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

Verification of Dynamic Authorization Protocol

Felipe Rodopoulos de Oliveira

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof.a Dr.a Cláudia Nalon

Coorientador
Prof. Dr. João da Costa Gondim

Brasília
2017

Ficha Catalográfica de Teses e Dissertações

Está página existe apenas para indicar onde a ficha catalográfica gerada para dissertações de mestrado e teses de doutorado defendidas na UnB. A Biblioteca Central é responsável pela ficha, mais informações nos sítios:

http://www.bce.unb.br
http://www.bce.unb.br/elaboracao-de-fichas-catalograficas-de-teses-e-dissertacoes

Esta página não deve ser inclusa na versão final do texto.

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

Verification of Dynamic Authorization Protocol

Felipe Rodopoulos de Oliveira

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof.a Dr.a Cláudia Nalon (Orientador)
CIC/UnB

Flávio L. C. de Moura
CIC/UnB

Marcelo Menezes de Carvalho
ENE/UnB

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, 29 de Novembro de 2017

Dedicatória

Agradecimentos

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 é uma estratégia interessante, onde uma chave compartilhada entre o banco e o usuário é estabelecida para futuras mensagens. Para 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, escaneando 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, pretendemos realizar a especificação e verificação formal do protocolo, usando uma técnica de prova de teoremas bem estabelecida, o Método Indutivo. No fim, esperamos ter um dos dois resultados: um certificado formal de correção das crenças de segurança do protocolo ou um contra-exemplo válido para a falha do mesmo.

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 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 further being used in future messages. 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 intend to formally specify and verify this protocol, using a well known theorem proving technique, the Inductive Method. At the end, we hope to have one of either two results: a formal correctness proof of the protocol safety or a solid counter-proof of its failure.

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

If computers were a well welcomed 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, but do not provide a factual proof of nonexistence of flaws, 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, this method will be explored and putted 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 will be formally specified and verified, hopefully giving a formal and trustful certification for its users' base.

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 security systems, criminals are still able to perform successful attacks on these infrastructures [7].

Promising works appeared over years using OTP schemes [8, 9], being them the basis for DAP. Still, no formal verification was done in any of these approaches. Therefore, there is no guarantee that they will not fail. Allied with correct 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 Goals

Based on the previous sessions, this work mainly tries to provide a formal specification of the *Dynamic Authentication Protocol*, and its formal verification, using the Inductive Method. Therefore, a proof or a counter-example of the protocol correctness will be obtained, leading to a formal and mathematical corroboration for the system.

Additionally, it is intended to incorporate new material for the base of the Inductive Method proofs [10], using the automatic assistant system Isabelle [11], contributing to its community.

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 ??** presents the intended methodology for the future work, displaying the intended approaches and already done tasks;

Chapter 2

Security Protocols

This chapter surveys the main aspects of security protocols, explaining its common properties, goals, and frequent attacks 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 [12]. More definitions on security protocols can be obtained at [13] and [14].

As a communication agreement, security protocols are inherently concurrent, leading to a big 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 difficult 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 are subtle: they depend on the environment, resources and the protocol architecture itself. Many goals may be present as, for instance, authentication, key establishment, confidentiality, integrity, availability and others. Some of those goals may have varying definitions in the literature, like authentication [12]. Nevertheless, some properties and concepts can be universal among security protocols, as well as some goals are a consensus in the literature [14].

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 [13]. 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 [15]. 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 [13]:

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 a 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 participants hold certified public keys;
3. An on-line public infrastructure may be used, where 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 the 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 [13].

2.4 Notation

Through this document, we will need a consistent notation for expressing protocol actions and symbols. We will use a system based on the one presented in [3], which brings aspects from the seminal work of [16], since it is our main reference method for verification in this work.

Primitive data instances, such messages, timestamps and *nounces* 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 further examples or 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 distinguishes protocol messages from sets and represents a 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 hypothetic 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 K_B.

As a background example intended to provide a 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

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

Figure 2.1: Example protocol for notation undestanding

Alice and Bob), wants to communicate securely. For that, they generate a session key K_{AB}, aided 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 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}, who Alice readily forwards to Bob, along with her identity. Over the next sections, we will identify problems and redesign the protocol.

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

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 required properties among security protocols. As seem 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 ciphertext set $\mathcal{C}$ and three main algorithms:

- **Key Generation:** outputing 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 ciphertext $c \in \mathcal{C}$, which is defined as $c = m_K$;
- **Decryption:** taking a ciphertext $c \in \mathcal{C}$ 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}, e.g. $K = K^{-1}$, it is stated that they are *symmetric* keys. Otherwise, they are *non-symmetric* keys, normally engaged in a process of public key encryption, commonly used nowadays [14].

Secrecy consistency can be discussed with respect to some degree of flexibility. An active spy should not be able to learn something on generated ciphertexts. The property of *semantic security* states that if an attacker can compute something from a ciphertext, it must also be able to compute it from its equivalent message. Stronger than that, *non-malleability* makes impossible to transform a given ciphertext in a related one without knowing the input plaintext. A detailed discussion on this topic is provided in [12].

Considering our previously presented example, it is easy to state that confidentiality is compromised in the protocol. The session key K_{AB} is transmitted in cleartext 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, 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 a correct implementation of this property among protocols. Such topic is well discussed in [17].

Here, we recall some definitions provided in [18] and [14]. Authentication can be defined as the oath on 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.

Thus, the received message must not get altered in any way. This assurance implies the property of **integrity**, which means that no message can be corrupted along its way to the recipient. For this reason, authentication and integrity are commonly associated into a single property. Ensuring such feature may be achieved by the use of manipulation detection codes (MDC) or message authentication codes (MAC) [19].

Other stronger properties may be needed. One is the pledge of the received message to be the first and original one, avoiding the acceptance of repeated messages. Concerning the protocol which will be studied in Chapter 4, this notion is crucial. A good discussion on this is also given in [17].

Now we focus on our working example. As stated, the attacker can intercept and modify messages, since it has full control of the network. We consider two situations, presented in Figures 2.4 and 2.5.

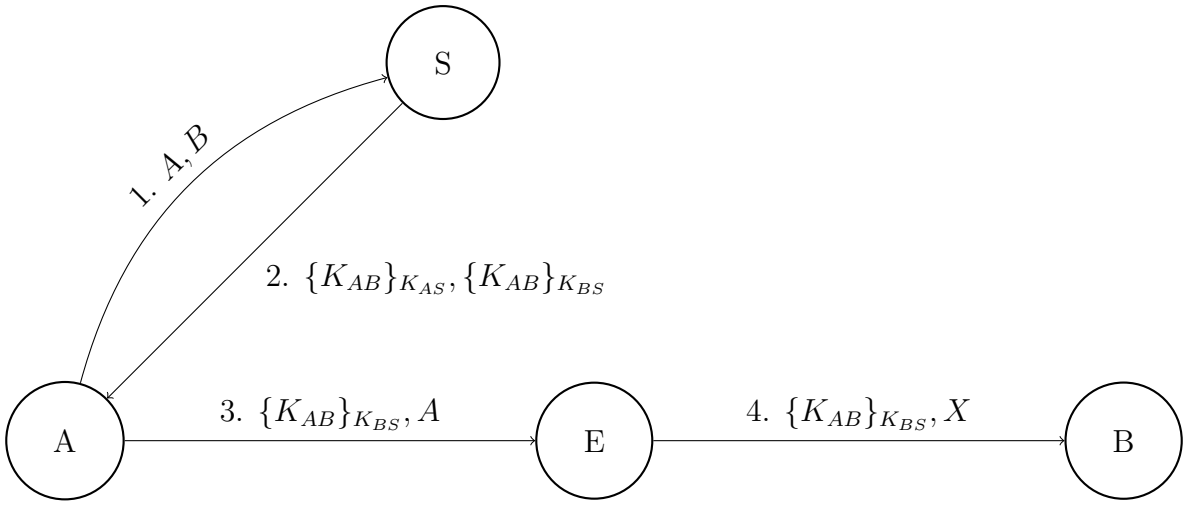

Figure 2.4: First attack possibility against Protocol 2.6

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.

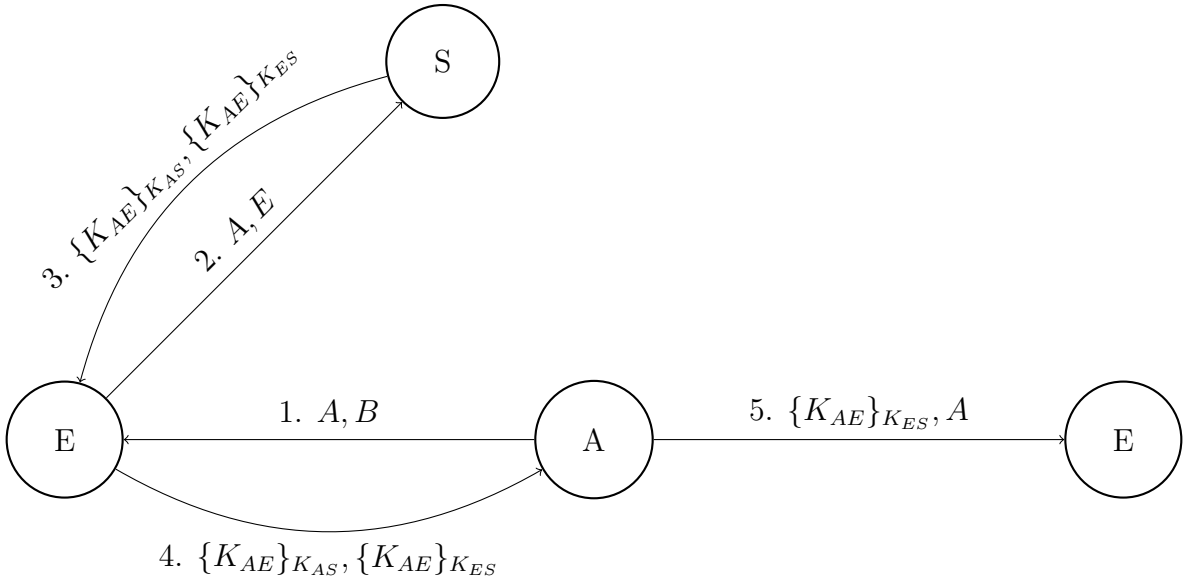

Figure 2.5: Second attack possibility against Protocol 2.6

We rewrite 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.

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

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 quality differs on authentication, where the former tries to provide a identity assurance to the system and the latter produces a formal and enduring proof of authorship on actions and messages, which can be check 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 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.

Based on common errors found in published protocols, the work in [20] presents interesting guidelines for designing reliable security protocols. Although these guidelines are not sufficient for building 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 over the working example, considering the freshness property. A possible attack, extracted from [13], 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 [21], which uses nonces, ciphered with shared keys, for guaranteeing freshness of the session

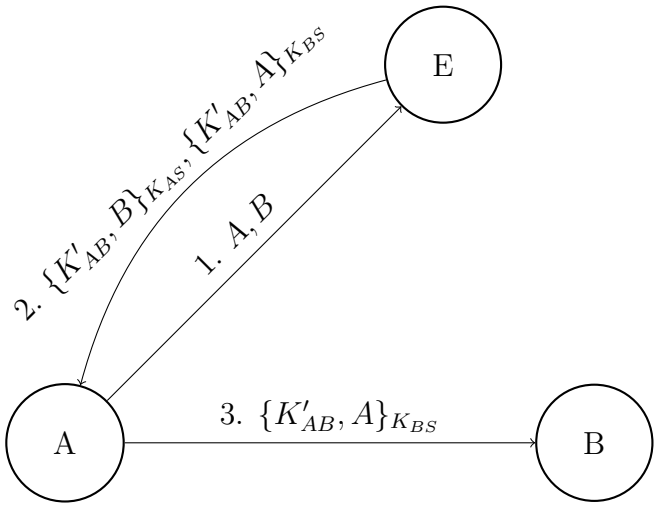

Figure 2.7: A replay attack on the protocol described on Figure 2.6. Key K'_{AB} is obtained by the Spy from by eavesdropping previous protocol runs

key. However, this approach was proven to contain failures [22]. That said, we will focus on a reliable solution, also described in [13] 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 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 sound.

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 [13], 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 on attack methods. Many attacks against systems and protocols involves 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 brings a brief introduction on formal verification and a more detailed explanation on the selected approach to be used in this work: the Inductive Method. Its syntax will be restrained for comprehension of the proposed problem.

3.1 Formal Verification

Using the previous definitions for security protocols, it is easy to perceive 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 designs [3]. However, not finding an error does not mean that a certain model does not hold one. Such method was failing to identify critical flaws in major security protocols, where the classical example of the Needham-Schroeder protocol [21] can be cited. The techniques in [22] exposed the protocol's limitations, despite the authors efforts on its specification and informal verification.

This scenario was optimal for the rise of formal verification on security protocols such as in the works presented in [23], [24] and [16], allowing a mathematical and profitable procedure for reasoning about abstract protocols models. Thus, it is important to emphasize that the method focus on the protocol properties and rules, and rarely on its execution.

Formal verification requires the definition of such protocols in a proper language, suitable for the application of such methods and deservedly automated through computational methods. Formal methods can be divided in two major categories, according to [13]:

- **Model checking**: protocol behaviors 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 its correctness;

- **Theorem proving**: postulating the protocol as a theorem, taking into account *all* possible protocol behaviors, and checking its correctness through the verification of the theorem claims. This method is more suitable for proving the correctness of the protocol rather than finding possible attacks on them.

Both methods are aided by computer assistance, where theorem proving may be a less automated approach. Concerning the protocol analyzed in this document and our goal to prove its correctness, it is trivial to guess that our chosen method fits the theorem proving set.

3.2 Method Introduction

Induction has been a commonly used mathematical proof method among theorem proving literature. Proving inductively a certain property $P(n)$ demands a base case $P(0)$ and the validation of general formula $P(i) \Rightarrow P(i+1)$, for all $i > 0$, concluding that $P(n)$ is true, for any positive integer n. It is reasonable to apply this procedure on security protocols, where one can check if a security property is preserved along a large population of participants and all possible system executions. The work of L. Paulson [2] was the first to propose such idea and it has been extended by G. Bella in [3].

The model considers a protocol $\mathcal{P}$ and an unlimited set of agents, which includes the Spy, an omnipotent entity. Agents actions result in network traffic, which is abstracted as a **trace**: a history of events in the network, displayed in reversed chronological order. Therefore, we can define the set P, containing all possible traces of infinite length as a formal model of the environment where the protocol ran. Such set is defined inductively by the rules of $\mathcal{P}$.

This approach has much in common with CSP formalizations [14], although the traces considered here are infinite. Besides, the method only focus on security properties, not considering liveness (denial of service), for example.

3.2.1 Isabelle

One of the most interesting characteristics of the Inductive Method is the possibility of proof mechanization by means of the theorem prover Isabelle [11]. The generated proofs are machine checkable, enabling a deep understanding of its anatomy and easy error maintenance.

Isabelle is a generic theorem prover, meaning that it is able to reason over several formal systems, in an interactive way: its not entirely automatic, requiring certain human interaction for concluding proofs. At the same time, it offers automatic provers, which can deal with several simple proofs, and it also provides simplifiers methods, enabling rewriting of expressions with adequate arithmetic procedures.

The version in which this document is based is Isabelle/HOL [25], based on high-order logics. This kind of logics can quantify over functions, predicates and sets, based on a strongly typed formalism. Moreover, most of the proofs may be produced inductively, where often subgoals are left for being simplified. In case any subgoals are left unsimplified, the user must provide lemmas for complementing the rewriting rules and finally proving the main theorem.

Due to that, some proofs may take time and patience. If a user fail to find a proof for a certain property, this may not be a certification of proof unsatisfiability. Despite, the proof or somes of its aspects may be built on top of wrong formalizations, meaning the user is not skilled enough.

Along this chapter, Isabelle syntax on security protocol properties will be presented. Basic operations and symbols will be explained when convenient, but aiming document readability, most part of the syntax will be omitted. An extensive and reliable literature on theorem proving on Isabelle is available in [25] and security protocol proofs in [3].

3.3 Formalizing Entities

For a suitable verification of the system, a prior definition of it is necessary. In the context of Isabelle, a theory is such specification, containing a set of definitions and theorems which need to be proven for a solid formalization. Isabelle has a large collection of theories, within many areas, including the one concerning this work [10].

The `Auth` library from Isabelle/HOL distribution [10] provides a set of theories, which consist of sets of definitions and theorems for reasoning about security protocols, enabling a researcher to build proofs with solid base and quick setup. Here, we introduce such concepts, which are essential for understanding the inductive way to work with formal verification of security protocols.

3.3.1 Agents

This is the basic type for defining participants in a protocol session. The model considers an infinite set, due to the possible large population of active peers. A simple bijection with the set of natural numbers can shape the main principals of a protocol: Friend, Spy and Server. Such entities can be compactly defined in Isabelle syntax as follows:

$$\texttt{datatype agent} \triangleq \texttt{Server} \mid \texttt{Spy} \mid \texttt{Friend nat}$$

The `Friend` definition is the common peer in a session. Since we can have a positive infinite number of participants, we simply define the i-th peer as `Friend` i. The entity `Spy` is used as a formal representation of a threat model (which will be defined later), defined with a nullary constructor. Finally, the `Server` entity, often represented as `S`, is a commonly trusted third-party agent, used for symmetric key generation operations. Additionally, this peer has access to all agents long-term secrets, i.e., the private keys of other agents or servers.

Thus, a characterization of compromised agents is made. These are embedded in an unspecified set named `bad`, where the Spy is also included. All elements within this set have their knowledge set exposed to the Spy, which includes private keys, shared keys with the Server and any other secrets. This property takes action along all protocol run, which means that if the agent notes any information during a session, the Spy is able to retrieve it.

3.3.2 Cryptographic Keys

Keys are the base entity for providing encryption and they have several groups. Here, they are declared as a free type, but interpreted as an integer and each kind of key has intrinsic properties, as follows.

For cryptographic symmetric keys, exchanged with trusted servers - the `Server` entity - the key is previously stored by an agent. So, it is defined as a agent dependent value:

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

This declaration is also applicable for asymmetric keys, where the notation is `priK` for private one and `pubK` for the public. This notation may differ when a signature key is used.

Symmetric keys are a bigger scope for other keys, belonging to a proper set `symKeys`. One example are session keys, which are used by peers for establishing a single private session of communication. Therefore, the valid definition is:

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

Such definition implies that the set of shared keys is different from the one containing session keys. Any other kind of keys not mentioned here must be manually defined.

3.3.3 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, the constructor of a `Message` variable accepts many kinds of formats, as stated below:

```
datatype ≜ Agent agent
         | Nonce nat
         | Key key
         | Mpair msg msg
         | Hash msg
         | Crypt key msg
```

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. It is important to notice that the `Mpair` constructor enables the definition of recursive concatenation of messages.

Another important concept is the `Crypt` directive. This is used for representing encryption of messages, such as message M, with key K, denoted by `Crypt K M`. In this model, encryption is perfect and collision-free, so the M is only accessible if the peer have the key K.

3.3.4 Events

Events are a basic entity along protocols. Here we present the definition of the three main kinds of events, 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.

However, it is necessary to settle the difference between `Notes` and `Gets`. The former relates to the action of noting down fragments of data from received messages, which is

particularly used by the Spy for deriving keys sent in inaccurate peers messages. The latter is technically equivalent to the `Notes` action. At the same time, it does not implies the storage of an information, not enriching the set of known entities of an agent or of a given trace. This leaves the definitions and rule construction leaner.

Now, the definition of a **trace** can be done. Intending to keep an history of the network, a trace is a list of events that occur in the network, displayed in reverse chronological order. Therefore, every single event that occurred in the network while the protocol was active must be present in the trace. Besides, the trace must respect the protocol model, since it can only be constructed upon a valid run of the given protocol.

3.3.5 Nonces, Timestamps and Guessable Numbers

Security protocols often resort on numeric entities for soundness of some properties. These numbers are modeled as well, but they come with premises for being compatible with reality.

Nonces are random numbers, hence they are modeled this way. In the Inductive Method, they are shaped as a big random natural number and are unguessable. Additionally, they are designed to provide freshness and, consequently, are supposed to be out of the set which contains all messages that are clear in the network. This set, named `used`, will be defined in a latter section.

Guessable numbers are the opposite. Intuitively, they are considered natural numbers, but they are guessable. As a result, they are always known by the Spy. Thus, the message datatype is extended to cover this primitive, with the declaration `Number N`.

Timestamps relies in the notion of time in protocols. Such concept is provided when each event of a trace carries the time instant when it happened, so it is straightforward to define a valid time model for the local trace. The authors note that the temporal relations are irrelevant between traces, since they are different runs of protocols. Also, this representation excludes the concept of concurrency: two events can never happen simultaneously in a trace. It is argued that this notion of concurrency can be represented when two events are sequentially interleaved in two different traces. Finally, the definition of current time is done:

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

This definition makes sense, since the time of the last event dictates the current time on the trace. Such definition need the function below:

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

3.3.6 Operators

In the Inductive Method, the actions of reasoning about and composing messages are treated as operations over sets of messages. In that way, the operators `analz`, `synth` and `parts` are defined upon operations on this kind of data:

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

The `analz` operator denotes the act of inspecting the components of a message and then breaking it up. Therefore, using the set of messages H, the set `analz` H uses the following rules:

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 ciphertext can only properly be analyzed and further retrieved in possession of its decryption key.

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

It is significant to mention the set `analz(spies)` *`evs`*. This set holds all data that the Spy could gather from the network in the trace *evs*, either by decomposing it from another messages or obtaining from decrypted ciphertexts, defining its current knowledge set. 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;

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

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.

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

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. In this operator, the extraction of a ciphertext from a encrypted message does not require a key. Thus, we can relate both operators in such way: $\texttt{analz} \subseteq \texttt{parts}$. Such concept simulates unbounded computational power and so, the rule is rewritten as follows:

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

Complementing the crucial sets for the method, it is introduced the set `used`, which is used to delimit the set of already used message components, belonging to some agent initial knowledge or which have appeared at some moment in the network. Such concept is important due to the necessity of working with the freshness concept and properly modeling it. The definition of the set follows:

$$\texttt{used} : \texttt{event list} \longrightarrow \texttt{msg set}$$

3.3.7 Initial Assumptions

In order to be compatible with real world context, the Inductive Method provide some basic initial definitions concerning several aspects of the environment and principals.

First, a formalization of agents initial knowledge set is provided. Such definition is important not only for stating the protocol initial premises, but for constructing the initial knowledge of the attacker, based in the compromised agents. Thereby, we define the function `initState`, which receives an agent as argument and returns the agent's initial knowledge message set:

$$\texttt{initState} : \texttt{agent} \longrightarrow \texttt{msg set}$$

For every different kind of agent, we have a different type of initial knowledge set:

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

$$\begin{aligned}\texttt{initState Server} \triangleq &(\texttt{Key range shrK}) \cup \\ &\{\texttt{Key (priEK Server)}\} \cup \{\texttt{Key (priSK Server)}\} \cup \\ &(\texttt{Key range pubEK}) \cup (\texttt{Key range pubSK})\end{aligned}$$

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

$$\begin{aligned}\texttt{initState (Friend } i\texttt{)} \triangleq &\{\texttt{Key (shrK (Friend } i\texttt{))}\} \cup \\ &\{\texttt{Key (priEK (Friend } i\texttt{))}\} \cup \\ &\{\texttt{Key (priSK (Friend } i\texttt{))}\} \cup \\ &(\texttt{Key range pubEK}) \cup (\texttt{Key range pubSK})\end{aligned}$$

3. The Spy knows all secrets from the agents who belong to the compromised set, which includes shared and private keys, including herself, since she is included in the set of compromised peers. Additionally, she also knows all public keys.

$$\begin{aligned}\texttt{initState Spy} \triangleq &(\texttt{Key shrK bad}) \cup \\ &(\texttt{Key priEK bad}) \cup (\texttt{Key priSK bad}) \cup \\ &(\texttt{Key range pubSK}) \cup (\texttt{Key range pubSK})\end{aligned}$$

Concerning the attacker formalization, some properties are defined to properly model the attacker behavior based on [15]. The earlier definition of agents formally characterizes the Spy as a legitimate agent. The definitions of `synth` operator details how any agent, including the Spy, are able to generate all kind of legal messages, but excluding nonces, which are treated as unguessable elements. Now, the property that gives full control of the network to the attacker must be described. For that, the function `spies` is presented, where its syntax follows:

$$\texttt{spies : event list} \longrightarrow \texttt{msg set}$$

And its recursive definition is provided:

1. The Spy knows her initial state at any trace, including the empty one;

$$\texttt{spies[]} \triangleq \texttt{initState Spy}$$

2. Any message sent by anyone in the network is know to the Spy at that trace;

$$\texttt{spies ((Says A B X) \# evs)} \triangleq X \cup \texttt{spies evs}$$

3. Any message noted by a compromised agent in a trace is known to the Spy at that trace.

$$\texttt{spies((Notes A X) \# evs)} \triangleq \begin{cases} X \cup \texttt{spies } \textit{evs}, & \text{if } A \in \texttt{bad} \\ \texttt{spies } \textit{evs}, & \text{otherwise} \end{cases}$$

Such definition also aims to provide a formal scope for the dynamic nature of the Spy knowledge during the protocol run.

3.3.8 Assembling the Model

Since all elements are properly defined, we are able to formally define the protocol model. As previously stated, the model will be structured as an unbounded set of lists of traces (events), where each trace will represent a possible network history, induced by the protocol.

Since the technique relies on induction, the base case, where no events occurs, is the first defined rule. Furthermore, each protocol step is formalized, where the main strategy consists in extending the previous rule, adding the new event which will properly formalize the legal step or any valid action. Note that, if a protocol has n steps defined in its specification, the model must have n `Says` events in its formal model, at least. At the end, a vast possibility of runs can be modeled which this method.

We define a constant, which contains the model traces. Each rule is composed by a set of preconditions and a postcondition. The latter will add the rule event at the beginning of the trace, suiting the trace reverse order condition. At the end, the rule add the possible trace to the constant.

We use the previously defined protocol given in Figure 2.1 and its proper model, Model 3.1, as examples for illustrating the model construction process. In this example, each rule can refer to a generic trace, but it is also possible to use a single trace as well. For both rules formalizing the protocol steps, `DSP1` and `DSP2`, its possible to see that if the initial trace, *evs1* and *evs2* belongs to the protocol model, then its extensions, defined at the end of the rule, will also belong to it.

```
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:
⟦ evs0 ∈ dsp;
  Says B A (Crypt (priSK B) {|Nonce Na, Key K|}) ∈ set evs0 ⟧
⟹ Notes Spy {|Nonce Na, Key K|} # evs0 ∈ dsp
```

Figure 3.1: Formal model for the protocol displayed in Figure 2.1. Figure is already following Isabelle's syntax

Two extra rules are defined. The first, *Fake* relates to the spy ability to send any message that she can fake, i.e., any message $X \in$ `synth(analz(spies` *evs*`))`. Such rule is consistent with the Dolev-Yao attacker abstraction. The last rule, *Oops*, abstracts the loss of a session key to the Spy, which is treated as a local security breach. This flaw may affect the whole system and this rule allows to detect such situations. Such rules may arise as a subgoal in the Isabelle system, needed to be manually stated, implying that the protocol has a natural flaw.

Using this model, a large gamma of traces can be constructed, although not every conceivable trace can be derived from it, since the model should impose some restrictions to the traces. Also, any desired protocol property should be verified against each model rule, a process which Isabelle helps to automatize. Thus, we defined that any property of the model is an attempt to formalize a protocol goal.

3.4 Goals Verification

Having a consistent model of the protocol is the first step to its full verification. Thus, it is necessary to identify the model properties and correlate them with its goals. Since it is an inductive method, a given property must be checked against all system points, including its rules. The formalization and latter verification of the goals innate to the protocol are the last part of the Inductive Method.

3.4.1 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 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 proof. Many of these proofs are aided by Isabelle's simplifiers, which can easily reason about an given property inductively.

3.4.2 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 (or private 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, it is easy to derive rules such:

$$\text{Key(priK A)} \in \text{parts(spies } evs) \leftrightarrow A \in bad$$
$$\text{Key(priK A)} \in \text{analz(spies } evs) \leftrightarrow A \in bad$$

3.4.3 Confidentiality

Holding confidentiality in a protocol can be simply translated as to preventing the disclosure of certain message to the Spy, that is, a message X cannot belong the Spy knowledge set. Since messages are usually protected by encryption, this property is highly related to the use of keys. Precisely, the confidentiality of session keys is a major issue for the protocol confidentiality, because if a spy obtain a given session key, all messages encrypted with it could be easily altered by her. If we define a session key K_{AB}, we have to be certain that at the end of all traces the key does not belong to the knowledge set of the Spy. That said, we have the following:

$$\text{Key } K_{AB} \notin \text{analz(spies } evs)$$

Note that, if key K_{AB} 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. Nonces are commonly used for computation of other protocol details, as keys and MACs [3]. Therefore, the model must provide confidentiality to all nonces used for composing important protocol pieces.

3.4.4 Unicity

The creation of fresh components in protocol is vastly used. It is seen in the production of 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 the events are identical. Further, events containing fresh message components cannot occur more than once, otherwise they violate the unicity concept.

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.

3.4.5 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. Conversely, 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 verifications.

3.4.6 Authentication

As discussed in Section 2.6, authentication may assume many properties. Supposing 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 messages 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 simply 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 bigger 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 [26] that this property is, indeed, stronger than authentication and thus, authentication itself relies on key distribution. Additionally, the authors of the Inductive Method proves that if authentication holds, so does key distribution, specifically non-injective agreement on a session key.

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 banking application, creating an encrypted authorization scheme for each banking transaction over a previously authenticated session between User and Bank.

First, it must establish a shared key between User and Bank, which will be stored both at the User's uncompromised smartphone and the Bank Server. There are two different ways to generate such key, a symmetric and an asymmetric method, where such operation is done 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 his smartphone, which interprets, checks and displays the operation, asking for its authorization. If 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 done 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.

4.1 Internet Banking

Internet banking is an online environment which provides a communication channel between bank and 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 proved 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 - user, bank and Internet - resembling a client-server model architecture. This pattern is commonly used over the literature [6].

At the client side, many communication channels 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 highly sought target among hackers and other ciber-criminals [27, 6]. Several techniques for credential robbery and transaction manipulation for internet banking systems have been developed over the years, where a formal classification attempt can be find on [27] and [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 his 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 envolves the robbery 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 [27] and [7], the authors provides a listing of current common security models found among banks.

Still in [27] and [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 will 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 the biggest bank in Brazil, *Banco do Brasil*. 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 an unique transaction on each run. The user must be already authenticated before prompting a transaction authorization

and private authentication keys must be previously exchange between server and client. In a protocol run, the user receives a series of challenges, in order to verify the operation, where those challenges contain pieces of information about the given transaction.

One of these tasks envolves 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 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 serie 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 transfered 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 define 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 store 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 [28] for providing an extra authentication layer for protocols.

4.3.1 Initial Assumptions

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

1. The Spy has a full control of the network, being able to intercept, block, produce or inject any kind of message, by any agent enjoying the channel;
2. The User personal computer belongs to the set of compromised agents, i.e. the Spy has access to any information store, 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 thesis 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 ilegal 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 focus 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 uncompromised. 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, he must be previously authenticated in the Internet Banking system, on his personal computer.

The Server receives the request and generates three entities:

- The shared key K, which will not be sent yet. This key is based in the User's ID and the operation request timestamp;
- A nonce N_K, being a random sequence of 2^{288} bits;
- A key K', which is obtained by the binary summation of the previous entities, i.e. $K' = K \oplus N_K$.

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

At the ATM machine, the user must present his personal smartcard and password for authentication. Then, the terminal requests the second part of the key to the Server, who replies with key K'. Once more, the ATM shows the message as a QR-code, which is scanned by the User's smartphone.

Now, the smartphone can compute key K, using N_K and K' as inputs. At the end of the computation, the inputs are discarded and K is stored at the User's smartphone. Finally, 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 [29]. The User must issue the generation of key K at his 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.

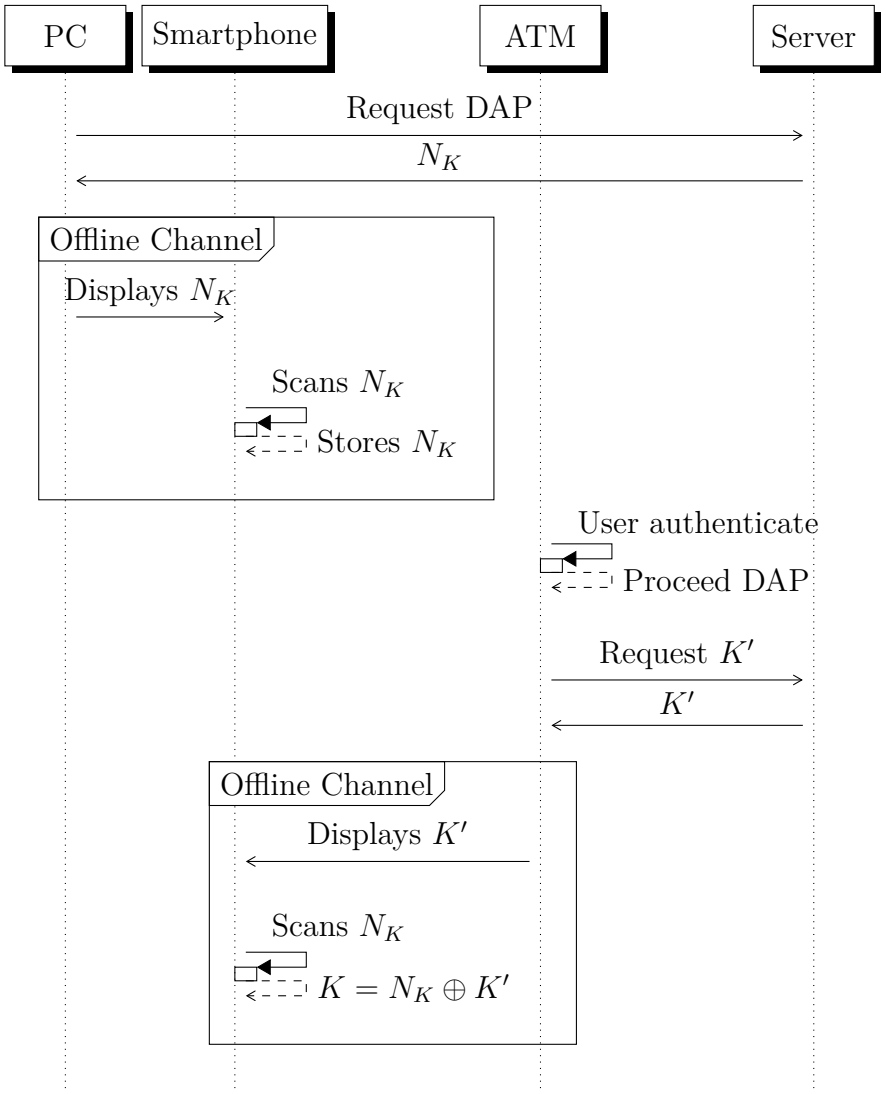

Figure 4.1: Diagram of a symmetric method for generation of key K

As soon the Server acknowledge the User public key, he signs it with his private key sK_S and sends his public key to the User. When the latter notes the Server's public key, he stores it, encrypting it with his 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 [30], 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 [31], protecting messages or its components over clear channels.

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

1. The User prompts a banking transaction through the Internet Banking application, over a personal computer. Hence, the User must be previously authenticated. The queried transaction is abstracted in message m, which is sent in cleartext to the Server;

2. Once the Server receives m, it generates a positive natural random nonce r, named **transaction authorization nonce** (TAN). This nonce will uniquely identify the transaction contained in the message, acting as a reference for further authorization;

3. Then, the Server encrypts r using key k_2 and AES encryption scheme, generating $r' = E_{k_2}(r)$. Further, the HMAC-SHA1 hash h_S is produced, using key k_1 as the encryption key. The cipher is obtained by concatenation of m and r', leading to $h_S = H_{k_1}(m, r')$;

4. The server composes the final message to be sent to the User, concatenating m, r and h_S. Hence, it obtains $m' = \{\!|m, r', h_S|\!\}$ and sends it to the User;

5. At the User side, the personal computer receives m', visually displaying it to the User as a QR-code. The User may now use his smartphone to scan the code, retrieving m and r' by decomposing m'.

6. The device performs the consistency check, calculating $h_U = H_{k_1}(m, r')$ and comparing it with h_S. If the ciphers match, then the message is reliable and the protocol proceeds. Note that the smartphone does not need an active internet connection, since it does not use the network at anytime;

7. The operation contained in m is presented to the User for proper verification. If a strange operation is received, they can simply drop the process. Otherwise, they can proceed and the nonce r is displayed at the screen. Such code can be easily obtained by decryption of r' with key k_2, stored in the smartphone, i.e. $D_{k_2}(r') = r$;

8. The User can now input code r on their PC and send it to the Server. The Server compares it with its r. If the check succeeds, it sends a confirmation message to the User and end the protocol.

4.6 Security Claims

Finally, we discuss the correctness arguments concerning security on DAP. The authors postulate such guarantee as an 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 is totally compromised;
- The ATM machine is not compromised;
- The User smartphone is not compromised;
- The Server is not compromised, but the Spy has full control of the network;

Additionally, note that the smartphone never sends any data over the network, so there is no data leak at this device. Based on that, authors claim that any secrets held on such devices are secured.

Thus, concerning a protocol run, if the Spy intercepts a message, alters its data and relays it to the valid user, the latter still must verify the transaction through the QR-code scanning, being able to drop an unknown operation.

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

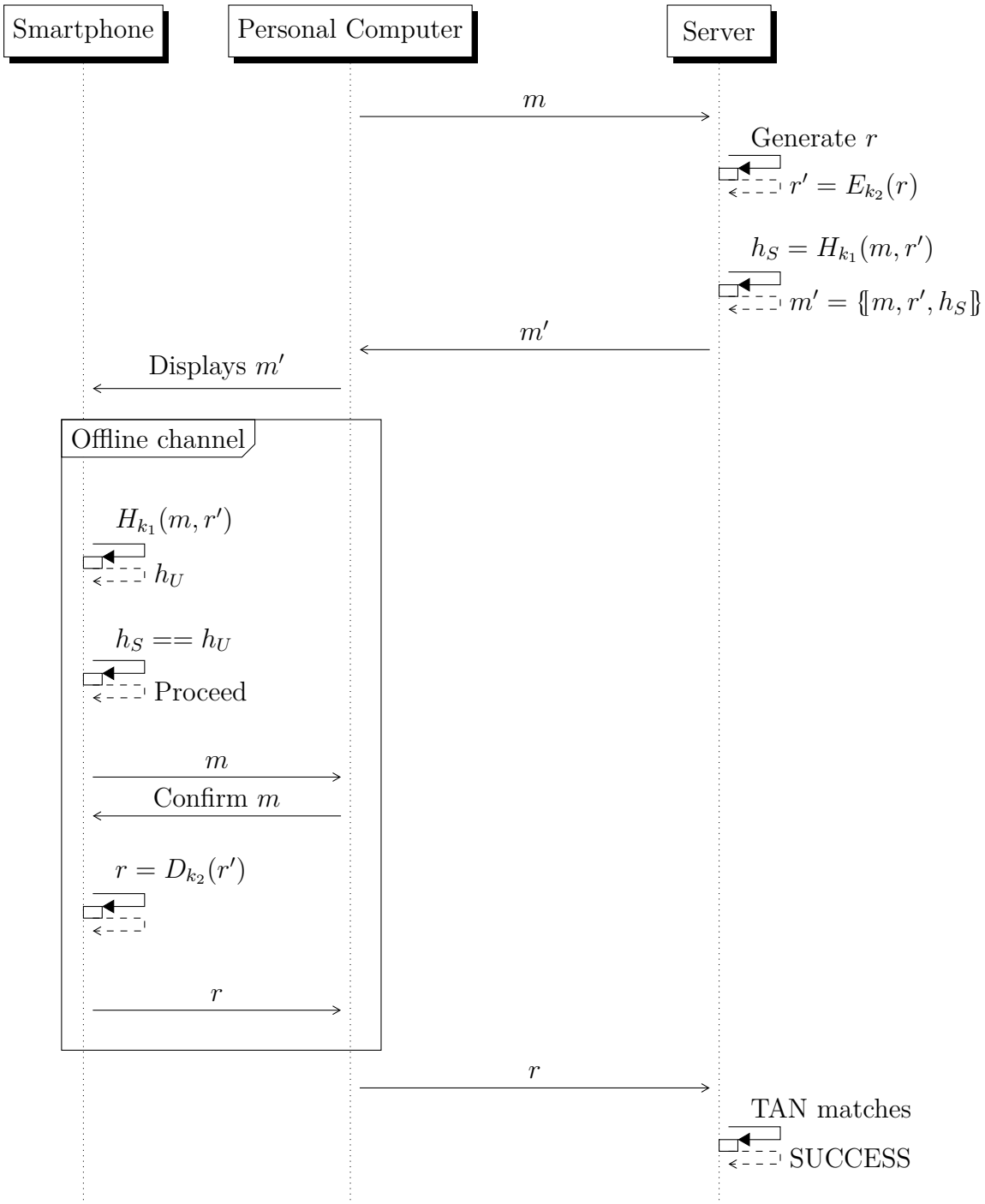

Figure 4.2: Diagram of a message transation in DAP

Chapter 5

Offline Channel Formalization

As seem in Chapter 4, DAP main strategy for securely authorizing transactions is providing a verification phase between the human User and his smartphone device, where the transaction will be revised and further confirmed. All these operations takes place in an offline physical channel, which is supposed inviolable by the Spy. Moreover, the DAP specification makes strong assumptions about the smartphone security model, which needs deeper research.

We stress that this kind of channel and the smartphone peculiarities are not properly formalized in the Inductive Method theory. In order to verify the protocol, we need to provide a reliable and suitable model for this context. In this chapter, we provide this formal model, taking into account all the real world details surrounding these entities, adapting the protocol assumptions into a conceivable scenario.

5.1 Context Analysis

The DAP specification describes the User smartphone as a key component for the protocol operation and, further, vital for its security. Meanwhile, 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 done by the smartphone on the hash verification phase and decryption of the TAN;

2D code reading capability: it must be able to scan and decode any presented QR code through a physical optical reader. In the protocol, such action is needed in order to receive the pieces of the symmetric key K and further transactions in the User personal computer screen;

On-screen data output: in order to show any received message and further operations to the User, the device must provide a visual physical channel to display such data. A screen is the selected way in the specification, due to its clarity;

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 action on the phone.

These actions can be embedded on a dedicated hardware, which could be issued by the bank to the protocol users. Each client would have his own device, already packed with any required encryption key, establishing the proper link between Bank and User. Similar schemes are already implemented by banks: 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 [32, 33].

However, the protocol authors enjoyed the convenience of the smartphone. Not only the device appropriately fits the requirements for the protocol desired hardware but also has a strong and known acceptance among users around the world REFERENCIA. Also, it has the ownership link with its user. Therefore, the device is a suitable choice.

5.1.1 Smartphone Scope

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

Recalling the DAP specification, it is known that the User's smartphone is the entrusted entity for securely holding long-term secrets, shared between User and Bank. Such pair of shared keys will create the proper link between User and Bank and, for this reason, note that it is important that the User uses his own smartphone for establishing the shared key. Plus, the specification also states that any communication between the smartphone and other devices must be done by visual means: either output data through a screen or obtain 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 an offline, out-of-band and non-interposable channel [1].

However, it is important to note that the smartphone performs more actions than the protocol needs. Due to its set of capabilities, the smartphone can act like a personal computer, even connecting to the network channel. 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 [34, 35], enabling remote operations by an attacker, as well.

Moreover, it is reasonable to consider the situation of theft, a scenario which is effectively contemplated in Inductive Method extended theory [3].

Therefore, in contrast with DAP specification, but in accordance with the formal method for verifying a protocol, we must consider the real world scenario cases, where the Spy can exploit or obtain the User smartphone. We stress that these situations provide different sets of information to the Spy, presenting two distinct formalization for both of them in further sections.

5.1.3 Security Context

The offline communication environment depicted in Section 5.1.1 restrains the Spy from its omniscient features, introducing a channel where she does not have full control. However, with the new postulations introduced in Section 5.1.2, a feasible manners to exploit some protocol entities can be found.

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 a regular computer, the smartphone can be exploited in such manner so that the attacker can disclose stored data, eavesdropping communication, faking data, and 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, perceive that the offline channel remains secure and what is compromised it is only the User device.

Finally, the situation of smartphone robbery must also be taken in account. Here, the Spy could not only operate the smartphone but, if skilled enough, exploit the device, obtaining the same privileges as if the smartphone is compromised. Hence, in such situation, we indicate that such 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 1 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 for 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, identifying the User towards the Bank. This entity is defined in a software context, in the smartphone memory, thereby it may be exploited. Furthermore, a hardware link between User and smartphone can also be found in the SIM card, present in any smartphone in order to it receive a 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 comprehend 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 comprehend how such aspects must be formalized in our final model:

- **Theft:** mobile phones are highly susceptible to robbery, considering that its reduced size also increases its risk to loss and stealing. Hence, we need to formalize devices that are robbed 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 `bad`, 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 ways.

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 REFERENCIA. Hence, we state that if a smartphone is robbed, it is automatically compromised:

$$stolen \subseteq bad$$

5.2.2 Keys

A discussion in smartphones keys is necessary due to the nature of the device itself and the QR code technology involved in this work.

Several smartphone's operational systems present to their users the possibility to be operated until a passphrase is inputted, protecting them from any unwanted access. In this way, such key protects a stolen phone from an illegal access. However, it is important to notice that such security scheme is irrelevant for compromised smartphones, since the spy has full device control.

We also mention the QR code ciphering scheme that happens over some steps of DAP. The protocol standard 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 any operation of this kind is defined by the protocol. Pegar referências sobre QR code, que aparentemente, possui uma especificação proprietaria.

The definition for shared keys among agents is kept. Smartphones also work as a storage for its owner's long-term keys, specially in DAP. How such keys are stored does not require an explicit formalization, but only how they are handled by agents and its devices. Thus, any important definition on such knowledge must be made on agents initial knowledge and further operations over it.

5.2.3 Usability

Usability concerns the characterization of an smartphone operations, hence modeling this property affects both legal agents and the Spy. For defining legal or illegal uses, it is necessary to investigate which actions are interesting to focus, perceiving how the mobile devices communicate with agents.

We note that the set of actions of the smartphone is constrained to physical operation, so, in order to perform an illegal action, the Spy must have physical and direct access to

the User smartphone, meaning that, it must be stolen. Hence, the propositions involving smartphone exploitation will not add complexity in this particular case, since they do not involve such operations.

Legal actions are done by legal users, consequently it is easy to define the legal operation:

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

As discussed, the illegal action is constrained to physical actions, thus, for performing any of these, the Spy must physically posses the smartphone, by robbery, so:

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

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 does not need to use her own card illegally, given that such action would not result in any more relevant information to her knowledge set. Hence:

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

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
               | Gets_s smartphone msg
               | Outputs smartphone agent msg
               | Gets_a agent msg
```

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

Inputs

Gets_s

Outputs

Gets_a

The `Shows` event translates the act of an agent displaying some data on its screen. It is important to add the fact that the process which displays such data does not directly point such message to some other agent or smartphone, hence he simply displays the data and expects that someone scans it. Therefore, we do not add a recipient to the event. Plus,

Conversely, the `Scans` event needs two entities: the one which is reading the data and the one providing it. Therefore, the event is defined with the agent smartphone which reads an agent screen, receiving some message from it.

With such events, the process of displaying and scanning information from a screen is not only formalized, but properly correlated: a smartphone can only scans the screen of an agent with the `Scans` event if there was some agent which displayed some message with the `Shows` event in the trace.

The formal definition of the *used* function is extended for account the new events presented:

4. All components of a message that an agent displays on his screen in a trace are used on that trace:

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

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

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

The formal definition of the *knows* function is extended for account the new events presented:

1. An agent knows what he inputs to any smartphone in a trace;
2. No legal agent can extend his knowledge with any of the message received by any smartphone in a trace, since he already know them by case 4. However, if the smartphone is connect to the network and compromised, the Spy can learn any information received by the device.
3. An agent, including the Spy, knows what he is output from his smartphone a in trace. Also, the Spy knows what all connected and compromised smartphones output.
4. An agent other than the Spy knows what he receives from his smartphone in a trace.

5.2.5 Agents' Knowledge

Here, the definitions on agents' knowledge sets will be extended both on initial states and on further protocol operations. We consider the former first. Similarly to Section 3.3.7, a review on each kind of the agent's initial knowledge set is done.

1. The Server's initial knowledge set consists of all long-term secrets, both for those who abides in agents and smartphones. For this reason, we extend its initial knowledge set to cover such keys:

$$\begin{aligned}\texttt{initState Server} \triangleq &(\texttt{Key range shrK}) \cup \\ &\{\texttt{Key (priEK Server)}\} \cup \{\texttt{Key (priSK Server)}\} \cup \\ &(\texttt{Key range pubEK}) \cup (\texttt{Key range pubSK}) \cup \\ &(\texttt{Key shrK}\{A.\ Card\ A\})\end{aligned}$$

2. Each legitimate agent (`Friend`) initially knows all public keys and his own shared and private keys. Since secrets stored at the smartphone are not revealed to the User without interaction, his initial knowledge set remains the same as stated in previous sections;

3. The Spy knows all public keys and all secrets from agents who belong to the compromised set, which includes herself and compromised smartphones. Thus, secrets living on them are also disclosed to the Spy. We extend her initial knowledge set:

$$\begin{aligned}\texttt{initState}\ Spy \triangleq\ &(\texttt{Key shrK bad}) \cup \\ &(\texttt{Key priEK bad}) \cup (\texttt{Key priSK bad}) \cup \\ &(\texttt{Key range pubSK}) \cup (\texttt{Key range pubSK}) \cup \\ &(\texttt{Key shrK}\{A.\ Card\ A \in bad\})\end{aligned}$$

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, 29, 30, 34, 39

[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, 14, 15, 16, 26, 27, 40

[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

[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

[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, 29

[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, 29, 30

[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, 30

[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, 30

[10] 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. 2, 16

[11] Paulson, Lawrence C. and Tobias Nipkow: *Isabelle: A Generic Theorem Prover*. Springer, 1994. 2, 15

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

[13] 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

[14] 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

[15] 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, 22

[16] 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

[17] 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

[18] 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

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

[20] 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

[21] 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

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

[23] 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

[24] 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

[25] Nipkow, Tobias, Lawrence C. Paulson, and Markus Wenzel: *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. 16

[26] 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. 27

[27] 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. 29, 30

[28] 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. 31

[29] 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. 33

[30] 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. 35

[31] *Specification for the advanced encryption standard (aes).* Federal Information Processing Standards Publication 197, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. 35

[32] 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. 39

[33] 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. 39

[34] Zhou, Y. and X. Jiang: *Dissecting android malware: Characterization and evolution.* In *2012 IEEE Symposium on Security and Privacy*, pages 95–109, May 2012. 40

[35] 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. 40