not verifiable by all peers. The guarantees concerning the messages that arrive at the agent's personal computer can not be verified by her, since she does not understand the ciphered entities of the message. Yet, this seems to be a crucial aspect to the protocol security: since the agent cannot understand such messages, it cannot disclose them as well to any the Spy. The protocol security heavily relies in that fact.

Chapter 7

Discussion and Conclusion

This chapter brings a brief discussion about the results, challenges and future work regarding the DAP formalization and verification. The resulting work demanded the understanding of security protocols, formal verification and the DAP specification itself.

7.1 Results

The protocol could be totally formalized. There are lemmas confirming the model reliability regarding the protocol, smartphones, agents and server operation, where the threat scenario built by the Spy is considered. No entity can operate under unrealistic scenarios such as having unlimited power to produce inconceivable messages.

Regarding its properties, it was found that the DAP has issues with privacy, where the transaction issued by the user goes on clear text through the network. Since it is a banking protocol, it seems odd that the protocol does not addresses such concern.

However, the protocol fulfills great part of its goals. The Transaction Authorization Nonce (TAN) uniquely identifies transactions and acts as a valid token for the server to confirm that a transaction was effectively confirmed at the smartphone. Additionally, the produced ciphers at the server — the encrypted TAN and checksum hash — are used at the smartphone side to confirm that a trusted entity generated both of them, providing authenticity. At the same time, the integrity of such messages is guaranteed by the use of the long-term shared key K between the smartphone user and the server.

The guarantees at the smartphone are condensed in the reliability properties of the model. These lemmas provide proofs that the protocol can legally move on when important steps were taken. In particular, the deciphered TAN is only presented in the smartphone screen if the checksum phase succeeded and the user confirmed the transaction at her smartphone.

Lemmas regarding rules $DT3$ and $DT6$ presented issues, where strong lemmas could not be proved, specially when considering the situation of smartphone theft. In this case, the Spy could input arbitrary data in the stolen phone and confirm any desired transaction.

7.2 Challenges

The use of the Inductive Method proved to be a reasonable choice, considering it provided a good framework that could be extended in order to model smartphones and its operation. The obtained model was able to represent different kinds of events and the smartphone operation with its available secrets. The sets of compromised ($badP$) and stolen ($stolen$) phones could reliably portray the situation where a malicious agent interferes in the protocol, affecting the devices according. However, for the targeted protocol, it was necessary to conceive some rules ($DT4\_Fake$ and $DT6\_Fake$) which described how the Spy could obtain more information through some events, where such rules have the precondition of an illegal use of the smartphone.

The initial protocol formalization did not consider that smartphones could be remotely exploited. With only this scenario, many properties could be explored and some were left out.

It was noted that the lemmas regarding the third rule of the model ($DT3$) presented some issues, where strong properties could not be proved. This event carries a concern: it is the phase where the User inputs the main message to her smartphone, without understanding its meaning due to encryption. Hence, it is difficult to extract properties which can be enjoyed by the user, since many of the security aspects available at the message are not clear to her. However, it is where the protocol's major security premises lies. Since the agent cannot understand what she is receiving, she cannot disclose such messages to malicious peers, but only her smartphone. In a context where the mobile devices cannot be exploited, this conserves the security properties of the protocol.

Also, on further rules, there are more interactions between the user and the smartphone that also presented some trouble to be formalized and inspected, due to its hardship to be translated into formal rules, using the available framework. The security protocol ceremony [40] appears to be a more reliable path for formalizing the details in the interaction between a human and the smartphone devices, where such task is split in layers which can be properly formalized and have its particularities better identified. A security property is valid in the protocol when it transverses all layers. This could potentially bring better insights in the transaction confirmation phase at the smartphone.

7.3 Future Work

As discussed, some properties were left out from the verification. The main focus was given to basic properties and another ones regarding the protocol message contents.

One of the main tasks delegated to the future is the use of concepts from the security protocol ceremony, which could bring more interesting details about the phases where there are interactions between the user and her smartphone.

Additionally, the DAP formalization lacks the scenario where smartphones can be exploited. This is an important scenario since it corresponds to the real world. This formalization should take into account that the Spy can trigger any desired action in the smartphones, access their data and obtain any given information in this devices. In a more threatening scenario, some of the proved properties for the secured DAP version can become unprovable, since many of the protocol guarantees rely in the fact that the devices cannot be compromised.

Finally, the formalization of the shared key generation phase was left out, both for the symmetric and asymmetric schemes. The former may present more interesting details about the protocol and the key itself, since it uses the peculiarities of the DAP as the use of smartphone and message exchange base in QR-codes.

References

[1] Melo, Laerte Peotta de: *DAP (Dynamic Authorization Protocol): Uma Abordagem Segura Out-Of-Band Para E-Bank Com Um Segundo Fator De Autenticação Visual.* PhD thesis, University of Brasília, August 2012. `http://repositorio.unb.br/handle/10482/12963`. 1, 31, 32, 36, 41, 42

[2] Paulson, Lawrence C.: *The inductive approach to verifying cryptographic protocols.* Journal of Computer Security, 6(1-2):85–128, January 1998, ISSN 0926-227X. `http://dl.acm.org/citation.cfm?id=353677.353681`. 2, 15

[3] Bella, Giampaolo: *Formal Correctness of Security Protocols.* Springer, 2007, ISBN 978-3-540-68134-2. 2, 4, 6, 9, 14, 15, 19, 27, 28, 42

[4] Bella, Giampaolo and Lawrence C. Paulson: *Accountability protocols: Formalized and verified.* ACM Trans. Inf. Syst. Secur., 9(2):138–161, May 2006, ISSN 1094-9224. 2, 15

[5] Paulson, Lawrence C.: *Inductive analysis of the internet protocol tls.* ACM Trans. Inf. Syst. Secur., 2(3):332–351, August 1999, ISSN 1094-9224. `http://doi.acm.org/10.1145/322510.322530`. 2, 15

[6] Hutchinson, Damien and Matthew Warren: *Security for internet banking: a framework.* Journal of Enterprise Information Management, 16(1):64–73, 2003, ISSN 1741-0398. 2, 31, 52, 63

[7] Adham, Manal, Amir Azodi, Yvo Desmedt, and Ioannis Karaolis: *How to attack two-factor authentication internet banking.* 17th International Conference on Financial Cryptography and Data Security, pages 322–328, 2013. 2, 31, 32

[8] Starnberger, Guenther, Lorenz Froihofer, and Karl M. Göschka: *Qr-tan: Secure mobile transaction authentication.* Proceedings of the 4th International Conference on Availability, Reliability and Security, ARES 2009, pages 578–583, March 2009. 2, 32

[9] Lee, Young Sil, Nack Hyun Kim, Hyotaek Lim, HeungKuk Jo, and Hoon Jae Lee: *Online banking authentication system using mobile-otp with qr-code.* In *5th International Conference on Computer Sciences and Convergence Information Technology*, pages 644–648, Nov 2010. 2, 32

[10] Abadi, Martín: *Security protocols and their properties.* Foundations of Secure Computation, 26(1):39–60, Jan 1999. 4, 8

[11] Boyd, Colin and Anish Mathuria: *Protocols for Authentication and Key Establishment.* Springer, 2003rd edition, 2008, ISBN 3540431071,9783540431077. 4, 5, 6, 11, 12, 13, 14

[12] Ryan, Peter and Steve Schneider: *The Modelling and Analysis of Security Protocols: The Csp Approach.* Addison-Wesley Professional, 2nd edition, 2010, ISBN 0-201-67471-8. 4, 8, 15

[13] Dolev, Danny and Andrew C. Yao: *On the security of public key protocols.* In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society. 5, 19

[14] Burrows, Michael, Martin Abadi, and Roger Needham: *A logic of authentication.* ACM Trans. Comput. Syst., 8(1):18–36, February 1990, ISSN 0734-2071. 6, 14

[15] Gollmann, Dieter: *On the verification of cryptographic protocols.* Electronic Notes in Theoretical Computer Science, 32:42 – 58, 2000, ISSN 1571-0661. Workshop on secure architectures and information flow. 8, 9

[16] Diffie, Whitfield, Paul C. Van Oorschot, and Michael J. Wiener: *Authentication and authenticated key exchanges.* Des. Codes Cryptography, 2(2):107–125, June 1992, ISSN 0925-1022. `http://dx.doi.org/10.1007/BF00124891`. 8

[17] Anderson, Ross J.: *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley Publishing, 2nd edition, 2008, ISBN 9780470068526. 9

[18] Abadi, Martín and Roger Needham: *Prudent engineering practice for cryptographic protocols.* IEEE Trans. Softw. Eng., 22(1):6–15, January 1996, ISSN 0098-5589. `http://dx.doi.org/10.1109/32.481513`. 11

[19] Needham, Roger and Michael D. Schröeder: *Using encryption for authentication in large networks of computers.* Commun. ACM, 21(12):993–999, 1978, ISSN 0001-0782. 11, 14

[20] Lowe, Gavin: *Breaking and fixing the Needham-Schröeder public-key protocol using fdr.* In *TACAS*, pages 147–166, 1996. 12, 14

[21] Meadows, Catherine: *The nrl protocol analyzer: An overview.* The Journal of Logic Programming, 26:113–131, February 1996. 14, 15

[22] Meadows, Catherine: *Formal verification of cryptographic protocols: A survey.* In *Proceedings of the 4th International Conference on the Theory and Applications of Cryptology: Advances in Cryptology*, ASIACRYPT '94, pages 135–150, London, UK, UK, 1995. Springer-Verlag, ISBN 3-540-59339-X. `http://dl.acm.org/citation.cfm?id=647092.714095`. 14

[23] Kemmerer, Richard A., Catherine Meadows, and Jonathan K. Millen: *Three systems for cryptographic protocol analysis.* Journal of Cryptology, 7(2):79–130, June 1994, ISSN 0933-2790. `http://dx.doi.org/10.1007/BF00197942`. 14

[24] Bella, Giampaolo: *Inductive verification of smart card protocols.* J. Comput. Secur., 11(1):87–132, February 2003, ISSN 0926-227X. `http://dl.acm.org/citation.cfm?id=773065.773068`. 15

[25] Paulson, Lawrence C. and Tobias Nipkow: *Isabelle: A Generic Theorem Prover.* Springer, 1994. 16

[26] Paulson, Lawrence C.: *Isabelle hol-auth libraries*, 1993. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/library/HOL/HOL-Auth/index.html`, visited on 2017-10-08. 16

[27] Bellare, Mihir and Phillip Rogaway: *Entity authentication and key distribution.* In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 232–249, New York, NY, USA, 1994. Springer-Verlag New York, Inc., ISBN 0-387-57766-1. `http://dl.acm.org/citation.cfm?id=188105.188164`. 28

[28] Melo, Laerte Peotta de, Marcelo Holtz, Bernardo David, Flavio Deus, and Rafael de Sousa Junior: *A formal classification of internet banking attacks and vulnerabilities.* International Journal of Computer Science and Information Technology, 3, February 2011. 31, 32

[29] Shoup, Victor and Avi Rubin: *Session key distribution using smart cards.* In *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'96, pages 321–331, Berlin, Heidelberg, 1996. Springer-Verlag, ISBN 3-540-61186-X. `http://dl.acm.org/citation.cfm?id=1754495.1754534`. 33

[30] Rivest, Ronald L., Adi Shamir, and Leonard M. Adleman: *A method for obtaining digital signatures and public-key cryptosystems.* Commun. ACM, 21(2):120–126, February 1978, ISSN 0001-0782. `http://doi.acm.org/10.1145/359340.359342`. 35

[31] Bellare, Mihir, Ran Canetti, and Hugo Krawczyk: *Keying hash functions for message authentication.* In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 1–15, London, UK, UK, 1996. Springer-Verlag, ISBN 3-540-61512-1. `http://dl.acm.org/citation.cfm?id=646761.706031`. 37

[32] *Specification for the advanced encryption standard (aes).* Federal Information Processing Standards Publication 197, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`. 37

[33] Bonneau, J., C. Herley, P. C. v. Oorschot, and F. Stajano: *The quest to replace passwords: A framework for comparative evaluation of web authentication schemes.* In *2012 IEEE Symposium on Security and Privacy*, pages 553–567, May 2012. 41

[34] Claessens, Joris, Valentin Dem, Danny De Cock, Bart Preneel, and Joos Vandewalle: *On the security of today's online electronic banking systems.* Computers & Security, 21(3):253 – 265, 2002, ISSN 0167-4048. `http://www.sciencedirect.com/science/article/pii/S0167404802003127`. 41

[35] Newzoo: *Global mobile market report*. Technical report, Newzoo, 2018. 41

[36] Zhou, Y. and X. Jiang: *Dissecting android malware: Characterization and evolution*. In *2012 IEEE Symposium on Security and Privacy*, pages 95–109, May 2012. 42

[37] Han, Jin, Su Mon Kywe, Qiang Yan, Feng Bao, Robert Deng, Debin Gao, Yingjiu Li, and Jianying Zhou: *Launching generic attacks on ios with approved third-party applications*. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13, pages 272–289, Berlin, Heidelberg, 2013. Springer-Verlag, ISBN 978-3-642-38979-5. `http://dx.doi.org/10.1007/978-3-642-38980-1_17`. 42

[38] Rao, S. P., S. Holtmanns, I. Oliver, and T. Aura: *Unblocking stolen mobile devices using ss7-map vulnerabilities: Exploiting the relationship between imei and imsi for eir access*. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1171–1176, Aug 2015. 44

[39] *Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification*. Standard, International Organization for Standardization, Geneva, CH, February 2015. 44

[40] M. Ellison, Carl: *Ceremony design and analysis*. IACR Cryptology ePrint Archive, 2007:399, January 2007. 50, 68

[41] Bella, Giampaolo and Lizzie Coles-Kemp: *Layered analysis of security ceremonies*. In *IFIP Advances in Information and Communication Technology*, volume 376, pages 273–286, June 2012. 50

[42] *Formal verification of the dynamic authorization protocol*. Github, 2019. `https://github.com/rodopoulos/dap-verification`. 51

Appendix A

Smartphone Model Proof Script

```
theory Smartphone imports "./EventSP" "~~/src/HOL/Auth/All_Symmetric" begin


axiomatization
  shrK  ::  "agent ⇒ key"

where
  inj_shrK : "inj shrK"
```

A.1 Smartphone Usage

```
definition legalUse :: "smartphone ⇒ bool" ("legalUse (_)") where
  "legalUse P == P ∉ stolen"


primrec illegalUse :: "smartphone ⇒ bool" where
  illegalUse_def: "illegalUse (Smartphone A) = (
    (insecureP ∧ (Smartphone A ∈ stolen) ∨ (Smartphone A ∈ badP)) ∨
    (secureP   ∧ (Smartphone A ∈ stolen)))"
```

A.2 Agents' Initial State

```
overloading initState ≡ initState
  begin
  primrec initState where
```

```
    initState_Server : "initState Server = (Key‘ (range shrK))" |
    initState_Friend : "initState (Friend i) = {}" |
    initState_Spy : "initState Spy = (Key‘ (shrK‘ {A. Smartphone A ∈ badP}))"
end

axiomatization where
  Key_supply_ax: "finite KK ⟹ ∃ K. K ∉ KK & Key K ∉ used evs " and


  Nonce_supply_ax: "finite NN ⟹ ∃ N. N ∉ NN & Nonce N ∉ used evs"
```

A.3 Shared Key Properties

```
declare inj_shrK [THEN inj_eq, iff]


lemma invKey_K [simp]: "invKey K = K"
apply (insert isSym_keys)
apply (simp add: symKeys_def)
done


lemma analz_Decrypt' [dest]:
  "⟦ Crypt K X ∈ analz H; Key K ∈ analz H ⟧ ⟹ X ∈ analz H"
apply (erule analz.Decrypt)
apply (simp)
done


declare analz.Decrypt [rule del]


lemma parts_image_Nonce [simp] :
  "parts (Nonce‘ N) = Nonce‘ N"
by auto


lemma keysFor_parts_initState [simp] :
  "keysFor (parts (initState C)) = {}"
```

apply *(unfold keysFor_def)*
apply *(induct_tac "C", auto)*
done

lemma *keysFor_parts_insert :*
"⟦ K ∈ keysFor (parts (insert X G)); X ∈ synth (analz H) ⟧
⟹ K ∈ keysFor (parts (G ∪ H)) | Key K ∈ parts H"
by *(force dest: EventSP.keysFor_parts_insert)*

lemma *Crypt_imp_keysFor :*
"Crypt K X ∈ H ⟹ K ∈ keysFor H"
by *(drule Crypt_imp_invKey_keysFor, simp)*

lemma *shrK_in_initState [iff]: "Key (shrK A) ∈ initState Server"*
apply *(induct_tac "A")*
apply *simp_all*
done

lemma *shrK_not_in_other [iff]: "Key (shrK (Friend x)) ∈ initState (Friend y) ⟹ (x = y)"*
apply *simp*
done

lemma *shrK_in_used [iff]: "Key (shrK A) ∈ used evs"*
apply *(rule initState_into_used)*
apply *blast*
done

If a key is fresh, then it must not a long-term key

lemma *Key_not_used [simp]: "Key K ∉ used evs ⟹ K ∉ range shrK"*
by *blast*

lemma *shrK_neq [simp]: "Key K ∉ used evs ⟹ shrK B ≠ K"*
by *blast*

```
declare shrK_neq [THEN not_sym, simp]
```

A.4 Function *knows*

```
lemma Spy_knows_bad_phones [intro!] :
  "Smartphone A ∈ badP ⟹ Key (shrK A) ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) add: imageI knows_Cons split: event.split)
done



lemma Spy_knows_stolen_phones [intro!] :
  "Smartphone A ∈ stolen ⟹ Key (shrK A) ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) add: imageI knows_Cons split: event.split)
using Stolen_in_badP by blast



lemma Crypt_Spy_analz_bad :
  "⟦ Crypt (shrK A) X ∈ analz (knows Spy evs); Smartphone A ∈ badP ⟧
    ⟹ X ∈ analz (knows Spy evs)"

  apply (erule analz.Decrypt)
  apply (simp add: Spy_knows_bad_phones)
done

lemma Crypt_Spy_analz_stolen :
  "⟦ Crypt (shrK A) X ∈ analz (knows Spy evs); Smartphone A ∈ stolen ⟧
    ⟹ X ∈ analz (knows Spy evs)"

  apply (erule analz.Decrypt)
  apply (simp add: Spy_knows_stolen_phones)
done
```

A.5 Nonce Lemmas

```
lemma Nonce_notin_initState [iff]: "Nonce N ∉ parts (initState (Friend i))"
```

```
by auto

lemma subset_Compl_range_shrK: "A ⊆ - (range shrK) ⟹ shrK x ∉ A"
by blast

lemma insert_Key_singleton: "insert (Key K) H = Key ‘ {K} ∪ H"
by blast

lemma insert_Key_image: "insert (Key K) (Key‘KK ∪ C) = Key‘(insert K KK) ∪ C"
by blast


lemma Nonce_supply :
  "Nonce (SOME N. Nonce N ∉ used evs) ∉ used evs"
apply (rule finite.emptyI [THEN Nonce_supply_ax, THEN exE])
apply (rule someI, blast)
done

lemmas analz_image_freshK_simps =
       simp_thms mem_simps — these two allow its use with only:
       disj_comms
       image_insert [THEN sym] image_Un [THEN sym] empty_subsetI insert_subset
       analz_insert_eq Un_upper2 [THEN analz_mono, THEN [2] rev_subsetD]
       insert_Key_singleton subset_Compl_range_shrK
       Key_not_used insert_Key_image Un_assoc [THEN sym]


lemma analz_image_freshK_lemma:
     "(Key K ∈ analz (Key‘nE ∪ H)) ⟶ (K ∈ nE | Key K ∈ analz H)  ⟹
         (Key K ∈ analz (Key‘nE ∪ H)) = (K ∈ nE | Key K ∈ analz H)"
by (blast intro: analz_mono [THEN [2] rev_subsetD])
```

A.6 Tactics for possibility theorems

```
ML
⟨
structure Smartphone =
struct
```

```
(*Omitting used_Says makes the tactic much faster: it leaves expressions
    such as  Nonce ?N ∉ used evs that match Nonce_supply*)
fun possibility_tac ctxt =
   (REPEAT
    (ALLGOALS (simp_tac (ctxt
      delsimps @{thms used_Cons_simps}
      setSolver safe_solver))
     THEN
     REPEAT_FIRST (eq_assume_tac ORELSE'
                   resolve_tac ctxt [refl, conjI, @{thm Nonce_supply}])))

(*For harder protocols (such as Recur) where we have to set up some
  nonces and keys initially*)
fun basic_possibility_tac ctxt =
    REPEAT
    (ALLGOALS (asm_simp_tac (ctxt setSolver safe_solver))
     THEN
     REPEAT_FIRST (resolve_tac ctxt [refl, conjI]))

val analz_image_freshK_ss =
  simpset_of
   (@{context} delsimps [image_insert, image_Un]
               delsimps [@{thm imp_disjL}]    (*reduces blow-up*)
               addsimps @{thms analz_image_freshK_simps})
end
⟩
```

lemma *invKey_shrK_iff [iff]:*
```
     "(Key (invKey K) ∈ X) = (Key K ∈ X)"
```
by *auto*

method_setup *analz_freshK = ⟨*
```
    Scan.succeed (fn ctxt =>
     (SIMPLE_METHOD
      (EVERY [REPEAT_FIRST (resolve_tac ctxt [allI, ballI, impI]),
          REPEAT_FIRST (resolve_tac ctxt @{thms analz_image_freshK_lemma}),
          ALLGOALS (asm_simp_tac (put_simpset Smartphone.analz_image_freshK_ss ctxt))])))⟩
    "for proving the Session Key Compromise theorem"
```

```
method_setup possibility = ‹
    Scan.succeed (fn ctxt =>
        SIMPLE_METHOD (Smartphone.possibility_tac ctxt))›
    "for proving possibility theorems"

method_setup basic_possibility = ‹
    Scan.succeed (fn ctxt =>
        SIMPLE_METHOD (Smartphone.basic_possibility_tac ctxt))›
    "for proving possibility theorems"

lemma knows_subset_knows_Cons: "knows A evs ⊆ knows A (e # evs)"
by (induct e) (auto simp: knows_Cons)


declare legalUse_def [iff] illegalUse_def [iff]

end
```

Appendix B

Smartphone Events Proof Script

```
theory EventSP imports "~~/src/HOL/Auth/Message" "~~/src/HOL/Library/Simps_Case_Conv"
begin

consts
  initState :: "agent => msg set"

datatype smartphone = Smartphone agent

datatype
  event = Says    agent agent msg
        | Notes   agent       msg
        | Gets    agent       msg
        | Scans   agent       smartphone msg
        | SGets  smartphone  msg
        | Shows smartphone  agent msg
        | AGets  agent       msg
        | Inputs  agent       smartphone msg

consts
  bad        :: "agent set"
  badP       :: "smartphone set"
  stolen     :: "smartphone set"
  secureP    :: "bool"

abbreviation
  insecureP :: bool where
  "insecureP == ¬secureP"

specification (bad)
  Spy_in_bad     [iff]: "Spy ∈ bad"
  Server_not_bad [iff]: "Server ∉ bad"
  apply (rule exI [of _ "{Spy}"], simp)
done

specification (badP)

  Spy_phone_not_badP     [iff]: "Smartphone Spy ∉ badP"
  Server_phone_not_badP [iff]: "Smartphone Server ∉ badP"
  apply blast
done

specification (stolen)
  Server_phone_not_stolen [iff]: "Smartphone Server ∉ stolen"
  Spy_phone_not_stolen    [iff]: "Smartphone Spy ∉ stolen"
  Stolen_in_badP [iff] : "stolen ⊆ badP"
  apply blast
done


primrec knows :: "agent ⇒ event list ⇒ msg set" where
  knows_Nil :  "knows A [] = initState A" |
  knows_Cons : "knows A (ev # evs) =
    (case ev of
```

— An agent knows what he sends to anyone. The Spy knows everything sent on a trace

```
      Says A' B X ⇒
        if (A = A' | A = Spy) then insert X (knows A evs)
        else (knows A evs)
```

— An agent knows what he notes. The Spy knows what compromised agents knows on a trace

```
      | Notes A' X ⇒
        if (A = A' | (A = Spy & A' ∈ bad)) then insert X (knows A evs)
        else knows A evs
```

— An agent, except the Spy, knows what she receives in a trace. Due to the Says event and reception invariant, the Spy knowledge does not need to be extended

```
| Gets A' X ⇒
  if (A = A' & A ≠ Spy) then insert X (knows A evs)
  else knows A evs
```

— An agent knows what she shows to her smartphone to scan. The Spy knows what a compromised agent shows to her smartphones to scan

```
| Scans A' P X ⇒
  if (A = A' | (A = Spy & A' ∈ bad)) then insert X (knows A evs)
  else knows A evs
```

— Due to reception invariant of Scans event, an agent does not enrich her knowledge set from what her smartphone receives

```
| SGets P X ⇒
  if secureP then knows A evs
```

— However, if devices can be compromised, the Spy knows what a compromised phone receives

```
  else
    if (A = Spy & P ∈ badP) then insert X (knows A evs)
    else knows A evs
```

— An agent, including the Spy, knows what her smartphone shows to her

```
| Shows P A' X ⇒
  if secureP then
    if (A = A') then insert X (knows A evs)
    else knows A evs
```

— When insecure devices hold, the Spy knows what compromised devices shows

```
  else
    if (A = Spy & P ∈ badP) then insert X (knows A evs)
    else knows A evs
```

— An agent knows what she receives from her smartphone. The Spy already knows from the previous event

```
| AGets A' X ⇒
  if (A = A' & A ≠ Spy) then insert X (knows A evs)
  else knows A evs
```

— An agent, and only her, knows what she manually inputs to her smartphone

```
| Inputs A' P X ⇒
  if (A = A') then insert X (knows A evs)
  else knows A evs
  )"


primrec used :: "event list ⇒ msg set" where
  used_Nil  : "used [] = (⋃ B. parts (initState B))" |
  used_Cons : "used (ev # evs) =
    (case ev of
        Says A B X ⇒ parts {X} ∪ (used evs)
      | Notes A X ⇒ parts {X} ∪ (used evs)
      | Gets A X ⇒ used evs
      | Scans A P X ⇒ parts {X} ∪ used evs
      | SGets P X ⇒ parts {X} ∪ used evs
      | Shows P A X ⇒ parts {X} ∪ used evs
      | AGets A X ⇒ used evs
      | Inputs A P X ⇒ parts {X} ∪ used evs
  )"
```

Describing how some the set used evs is enriched given our events

```
lemma Notes_imp_used [rule_format] :
  "Notes A X ∈ set evs ⟶ X ∈ used evs"
apply (induct_tac evs)
apply (auto split: event.split)
done

lemma Says_imp_used [rule_format] :
  "Says A B X ∈ set evs ⟶ X ∈ used evs"
apply (induct_tac evs)
apply (auto split: event.split)
done

lemma Scans_imp_used [rule_format] :
  "Scans A P X ∈ set evs ⟶ X ∈ used evs"
apply (induct_tac evs)
apply (auto split: event.split)
done

lemma Shows_imp_used [rule_format] :
  "Shows P A X ∈ set evs ⟶ X ∈ used evs"
apply (induct_tac evs)
apply (auto split: event.split)
```

```
done

lemma Inputs_imp_used :
  "Inputs A P X ∈ set evs ⟶ X ∈ used evs"
apply (induct_tac evs)
apply (auto split: event.split)
done
```

B.1 Function *knows*

```
lemmas parts_insert_knows_A = parts_insert [of _ "knows A evs"] for A evs

lemma knows_Spy_Says [simp] :
  "knows Spy (Says A B X # evs) = insert X (knows Spy evs)"
by simp

lemma knows_Spy_Gets [simp] :
  "knows Spy (Gets B X # evs) = knows Spy evs"
by simp

lemma knows_Spy_Notes [simp] :
  "knows Spy (Notes A X # evs) =
    (if A ∈ bad then insert X (knows Spy evs)
     else knows Spy evs)"
by simp

lemma knows_Spy_Scans [simp] :
  "knows Spy (Scans A P X # evs) =
    (if A ∈ bad then insert X (knows Spy evs)
     else knows Spy evs)"
by simp

lemma knows_Spy_SGets_secureP [simp] :
  "secureP ⟹ knows Spy (SGets P X # evs) = knows Spy evs"
by simp

lemma knows_Spy_SGets_insecureP [simp] :
  "insecureP ⟹ knows Spy (SGets P X # evs) =
    (if (P ∈ badP) then insert X (knows Spy evs)
    else knows Spy evs)"
by simp

lemma knows_Spy_Shows_secureP [simp] :
  "secureP ⟹ knows Spy (Shows P A X # evs) =
    (if A = Spy then insert X (knows Spy evs)
     else knows Spy evs)"
by simp

lemma knows_Spy_Shows_insecureP [simp] :
  "insecureP ⟹ knows Spy (Shows P A X # evs) =
    (if P ∈ badP then insert X (knows Spy evs)
     else knows Spy evs)"
by simp

lemma knows_Spy_AGets [simp] :
  "knows Spy (AGets A X # evs) = knows Spy evs"
by simp

lemma knows_Spy_Inputs [simp] :
  "knows Spy (Inputs A P X # evs) =
    (if A = Spy then insert X (knows Spy evs)
     else knows Spy evs)"
by simp


lemma knows_Spy_subset_knows_Spy_Says :
  "knows Spy evs ⊆ knows Spy (Says A B X # evs)"
by (simp add: subset_insertI)

lemma knows_Spy_subset_knows_Spy_Notes :
  "knows Spy evs ⊆ knows Spy (Notes A X # evs)"
by force

lemma knows_Spy_subset_knows_Spy_Gets :
  "knows Spy evs ⊆ knows Spy (Gets A X # evs)"
by (simp add: subset_insertI)

lemma knows_Spy_subset_knows_Spy_Scans :
  "knows Spy evs ⊆ knows Spy (Scans A P X # evs)"
```

```
by auto

lemma knows_Spy_subset_knows_Spy_SGets :
  "knows Spy evs ⊆ knows Spy (SGets P X # evs)"
by (simp add: subset_insertI)

lemma knows_Spy_subset_knows_Spy_Shows :
  "knows Spy evs ⊆ knows Spy (Shows P A X # evs)"
by (simp add: subset_insertI)

lemma knows_Spy_subset_knows_Spy_AGets :
  "knows Spy evs ⊆ knows Spy (AGets A X # evs)"
by (simp add: subset_insertI)

lemma knows_Spy_subset_knows_Spy_Inputs :
  "knows Spy evs ⊆ knows Spy (Inputs A P X # evs)"
by (simp add: subset_insertI)


lemma Says_imp_knows_Spy [rule_format (no_asm)] :
  "Says A B X ∈ set evs ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all split: event.split)
done

lemma Notes_imp_knows_Spy [rule_format (no_asm)] :
  "Notes A X ∈ set evs ⟶ A ∈ bad ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done
```

— Nothing can be stated about Gets event

```
lemma Scans_imp_knows_Spy [rule_format (no_asm)] :
  "Scans A P X ∈ set evs ⟶ A ∈ bad ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
```

```
done

— Nothing can be stated on a SGets when phones are secured event
lemma SGets_imp_knows_Spy_insecureP [rule_format (no_asm)] :
  "SGets P X ∈ set evs ⟶ (insecureP ∧ P ∈ badP) ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done

lemma Shows_imp_knows_Spy_secureM [rule_format (no_asm)] :
  "Shows P Spy X ∈ set evs ⟶ secureP ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done

lemma Shows_imp_know_Spy_insecureM [rule_format (no_asm)] :
  "Shows P A X ∈ set evs ⟶ (insecureP ∧ P ∈ badP) ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done

— Nothing can be stated here on a AGets event
lemma Inputs_imp_knows_Spy [rule_format] :
  "Inputs Spy P X ∈ set evs ⟶ X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done

lemmas knows_Spy_partsEs =
     Says_imp_knows_Spy [THEN parts.Inj, elim_format]
     parts.Body [elim_format]


lemma knows_Says: "knows A (Says A B X # evs) = insert X (knows A evs)"
by simp
```

```
lemma knows_Notes: "knows A (Notes A X # evs) = insert X (knows A evs)"
by simp

lemma knows_Gets:
  "A ≠ Spy ⟶ knows A (Gets A X # evs) = insert X (knows A evs)"
by simp

lemma knows_Scans: "knows A (Scans A P X # evs) = insert X (knows A evs)"
by simp

lemma knows_SGets:
  "A ≠ Spy ⟶ knows A (SGets P X # evs) = knows A evs"
by simp

lemma knows_Shows_secureP:
  "secureP ⟶ knows A (Shows P A X # evs) = insert X (knows A evs)"
by simp

lemma knows_Shows_insecureP:
  "(insecureP ∧ A ≠ Spy) ⟶ knows A (Shows P A X # evs) = knows A evs"
by simp

lemma knows_Inputs:
  "knows A (Inputs A P X # evs) = insert X (knows A evs)"
by simp

lemma knows_subset_knows_Says: "knows A evs ⊆ knows A (Says A' B X # evs)"
by (simp add: subset_insertI)

lemma knows_subset_knows_Notes: "knows A evs ⊆ knows A (Notes A' X # evs)"
by (simp add: subset_insertI)

lemma knows_subset_knows_Gets: "knows A evs ⊆ knows A (Gets A' X # evs)"
by (simp add: subset_insertI)

lemma knows_subset_knows_Scans: "knows A evs ⊆ knows A (Scans A' P X # evs)"
by (simp add: subset_insertI)

lemma knows_subset_knows_SGets: "knows A evs ⊆ knows A (SGets P X # evs)"
by (simp add: subset_insertI)
```

```
lemma knows_subset_knows_Shows: "knows A evs ⊆ knows A (Shows P A' X # evs)"
by (simp add: subset_insertI)

lemma knows_subset_knows_AGets: "knows A evs ⊆ knows A (AGets A' X # evs)"
by (simp add: subset_insertI)

lemma knows_subset_knows_Inputs: "knows A evs ⊆ knows A (Inputs A' P X # evs)"
by (simp add: subset_insertI)
```

— Agents know what they say

```
lemma Says_imp_knows [rule_format] :
  "Says A B X ∈ set evs ⟶ X ∈ knows A evs"
  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
  apply (auto)
done
```

— Agents know what they note

```
lemma Notes_imp_knows [rule_format] :
  "Notes A X ∈ set evs ⟶ X ∈ knows A evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
  apply (auto)
done
```

— Agents know what they receive

```
lemma Gets_imp_knows [rule_format] :
  "A ≠ Spy ⟶ Gets A X ∈ set evs ⟶ X ∈ knows A evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done
```

— Agents know what their smartphone scans

```
lemma Scans_imp_knows [rule_format] :
  "Scans A (Smartphone A) X ∈ set evs ⟶ X ∈ knows A evs"
  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
  apply (auto)
done
```

— Agents know what they input to their smartphone

```
lemma Inputs_imp_knows [rule_format] :
  "Inputs A (Smartphone A) X ∈ set evs ⟶ X ∈ knows A evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
  apply (auto)
done
```

— Agents do not know what they smartphones reads...

— Agents know what their smartphones shows to them, if the device are secured

```
lemma Shows_imp_knows_secureP [rule_format] :
  "secureP ⟶ Shows (Smartphone A) A X ∈ set evs ⟶ X ∈ knows A evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done

lemma Shows_imp_knows_insecureP [rule_format] :
  "(insecureP ∧ (Smartphone A) ∈ badP) ⟶ Shows (Smartphone A) A X ∈ set evs ⟶
X ∈ knows Spy evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done
```

— Agents know what they receive from a smartphone

```
lemma AGets_imp_knows [rule_format] :
  "A ≠ Spy ⟶ AGets A X ∈ set evs ⟶ X ∈ knows A evs"

  apply (induct_tac "evs")
  apply (simp_all (no_asm_simp) split: event.split)
done
```

```
lemma parts_knows_Spy_subset_used :
  "parts (knows Spy evs) ⊆ used evs"
apply (induct_tac "evs", force)
apply (simp add: parts_insert_knows_A split: event.split)
apply (auto)
done

lemmas usedI = parts_knows_Spy_subset_used [THEN subsetD, intro]

lemma initState_into_used :
  "X ∈ parts (initState B) ==> X ∈ used evs"
apply (induct_tac "evs")
apply (simp_all add: parts_insert_knows_A split: event.split, blast)
done

simps_of_case used_Cons_simps [simp]: used_Cons
```

B.2 Function *used*

```
lemma Says_parts_used [rule_format (no_asm)] :
  "Says A B X ∈ set evs ⟶ (parts {X}) ⊆ used evs "
apply (induct_tac "evs")
apply (simp_all (no_asm_simp) split: event.split)
apply (auto)
done

lemma Notes_parts_used [rule_format (no_asm)] :
  "Notes A X ∈ set evs ⟶ (parts {X}) ⊆ used evs"
apply (induct_tac "evs")
apply (simp_all (no_asm_simp) split: event.split)
apply (auto)
done

lemma Scans_parts_used [rule_format (no_asm)] :
  "Scans A P X ∈ set evs ⟶ (parts {X}) ⊆ used evs"
apply (induct_tac "evs")
apply (simp_all (no_asm_simp) split: event.split)
apply (auto)
done
```

```
lemma Shows_parts_used [rule_format (no_asm)] :
  "Shows P A X ∈ set evs ⟶ (parts {X}) ⊆ used evs"
apply (induct_tac "evs")
apply (simp_all (no_asm_simp) split: event.split)
apply (auto)
done

lemma Inputs_parts_used [rule_format (no_asm)] :
  "Inputs A P X ∈ set evs ⟶ (parts {X}) ⊆ used evs"
apply (induct_tac "evs")
apply (simp_all (no_asm_simp) split: event.split)
apply (auto)
done


declare knows_Cons [simp del]
        used_Nil [simp del]
        used_Cons [simp del]

lemma knows_subset_knows_Cons :
  "knows A evs ⊆ knows A (e # evs)"
by (induct e, auto simp: knows_Cons)

lemma initState_subset_knows :
  "initState A ⊆ knows A evs"
apply (induct_tac evs, simp)
apply (blast intro: knows_subset_knows_Cons [THEN subsetD])
done

lemma keysFor_parts_insert:
     "⟦ K ∈ keysFor (parts (insert X G));  X ∈ synth (analz H) ⟧
      ⟹ K ∈ keysFor (parts (G ∪ H)) ∨ Key (invKey K) ∈ parts H"
by (force
    dest!: parts_insert_subset_Un [THEN keysFor_mono, THEN [2] rev_subsetD]
           analz_subset_parts [THEN keysFor_mono, THEN [2] rev_subsetD]
    intro: analz_subset_parts [THEN subsetD] parts_mono [THEN [2] rev_subsetD])

end
```

Appendix C

Secure DAP Proof Script

```
theory SDAP_Transaction imports "./Smartphone"

begin

axiomatization where
  sdaptrans_assume_insecure_devices [iff]: "evs ∈ sdaptrans ⟹ secureP"
```

C.1 Protocol Model

```
inductive_set sdaptrans :: "event list set" where
  Nil: "[] ∈ sdaptrans"

  | DT1: "⟦ evs1 ∈ sdaptrans; A ≠ Server ⟧
    ⟹ Says A Server ⦃ Agent A, Number T ⦄ # evs1 ∈ sdaptrans"

  | DT2: "⟦ evs2 ∈ sdaptrans; Nonce r ∉ used evs2;
           Gets Server ⦃ Agent A, Number T ⦄ ∈ set evs2 ⟧
    ⟹ Says Server A ⦃
          ⦃Agent A, Number T⦄,
          Crypt (shrK A) (Nonce r),
          Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
        ⦄ # evs2 ∈ sdaptrans"

  | DT3: "⟦ evs3 ∈ sdaptrans; legalUse (Smartphone A);
           Says A Server ⦃ Agent A, Number T ⦄ ∈ set evs3;
           Gets A ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs3 ⟧
  ⟹ Scans A (Smartphone A) ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ # evs3 ∈ sdaptrans"

 | DT4: "⟦ evs4 ∈ sdaptrans; legalUse(Smartphone A);
          Scans A (Smartphone A) ⦃
            ⦃Agent A, Number T⦄,
            Crypt (shrK A) (Nonce r),
            Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
          ⦄ ∈ set evs4 ⟧
  ⟹ Shows (Smartphone A) A ⦃ Agent A, Number T ⦄ # evs4 ∈ sdaptrans"

 | DT4_Fake: "⟦ evs4f ∈ sdaptrans; illegalUse(Smartphone A);
               Scans Spy (Smartphone A) ⦃
                ⦃Agent A, Number T⦄,
                Crypt (shrK A) (Nonce r),
                Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r)
⦄
               ⦄ ∈ set evs4f⟧
   ⟹ Shows (Smartphone A) Spy ⦃ Agent A, Number T ⦄ # evs4f ∈ sdaptrans"


 | DT5: "⟦ evs5 ∈ sdaptrans; legalUse(Smartphone A);
          Says A Server ⦃ Agent A, Number T ⦄ ∈ set evs5;
          Gets A ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs5;
          Scans A (Smartphone A) ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs5;
          Shows (Smartphone A) A ⦃ Agent A, Number T ⦄ ∈ set evs5 ⟧
  ⟹ Inputs A (Smartphone A) ⦃ Agent A, Number T ⦄ # evs5 ∈ sdaptrans"

 | DT6: "⟦ evs6 ∈ sdaptrans; legalUse(Smartphone A);
          Scans A (Smartphone A) ⦃
            ⦃Agent A, Number T⦄,
            Crypt (shrK A) (Nonce r),
            Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
          ⦄ ∈ set evs6;
          Shows (Smartphone A) A ⦃ Agent A, Number T ⦄ ∈ set evs6;
          Inputs A (Smartphone A) ⦃ Agent A, Number T ⦄ ∈ set evs6 ⟧
 ⟹ Shows (Smartphone A) A (Nonce r) # evs6 ∈ sdaptrans"

  | DT6_Fake: "⟦ evs6f ∈ sdaptrans; illegalUse(Smartphone A);
                Scans Spy (Smartphone A) ⦃
                  ⦃Agent A, Number T⦄,
                     Crypt (shrK A) (Nonce r),
                     Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce
r) ⦄
                   ⦄ ∈ set evs6f;
                   Shows (Smartphone A) Spy ⦃ Agent A, Number T ⦄ ∈ set evs6f;
                   Inputs Spy (Smartphone A) ⦃ Agent A, Number T ⦄ ∈ set evs6f
                ⟧
    ⟹ Shows (Smartphone A) Spy (Nonce r) # evs6f ∈ sdaptrans"

  | DT7: "⟦ evs7 ∈ sdaptrans;
            Says A Server ⦃ Agent A, Number T ⦄ ∈ set evs7;
            Gets A ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs7;
            Scans A (Smartphone A) ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs7;
            Shows (Smartphone A) A ⦃ Agent A, Number T ⦄ ∈ set evs7;
            Inputs A (Smartphone A) ⦃ Agent A, Number T ⦄ ∈ set evs7;
            Shows (Smartphone A) A (Nonce r) ∈ set evs7 ⟧
    ⟹ Says A Server (Nonce r) # evs7 ∈ sdaptrans"

  | DT8: "⟦ evs8 ∈ sdaptrans; A ≠ Server;
            Gets Server ⦃ Agent A, Number T ⦄ ∈ set evs8;
            Says Server A ⦃
              ⦃Agent A, Number T⦄,
              Crypt (shrK A) (Nonce r),
              Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
            ⦄ ∈ set evs8;
            Gets Server (Nonce r) ∈ set evs8 ⟧
    ⟹ Says Server A ⦃ Agent A, Number T ⦄  # evs8 ∈ sdaptrans"


  | Fake: "⟦ evsF ∈ sdaptrans; X ∈ synth(analz(knows Spy evsF));
             illegalUse(Smartphone A) ⟧
    ⟹ Says Spy B X #
        Scans Spy (Smartphone A) X # evsF ∈ sdaptrans"


  | Rcpt: "⟦ evsR ∈ sdaptrans; Says A B X ∈ set evsR ⟧ ⟹ Gets B X # evsR ∈ sdaptrans"


declare Fake_parts_insert_in_Un  [dest]
declare analz_into_parts [dest]
```

C.2 Message Reception Lemmas

```
lemma Gets_imp_Says :
  "⟦ Gets B X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ ∃ A. Says A B X ∈ set evs"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done

lemma Gets_imp_knows_Spy :
  "⟦ Gets B X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ X ∈ knows Spy evs"
by (blast dest!: Gets_imp_Says Says_imp_knows_Spy)

lemma Gets_imp_knows_Spy_analz :
  "⟦ Gets B X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ X ∈ analz (knows Spy evs)"
by (blast dest!: Gets_imp_Says Says_imp_knows_Spy)

lemma Gets_imp_knows_Spy_analz_Snd :
 "⟦ Gets B ⦃X, Y⦄ ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ Y ∈ analz (knows Spy evs)"

  apply (blast dest!: Gets_imp_Says Says_imp_knows_Spy analz.Inj analz.Snd)
done

lemmas Gets_imp_knows_Spy_parts [dest] = Gets_imp_knows_Spy_analz [THEN analz_into_parts]
lemmas Gets_imp_knows_Spy_parts_Snd [dest] = Gets_imp_knows_Spy_analz_Snd [THEN
analz_into_parts]

lemma SGets_imp_Scans :
  "⟦ SGets P X ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ ∃ A. (Scans A P X ∈ set evs) ∨ (Inputs A P X ∈ set evs)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done

lemma SGets_imp_knows_Spy :
  "⟦ SGets (Smartphone B) X ∈ set evs; (Smartphone B) ∈ badP; evs ∈ sdaptrans ⟧
    ⟹ X ∈ knows Spy evs"

  apply (erule rev_mp, erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done

lemma SGets_imp_knows_Spy_analz :
  "⟦ SGets (Smartphone B) X ∈ set evs; (Smartphone B) ∈ badP; evs ∈ sdaptrans ⟧
    ⟹ X ∈ analz (knows Spy evs)"
by (blast dest!: SGets_imp_knows_Spy)

lemmas SGets_imp_knows_Spy_parts [dest] = SGets_imp_knows_Spy_analz [THEN analz_into_parts

lemma AGets_imp_Shows :
  "⟦ AGets A X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ ∃ P. Shows P A X ∈ set evs"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done
```

C.3 Smartphone Device Lemmas

```
lemma Scans_imp_knows_Spy_insecureP_sdaptrans :
  "⟦ Scans Spy P X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ X ∈ knows Spy evs"

  apply (simp (no_asm_simp) add: Scans_imp_knows_Spy)
done

lemma knows_Spy_Scans_insecureP_sdaptrans_Spy :
  "evs ∈ sdaptrans ⟹ knows Spy (Scans Spy P X # evs) = insert X (knows Spy evs)"
by simp

lemma knows_Spy_Scans_insecureP_sdaptrans :
  "⟦ A ≠ Spy; A ∈ bad; evs ∈ sdaptrans ⟧
    ⟹ knows Spy (Scans A P X # evs) = insert X (knows Spy evs)"
by simp

lemma knows_Spy_Shows_secureM_sdaptrans_Spy :
  "evs ∈ sdaptrans ⟹ knows Spy (Shows P Spy X # evs) = insert X (knows Spy evs)"
by simp
```

```
lemma knows_Spy_Shows_secureP_sdaptrans :
  "⟦ P ∈ stolen; evs ∈ sdaptrans ⟧
    ⟹ knows Spy (Shows P Spy X # evs) = insert X (knows Spy evs)"
by simp

lemma knows_Spy_Inputs_sdaptrans_Spy :
  "evs ∈ sdaptrans ⟹ knows Spy (Inputs Spy P X # evs) = insert X (knows Spy evs)"
by simp

lemma knows_Spy_Inputs_sdaptrans :
  "⟦ A ≠ Spy; evs ∈ sdaptrans ⟧
    ⟹ knows Spy (Inputs A P X # evs) = knows Spy evs"
by simp
```

C.4 Reliability Lemmas

For reasoning about encrypted portion of messages

```
lemma DT3_analz_knows_Spy_fst :
 "⟦ Gets A ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ r' ∈ analz (knows Spy evs)"
by (blast dest!: Gets_imp_Says Gets_imp_knows_Spy_analz_Snd)

lemma DT3_analz_knows_Spy_snd :
 "⟦ Gets A ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ h_s ∈ analz (knows Spy evs)"
by (blast dest!: Gets_imp_Says Gets_imp_knows_Spy_analz_Snd)

lemmas DT3_parts_knows_Spy_fst = DT3_analz_knows_Spy_fst [THEN analz_into_parts]
lemmas DT3_parts_knows_Spy_snd = DT3_analz_knows_Spy_snd [THEN analz_into_parts]
```

C.4.1 Server Guarantees

```
lemma Says_Server_DT1_not_evs :
  "evs ∈ sdaptrans ⟹ Says Server Server ⦃ Agent Server, Number T ⦄ ∉ set evs"

  apply (erule sdaptrans.induct)
  apply (simp_all)
done

lemma Server_cannot_initiate :
  "⟦ Says A Server ⦃ Agent A, Number T ⦄ ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ A ≠ Server"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all)
done
```

The Server smartphone is not usable

```
lemma Scans_Agent_Server_not_evs [rule_format, simp] :
  "evs ∈ sdaptrans ⟹ Scans Server (Smartphone A) X ∉ set evs"

  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast dest:Server_cannot_initiate)
done

lemma Scans_Server_Agent_not_evs [rule_format] :
  "evs ∈ sdaptrans ⟹ Scans A (Smartphone Server) X ∉ set evs"

  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast dest:Server_cannot_initiate)+
done

lemma Shows_Agent_Server_not_evs [rule_format] :
  "evs ∈ sdaptrans ⟹ Shows (Smartphone A) Server X ∉ set evs"

  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast dest:Scans_Server_Agent_not_evs)+
done

lemma Shows_Server_Agent_not_evs [rule_format]:
  "evs ∈ sdaptrans ⟹ Shows (Smartphone Server) A X ∉ set evs"

  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast dest:Scans_Server_Agent_not_evs)+
done
```

Server expected message form to the sender

```
lemma Says_Server_DT2 :
  "⟦ Says Server A ⦃
       ⦃Agent A, Number T⦄,
       Crypt (shrK A) (Nonce r),
       Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
     ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ Gets Server ⦃Agent A, Number T⦄ ∈ set evs"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done
```

The Server only authorizes a transaction if it received a nonce that matches the produced TAN

```
lemma Says_Server_DT8 :
  "⟦ Server ≠ A;
    Says Server A ⦃ Agent A, Number T ⦄ ∈ set evs;
    evs ∈ sdaptrans ⟧
    ⟹ ∃ r.
      Gets Server ⦃Agent A, Number T⦄ ∈ set evs ∧
      Says Server A ⦃
        ⦃Agent A, Number T⦄,
        Crypt (shrK A) (Nonce r),
        Crypt (shrK A) ⦃⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r)⦄
      ⦄ ∈ set evs ∧
      Gets Server (Nonce r) ∈ set evs"

  apply (erule rev_mp)
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done
```

C.4.2 Smartphone Usability Guarantees

```
lemma Scans_Smartphone_legalUse :
  "⟦ Scans A (Smartphone A) X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ legalUse(Smartphone
A)"
apply (erule rev_mp, erule sdaptrans.induct)
apply (auto)
done
```

```
lemma Shows_Smartphone_legalUse :
  "⟦ Shows (Smartphone A) A X ∈ set evs; evs ∈ sdaptrans ⟧ ⟹ legalUse(Smartphone
A)"
apply (erule rev_mp, erule sdaptrans.induct)
apply (auto)
done



lemma Scans_Smartphone :
  "⟦ Scans A P X ∈ set evs; A ≠ Spy; evs ∈ sdaptrans ⟧
    ⟹ (P = (Smartphone A) ∧ legalUse(P)) ∨ (P = (Smartphone Spy) ∧ illegalUse(P))"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done

lemma Shows_Smartphone :
  "⟦ Shows P A X ∈ set evs; A ≠ Spy; evs ∈ sdaptrans ⟧
    ⟹ (P = (Smartphone A) ∧ legalUse(P)) ∨ (P = (Smartphone Spy))"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all)
done

lemma Scans_Shows_Smartphone :
  "⟦ Scans A P X ∈ set evs ∨ Shows P A X ∈ set evs; A ≠ Spy; evs ∈ sdaptrans ⟧
     ⟹ (P = (Smartphone A) ∧ legalUse(Smartphone A))"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast)+
done



lemma Scans_Smartphone_Spy :
  "⟦ Scans Spy P X ∈ set evs ∨ Shows P Spy X ∈ set evs; evs ∈ sdaptrans ⟧
      ⟹ (P = (Smartphone Spy)) ∧ (legalUse(Smartphone Spy)) ∨
          (∃ A. P = (Smartphone A) ∧ illegalUse(Smartphone A))"
  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done
```

C.4.3 Protocol termination

```
lemma Protocol_terminates :
  "∃ A T. ∃ evs ∈ sdaptrans. A ≠ Server ∧ Says Server A ⦃ Agent A, Number T ⦄
∈ set evs"

  apply (intro exI bexI)
  apply (rule_tac [2] sdaptrans.Nil [THEN sdaptrans.DT1, THEN sdaptrans.Rcpt,
        THEN sdaptrans.DT2, THEN sdaptrans.Rcpt,
        THEN sdaptrans.DT3, THEN sdaptrans.DT4,
        THEN sdaptrans.DT5, THEN sdaptrans.DT6,
        THEN sdaptrans.DT7, THEN sdaptrans.Rcpt,
        THEN sdaptrans.DT8])
  apply (possibility, auto)
done
```

C.4.4 Scans Event Guarantees

```
lemma Scans_A_honest_Smartphone_3 :
  "⟦ Scans A P ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs; A ≠ Spy; evs ∈ sdaptrans
⟧
    ⟹ (legalUse(P)) ∧ P = (Smartphone A) ∧
        Says A Server ⦃Agent A, Number T⦄ ∈ set evs ∧
        Gets A ⦃ ⦃ Agent A, Number T ⦄, r', h_s ⦄ ∈ set evs"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all)
done
```

This is an important guarantee: the protocol legally continues if the agent confirms the outputed message, which contains the transaction

```
lemma Inputs_A_honest_Smartphone_5 :
  "⟦ Inputs A P ⦃Agent A, Number T⦄ ∈ set evs; A ≠ Spy; evs ∈ sdaptrans ⟧
    ⟹ (legalUse(P)) ∧ P = (Smartphone A) ∧
        (∃ r' h_s.
          Says A Server ⦃Agent A, Number T⦄ ∈ set evs ∧
          Gets A ⦃ ⦃Agent A, Number T⦄, r', hs ⦄ ∈ set evs ∧
          Scans A (Smartphone A) ⦃ ⦃Agent A, Number T⦄, r', hs ⦄ ∈ set evs ∧
          Shows (Smartphone A) A ⦃Agent A, Number T⦄ ∈ set evs)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done

lemma Inputs_A_Smartphone_5 :
  "⟦ Inputs A P ⦃Agent A, Number T⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ (legalUse(P)) ∧ P = (Smartphone A) ∧
        (∃ r' hs.
          Says A Server ⦃Agent A, Number T⦄ ∈ set evs ∧
          Gets A ⦃ ⦃Agent A, Number T⦄, r', hs ⦄ ∈ set evs ∧
          Scans A (Smartphone A) ⦃ ⦃Agent A, Number T⦄, r', hs ⦄ ∈ set evs ∧
          Shows (Smartphone A) A ⦃Agent A, Number T⦄ ∈ set evs)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done
```

C.4.5 Shows Event Guarantees

```
lemma Shows_which_Smartphone_4 :
  "⟦ Shows (Smartphone A) A ⦃Agent A, Number T⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ (∃ r. Scans A (Smartphone A) ⦃
          ⦃Agent A, Number T⦄,
          Crypt (shrK A) (Nonce r),
          Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r)⦄
        ⦄ ∈ set evs)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done

lemma Shows_honest_A_Smartphone_4 :
  "⟦ Shows P A ⦃Agent A, Number T⦄ ∈ set evs; A ≠ Spy; evs ∈ sdaptrans ⟧
```

```
    ⟹ (legalUse(P)) ∧ P = (Smartphone A) ∧
        (∃ r. Scans A (Smartphone A) ⦃
          ⦃Agent A, Number T⦄,
          Crypt (shrK A) (Nonce r),
          Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
        ⦄ ∈ set evs)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all)
  apply (force+)
done



lemma Shows_A_Smartphone_4 :
  "⟦ Shows P A ⦃Agent A, Number T⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ (legalUse(P)) ∧ P = (Smartphone A) ∧
        (∃ r. Scans A (Smartphone A) ⦃
          ⦃Agent A, Number T⦄,
          Crypt (shrK A) (Nonce r),
          Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
        ⦄ ∈ set evs)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all, force+)
done



lemma Shows_uncompromised_A_Smartphone_6 :
  "⟦ Shows P A (Nonce r) ∈ set evs; legalUse(P); P ∉ badP; evs ∈ sdaptrans ⟧
    ⟹ (P = (Smartphone A)) ∧ (∃ T. Inputs A P ⦃ Agent A, Number T ⦄ ∈ set evs)"

  apply (erule rev_mp)
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast+)
done

lemma Shows_honest_A_Smartphone_6 :
  "⟦ Shows P A (Nonce r) ∈ set evs; A ≠ Spy; evs ∈ sdaptrans ⟧
```

```
    ⟹ (legalUse(P)) ∧ P = (Smartphone A) ∧
        (∃ T. Inputs A (Smartphone A) ⦃Agent A, Number T ⦄ ∈ set evs)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all, force+)
done

lemma Shows_Someone_Smartphone_6 :
  "⟦ Shows (Smartphone A) B (Nonce r) ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ (∃ T. Inputs B (Smartphone A) ⦃ Agent A, Number T ⦄ ∈ set evs)"
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (blast+)
done

lemma Shows_which_Smartphone_6 :
  "⟦ Shows (Smartphone A) A (Nonce r) ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ (∃ T.
          Scans A (Smartphone A) ⦃
            ⦃Agent A, Number T⦄,
            Crypt (shrK A) (Nonce r),
            Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
          ⦄ ∈ set evs ∧
          Inputs A (Smartphone A) ⦃Agent A, Number T ⦄ ∈ set evs)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done



lemma Scans_A_Smartphone_form_3 :
  "⟦ Scans A (Smartphone A) ⦃ Transaction, r', h_s ⦄ ∈ set evs;
     ∀ p q. Transaction = ⦃p, q⦄; evs ∈ sdaptrans ⟧
    ⟹ (∃ T r. Transaction = ⦃ Agent A, Number T ⦄)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done
```

```
lemma Shows_A_Smartphone_form_4 :
  "⟦ Shows (Smartphone A) A Transaction ∈ set evs; evs ∈ sdaptrans;
    ∀ p q. Transaction = ⦃p, q⦄ ⟧
    ⟹ ∃ T. Transaction = ⦃Agent A, Number T⦄"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done

lemma Shows_A_Smartphone_form_6 :
  "⟦ Shows (Smartphone A) A TAN ∈ set evs; evs ∈ sdaptrans;
    ∀ p q. TAN ≠ ⦃p, q⦄ ⟧
    ⟹ ∃ r. TAN = (Nonce r)"

  apply (erule rev_mp, erule sdaptrans.induct)
  apply (auto)
done
```

C.5 Spy Counterguarantees

```
lemma Spy_knows_Transaction :
  "⟦ Says A Server ⦃Agent A, Number T⦄ ∈ set evs; evs ∈ sdaptrans ⟧
     ⟹ Number T ∈ analz (knows Spy evs)"
by (blast dest!: Says_imp_knows_Spy [THEN analz.Inj, THEN analz.Snd])

lemma Spy_knows_TAN :
  "⟦ Says A Server (Nonce r) ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ Nonce r ∈ knows Spy evs"
by (blast dest!: Says_imp_knows_Spy)

lemma TAN_Says_Server_analz_knows_Spy :
  "⟦ Says Server A ⦃
       ⦃Agent A, Number T⦄,
       Crypt (shrK A) (Nonce r),
       Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
     ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
  ⟹ Crypt (shrK A) (Nonce r) ∈ analz (knows Spy evs)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all add: analz_insertI)
done

lemma Hash_Says_Server_analz_knows_Spy :
  "⟦ Says Server A ⦃
       ⦃Agent A, Number T⦄,
       Crypt (shrK A) (Nonce r),
       Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
     ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
  ⟹ Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄ ∈ analz (knows
Spy evs)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all add: analz_insertI)
done

lemma TAN_Scans_analz_knows_Spy :
  "⟦ Scans A (Smartphone A) ⦃
       ⦃Agent A, Number T⦄,
       Crypt (shrK A) (Nonce r),
       Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
     ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
  ⟹ Crypt (shrK A) (Nonce r) ∈ analz (knows Spy evs)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all add: analz_insertI)
  apply (blast dest!: DT3_analz_knows_Spy_fst)+
done

lemma Hash_Scans_analz_knows_Spy :
  "⟦ Scans A (Smartphone A) ⦃
       ⦃Agent A, Number T⦄,
       Crypt (shrK A) (Nonce r),
       Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
     ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
  ⟹ Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄ ∈ analz (knows
Spy evs)"
```

```
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all add: analz_insertI)
  apply (blast dest!: DT3_analz_knows_Spy_snd)+
done
```

C.6 Regularity Lemmas

```
lemma Spy_parts_keys [simp] :
  "evs ∈ sdaptrans ⟹ (Key (shrK A) ∈ parts (knows Spy evs)) = (Smartphone A ∈
badP)"

  apply (erule sdaptrans.induct)
  apply (frule_tac [4] DT3_parts_knows_Spy_fst)
  apply (frule_tac [5] DT3_parts_knows_Spy_snd)
  apply (simp_all)
  apply (auto intro!:parts_insertI)
done

lemma Spy_analz_shrK [simp] :
  "evs ∈ sdaptrans ⟹ (Key (shrK A) ∈ analz (knows Spy evs)) = (Smartphone A ∈
badP)"
by (auto dest!: Spy_knows_bad_phones)
```

C.7 Propeties Lemmas

```
lemma Server_transaction_unique :
  "⟦ Says Server A ⦃
      ⦃Agent A, Number T⦄,
      Crypt (shrK A) (Nonce r),
      Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
    ⦄ ∈ set evs ;
    Says Server A' ⦃
      ⦃Agent A', Number T'⦄,
      Crypt (shrK A') (Nonce r),
      Crypt (shrK A') ⦃ ⦃Agent A', Number T'⦄, Crypt (shrK A') (Nonce r) ⦄
     ⦄ ∈ set evs;
    evs ∈ sdaptrans⟧
```

```
   ⟹ A = A' ∧ T = T'"

  apply (erule rev_mp)
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all)
  apply (fastforce dest: Says_parts_used)
done

lemma Server_Transaction_not_unique :
  "⟦ Says Server A ⦃
      ⦃Agent A, Number T⦄,
      Crypt (shrK A) (Nonce r),
      Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
    ⦄ ∈ set evs ;
    Says Server A ⦃
      ⦃Agent A, Number T⦄,
      Crypt (shrK A) (Nonce r'),
      Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r') ⦄
     ⦄ ∈ set evs;
    evs ∈ sdaptrans⟧
   ⟹ ∃ r r'. (r = r')"

  apply (erule rev_mp)
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all)
done


lemma Ciphers_authentic :
  "⟦ Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ parts (knows Spy evs);
     (Smartphone A) ∉ badP; evs ∈ sdaptrans ⟧
     ⟹ r' = Crypt (shrK A) (Nonce r) ∧
         h_s = Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r)⦄ ∧
         Says Server A ⦃
            ⦃Agent A, Number T⦄,
            Crypt (shrK A) (Nonce r),
            Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
         ⦄ ∈ set evs"
  apply (erule rev_mp, erule sdaptrans.induct)
  apply (simp_all (no_asm_simp))
  apply (auto)
done

lemma TAN_A_identity_by_Server :
  "⟦ Says Server A' ⦃
      ⦃Agent A, Number T⦄,
      Crypt (shrK A) (Nonce r),
      Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄
     ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
     ⟹ A' = A"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (simp_all)
done
```

step2_integrity also is a reliability theorem

```
lemma Says_Server_message_form_DT2 :
  "⟦ Says Server A ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs; evs ∈ sdaptrans ⟧
    ⟹ (∃ r.
        r' = Crypt (shrK A) (Nonce r) ∧
        h_s = Crypt (shrK A) ⦃ ⦃Agent A, Number T⦄, Crypt (shrK A) (Nonce r) ⦄)"

  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done
```

Rule DT3 format is the form of the message which the agent redirect to its smartphone. It cannot be proven, since there is no way to the agent know what he is forwarding

```
lemma Scans_Smartphone_A_DT3_message_form_unprovable :
  "⟦ Scans A (Smartphone A) ⦃ ⦃Agent A, Number T⦄, r', h_s ⦄ ∈ set evs; evs ∈ sdaptrans
⟧
  ⟹ (∃ r'. (∃ r. r' ≠ Crypt (shrK A) (Nonce r)))"
  apply (erule rev_mp)
  apply (erule sdaptrans.induct)
  apply (auto)
done
```

```
end
```