PhD- FSTM-2020-59
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 03/11/2020 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Itzel VAZQUEZ SANDOVAL

A MULTIFACETED FORMAL ANALYSIS OF END-TO-END ENCRYPTED EMAIL PROTOCOLS AND CRYPTOGRAPHIC AUTHENTICATION ENHANCEMENTS

Dissertation defence committee

Dr Gabriele Lenzini, dissertation supervisor
Associate Professor, Université du Luxembourg

Dr Peter Y.A. Ryan, Chairman
Professor, Université du Luxembourg

Dr Luca Viganò, Vice Chairman
Professor, King's College London

Dr Pascal Lafourcade
Professor, Université Clermont Auvergne - LIMOS

Dr Fabio Martinelli
Professor, Instituto di Informatica e Telematica, Consiglio Nationale delle Ricerche

A tí Gildita ...
con todo cariño y admiración.

To M, C, N, G and A.
The driving force.

ABSTRACT

Largely owing to cryptography, modern messaging tools (e.g., Signal) have reached a considerable degree of sophistication, balancing advanced security features with high usability. This has not been the case for email, which however, remains the most pervasive and interoperable form of digital communication. As sensitive information (e.g., identification documents, bank statements, or the message in the email itself) is frequently exchanged by this means, protecting the privacy of email communications is a justified concern which has been emphasized in the last years.

A great deal of effort has gone into the development of tools and techniques for providing email communications with privacy and security, requirements that were not originally considered. Yet, drawbacks across several dimensions hinder the development of a global solution that would strengthen security while maintaining the standard features that we expect from email clients.

In this thesis, we present improvements to security in email communications. Relying on formal methods and cryptography, we design and assess security protocols and analysis techniques, and propose enhancements to implemented approaches for end-to-end secure email communication.

In the first part, we propose a methodical process relying on code reverse engineering, which we use to abstract the specifications of two end-to-end security protocols from a secure email solution (called p≡p); then, we apply symbolic verification techniques to analyze such protocols with respect to privacy and authentication properties. We also introduce a novel formal framework that enables a system's security analysis aimed at detecting flaws caused by possible discrepancies between the user's and the system's assessment of security. Security protocols, along with user perceptions and interaction traces, are modeled as transition systems; socio-technical security properties are defined as formulas in computation tree logic (CTL), which can then be verified by model checking. Finally, we propose a protocol that aims at securing a password-based authentication system designed to detect the leakage of a password database, from a code-corruption attack.

In the second part, the insights gained by the analysis in Part I allow us to propose both, theoretical and practical solutions for improving security and usability aspects, primarily of email communication, but from which secure messaging solutions can benefit too. The first enhancement concerns the use of password-authenticated key exchange (PAKE) protocols for entity authentication in peer-to-peer decentralized settings, as a replacement for out-of-band channels; this brings provable security to the so far empirical process, and enables the implementation of further security and usability properties (e.g., forward secrecy, secure secret retrieval). A second idea refers to the protection of weak passwords at rest and in transit, for which we propose a scheme based on the use of a one-time-password; furthermore, we consider potential approaches for improving this scheme.

The hereby presented research was conducted as part of an industrial partnership between SnT/University of Luxembourg and pEp Security S.A.

DECLARATION

I hereby declare that this dissertation is my own work and that the contents are based on my own research, except where specific reference is made to work in collaboration with others. This research has not been submitted in any previous application for a degree.

Esch-sur-Alzette, Luxembourg
September 3, 2020

Itzel Vázquez Sandoval

ACKNOWLEDGMENTS

My sincere gratitude to Gabriele Lenzini, for pushing me to find research questions and problems (bounded-)everywhere around when some directions seemed to be reaching an end, for his teachings about research itself, and for his trust, which encouraged me along the way. Thanks Gabriele!

I gratefully acknowledge too Peter Y.A. Ryan and Luca Viganò, for their valuable support and feedback during all this time.

Thanks as well to the jury members, Pascal Lafourcade, Gabriele Lenzini, Fabio Martinelli, Peter Y.A. Ryan and Luca Viganò, for agreeing and taking the time to review this work and providing much appreciated feedback.

I would also like to thank the p≡p team, for the (although not so frequent) fruitful meetings. In particular, thanks to Volker Birk for his feedback throughout my research, and to pEp Security S.A. for funding the project.

Special thanks to Arash, among others, for our peculiar and insightful discussions, his enlightening sharing of crypto knowledge, his (last-minute required) feedback on the thesis, and in summary, for his *solidarité* :P. [20.25.ch.20.1.16.19._shg^2_&∧·]

Above all, this work was motivated by those who are the source of happiness in my life, regardless their physical location. Mi más sincero cariño y agradecimiento para todos :).

Finally, thanks to all the friends and colleagues with whom I have had the pleasure to work and who have contributed to make enjoyable my walk through this segment of life.

CONTENTS

List of Acronyms

CA certificate authority
CDH Computational Diffie-Hellman
CRS common reference string
DDH Decisional Diffie-Hellman
DH Diffie-Hellman
DoS Denial-of-Service
E2EE end-to-end encryption
FS forward secrecy
HAI Human-Automation Interaction
HC Honeychecker
HISP human interactive security protocol
KC key confirmation
KDF key derivation function
LS Login Server
LWE Learning With Errors
MAC message authentication code
MITM man-in-the-middle
MSC Message Sequence Charts
MUJ Multi-layered user journey
NIZK non-interactive zero-knowledge
OOB out-of-band
OTP One-Time-Password
OTR Off-the-record Messaging
PAKE password-authenticated key exchange
p≡p Pretty Easy Privacy
PFS perfect forward secrecy
PKI public key infrastructure
PPSS password-protected secret sharing
PPT probabilistic polynomial-time
PQ post-quantum
RE reverse engineering
KR Key Register
RLWE Ring Learning With Errors
ROM random oracle model
SAS short authentication string
SMP socialist millionaires' problem
SVP Shortest Vector Problem
TOFU trust-on-first-use
TTP trusted third party
U User Interface
UX User Experience
ZK zero-knowledge

1 INTRODUCTION

Email is undoubtedly a key component in modern society. With approximately 4 billion of worldwide users[1] and 306 billion of daily exchanges [173], email is no longer used only as a means of communication; an email account acts as a digital identifier and enables access to almost every online service: online shopping, online education, financial operations, entertainment, medical appointments, etc.

Yet, the core protocols underlying email communication were specified, designed, and developed in an early age of computers and networking, when the participants involved did not have trust concerns regarding each other, and when communication links were assumed to be secure as they were owned by official institutions[2]. Therefore, privacy and security requirements were not initially considered.

As an implication, for emails received in standard settings (as it is the case with the most popular email providers, e.g. Gmail and Yahoo) there is no actual guarantee neither that the email was sent by the displayed sender, nor that the subject or email contents are as originally sent. Moreover, even if the email is received as intended, the possibility of someone reading and storing the cleartext content in transit exists.

With the extensive adoption of Internet, security risks and privacy started to be a concern. In 2013, Edward Snowden exposed evidence of mass surveillance carried out by governmental institutions on citizens worldwide [90]. These revelations motivated the development of novel security protocols and software solutions with security as a central feature, which primarily aim at ensuring the authenticity of both, the sender and the email carrier, and the confidentiality and integrity of the message, thus preventing emails from being easily eavesdropped and falsified.

Security protocols, also known as *cryptographic protocols*, are sequences of steps and interactions between entities, that rely on cryptography for providing different security properties in insecure environments. Defeating the cryptographic primitives involved in a security protocol is so expensive that usually resources are invested only to compromise the data of selected targets. However, the correct design of protocols is difficult and subject to the introduction of errors, from their conception until the way in which an implemented protocol is used; yet, providing strong guarantees about the achievement of specific security goals is essential.

For that purpose, formal approaches which provide mathematical frameworks and techniques to rigorously analyze and prove security of cryptographic primitives and protocols have been developed. In particular, an approach by which cryptographic primitives are abstracted as elementary functions in the model of a security protocol (symbolic model) has permitted the development of tools able to automatically prove or contradict whether a security protocol satisfies a certain property.

This thesis is concerned with the assurance of end-to-end secure communication principally

[1] The world population is 7.8 billions as of August 2020.

[2] Ray Tomlinson sent the first electronic mail over an academic and governmental network (ARPANET) in 1971, introducing the '@' symbol to split the username and the destination domain in an email address. The current email architecture started to be standardized in 1982 with the specification of the SMTP protocol, and its massive growth occurred along with the popularization of the Internet in the mid 90's.

over email but also via messaging. We apply formal techniques and introduce new approaches to assess and study the security of existing solutions. We also propose novel security protocols and the application of cryptographic techniques for enhancing the security of email and advancing the adoption of secure solutions.

1.1 Current Scene and Challenges

Mostly due to the innate ubiquity and interoperability features of email, secure email solutions struggle in practice to provide effective protection. As a result, email communication remains largely vulnerable to security and privacy threats [50].

Still, mature standards, protocols, techniques and tools for securing email communications at different layers already exist. In particular, interoperable end-to-end security solutions, such as S/MIME and OpenPGP, have been proved to provide effective privacy and security protection. Unfortunately, these solutions and techniques suffer from a lack of adoption and sometimes of proper implementations.

Although secure email solutions have been adopted within closed communities—e.g. enterprises, activists—a more widespread adoption is hindered principally by barriers concerning usability factors: major email providers (e.g. Gmail and Yahoo) provide neither support for nor smooth integration with end-to-end secure solutions; in addition, the underlying protocols might involve burdensome processes—e.g., key management or obtaining certificates—and might require specific knowledge—e.g., notions of public key cryptography—and IT skills not expected from the average user [120].

These reasons, along with differences regarding priorities, requirements, and technical support facilities that distinct types of users (enterprises, individuals, activist, journalists) have, make the emergence of a global end-to-end private email system improbable in the current landscape. An alternative is to implement interoperability between existing solutions, however, this solution requires interest from all the providers and is in general expensive to achieve [50].

Thus, the ongoing panorama presents open problems and challenges in several research directions, for instance: linking communities, bootstraping and automating the achievement of security goals (e.g. implementing privacy-by-default), implementing further secure email functionalities (e.g. search, messages that are deleted after a specified period of time), automating key management, protecting the content of email servers, and overcoming adoption hurdles.

Some of the previous challenges have been addressed by messaging solutions, such as the Signal application, WhatsApp and Telegram. Hinging on cryptography, modern messaging applications are able to offer advanced security features, ranging from end-to-end encryption to forward secrecy and deniability. Due to these properties, along with better usability, secure messaging has often been recommended by security experts as the go-to tool for secure communication.

However, one of the principal reasons that allow messaging solutions to achieve strong security goals is the implementation of proprietary solutions, which lack interoperability[3]. In fact, strong security goals are achieved in analogous email scenarios, i.e., proprietary solutions within restricted environments—e.g. Tutanota[4] and ProtonMail[5]. But in these cases, when nodes outside the

[3] E.g., secrecy is guaranteed among WhatsApp users, but they can only communicate with other WhatsApp users.

[4] Tutanota [178] is an open source, private-by-default, secure email service which, among others, offers easy usage, end-to-end encryption and anonymous registration.

[5] ProtonMail [148] is a claimed usable secure email service created in 2013, which offers built-in end-to-end encryption and state-of-the-art security features.

controlled environment are involved in the communication (i.e., when sending an email to a user with an account from a different provider) the offered security properties are no longer assured.

Hence, despite the success of secure messaging applications, the interoperability and openness that email offers make it the preferred (or the only) option in many cases, and also often when exchanges occur in official settings, involve longer messages, or attachments among others.

Secure email solutions rely on two approaches for establishing trust between two communicating partners: centralized and decentralized. Centralized architectures rely on trusted authorities to issue and manage certificates attesting the digital identity of users; on the contrary, in decentralized models the users are in charge of deciding which certificates to trust, primarily based on personal interaction at some point. Although the centralized approach has been the standard solution for some time, it has limitations and raises privacy concerns mainly originated from relying on a central trusted party.

A recent software solution in the decentralized setting that addresses some of the usability and interoperability issues referred to above was developed by *pEp foundation*[6] and *pEp Security SA*[7]. Pretty Easy Privacy (p≡p) is a software that implements several cryptographic standards in different digital communication channels to provide end-to-end encryption on a peer-to-peer scenario (see Section 2.8). Due to the mentioned features, we consider p≡p a tool whose study is worth considering.

1.2 Research Questions

We were interested in identifying and tackling problems hindering the advance and adoption of end-to-end secure email protocols and existing solutions, and in developing techniques that enable the formal study of their security. More specifically, in this thesis we seek to answer the following questions:

Q1: *In the context of decentralized end-to-end secure email solutions, do existing implementations provide the essential security properties that we expect from them?*

Q2: *Can the security of a system be affected by a human interacting with it? How could we better understand and study those situations?*

Q3: *Can we define security properties that have not been (or cannot be) expressed with current verification approaches?*

Q4: What are some obstacles hindering the improvement and adoption of secure email solutions and how can we solve them?

The relevance of answering these questions is spread across diverse domains.

Despite the central role of email in modern communication and the importance of securing it, most of the prevalent email clients do not provide built-in security services by default, and the ones that do, are implemented in a centralized setting, which brings a series of risks associated with trusting a single point. Furthermore, decades of research have indicated that prominent solutions that rely on decentralized settings (namely PGP) suffer from major usability shortcomings that hamper their adoption.

For these reasons, having usable end-to-end secure email solutions that work in a decentralized setting, and that can eventually find widespread adoption, is of significant importance.

[6] https://pep.foundation/
[7] https://www.pep.security/

Among state-of-the-art deployed software products, p≡p is a potential candidate due to the following reasons: it works in a decentralized setting, it preserves the original intentions of email regarding interoperability and openness, it builds upon well-established cryptographic protocols and secure email standards (OpenPGP) for providing end-to-end encryption, and more importantly, it claims to address the main longstanding usability issues that have plagued prevailing approaches.

A way to deal with **Q1** is with a formal proof of security for the cryptographic protocols underlying p≡p. Addressing this question requires working with implementations, which is by itself a challenging assignment as despite the great advances made in the last years (see Section 2.9), the formal verification of implementations is an area with fundamental open problems still being actively researched. We need then to figure out a way, and if necessary, develop techniques for carrying out a formal study of the security protocols involved in the implementations. These intermediate results would contribute to enriching the set of tools available for the formal study of security protocols, and also raise awareness about the challenges associated with such a task, as well as possible ways for overcoming them.

We also need to be aware that humans are the ultimate users of email, and current decentralized secure email solutions require them to perform actions, even if minimally, in the process of achieving security goals. However, as we have repeatedly emphasized, usability is the Achilles heel of secure email solutions.

Answering **Q2** requires us to state fundamental questions residing in an intersecting area which remains largely unexplored: socio-technical security; and come up with techniques to solve them. By being able to reason about the influence of the user on the security of a system using techniques rooted in mathematical grounds, one could identify vulnerabilities that are detected neither by a purely technical analysis, nor by a user-driven study.

The study of the security from this perspective regarding the interaction with the user might require the formulation of properties of a different nature than the cryptographic properties. This is where the relevance of answering **Q3** becomes evident. As this perspective is less explored, we need to precisely define properties that express security notions that allow to detect security vulnerabilities derived from users having a wrong perception of a system's security. Along with the expected outputs from answering Q2, this definitions would contribute to expand the formal techniques for analyzing so called socio-technical aspects, not only of email but in general of user-oriented systems.

Finally, **Q4** leaves open room for some degree of freedom, as we expect to get insights from the answers to the previous questions that influence our view of the overall situation.

We already consider that weaknesses in security originating from users could be eliminated if we were able to fully or partially relieve the user of tasks that they do not fully understand or that they find difficult to do, and instead put in place cryptographic protocols to achieve the same objective. Research on this direction promises to be an important step towards improving both, the security and the usability aspect of end-to-end secure email.

We also find quite important securing the passwords with which users access their mailbox, as they represent the key that opens the door to the user's privacy. In particular, perhaps even some readers might have found themselves in the situation of selecting a very simple password for creating an account that they thought would be a one time use, or having a common password for registering into "non-important" services. Looking into ways in which we could protect even those weak passwords is another direction that we address in this thesis.

1.3 Outline and contributions

The thesis consists of two parts and ten chapters, in which we address the previously mentioned research questions.

Chapter 2: Background Here, we provide contextual and theoretical background on topics relevant for the thesis. We review elementary cryptographic concepts. We include a brief overview of protocols for securing different aspects of email communication—e.g. the content of the message or the transport channel—and of decentralized versus PKI architectures. We also discuss factors that lead to greater advances in securing messaging than in securing email—e.g. synchronicity versus asynchronicity, interoperability of solutions, and the user's expectations. Another section is dedicated to introducing the industrial partner's software, p≡p. Finally, we provide an introduction to formal verification of secure systems from an all-inclusive perspective—e.g., verification of implementations, verified compilers, etc.

Chapter 3: A Hidden Requirement: Extracting Specifications of Security Protocols (Q1) The lack of specifications and in general of documentation for implemented security protocols is a frequent early obstacle in the analysis of deployed security protocols (e.g. [52]) which nonetheless must be sorted out in order to obtain unambiguous specifications, fundamental for any formal study of such protocols. In this chapter we present a methodical approach based on software reverse engineering, for obtaining precise specifications from open source code and scarce documentation. The technique considers the following assumptions: (i) open source code in C available, (ii) non-executable code, (iii) unknown specifications of the protocol, and (iv) unavailability or scarcity of documentation. To provide insight into the capabilities and limitations of existing reverse engineering tools, we systematize state-of-the-art tools relevant for each step of the approach. The contributions in this chapter are based on the publication [185].

After these preliminary chapters, the thesis is structured in two parts:

Part I. Formal methods for the analysis of security protocols and ceremonies This part is devoted to the analysis of security protocols and ceremonies, which extend security protocols by including the human in the models, via formal methods. More specifically, we apply known techniques and propose new ones to study protocols that aim at securing different aspects of email, from different perspectives.

Chapter 4: Formal Verification of Security Protocols An introductory chapter to symbolic formal analysis. We briefly introduce the research area, along with the formalism (applied Pi calculus) and tool (ProVerif) used in this thesis.

Chapter 5: The Honeywords System under a Code-corrupted Login Server (Q4) We begin by considering a broader problem which constitutes an independent area of research: ubiquitous access to mailboxes granted via a password. We develop upon the Honeywords system, a password-based authentication system designed to detect unauthorized logging attempts derived from a passwords database compromise. A Honeywords system consists of a login server providing the user with an interface to access the system, and a secure server which stores each password together with a set of indistinguishable decoy words, such that, an attacker stealing the passwords database and retrieving the words therein would still need to guess which of them is the password. An incorrect choice would reveal the leak. The research deals with making the system secure even in the presence of a code-corrupted login server.

This chapter is based on the publication [84] and its extended version [85].

Chapter 6: Security Analysis of p≡p Protocols (Q1) We move onto protocols that achieve private end-to-end communication. We present a formal security analysis of the p≡p protocols for key distribution and trust establishment, with respect to authentication and privacy goals and under a Dolev-Yao threat model. The abstractions that we present as Message Sequence Charts (MSC) constitute the first detailed technical documentation of such protocols and were delivered to and then validated by the industrial partner. The analysis verifies the claims of p≡p with respect to essential security guarantees and the correct assignment of privacy ratings to messages; however, we discuss limitations of the analysis and how it could be extended in several directions. We also sketch the proof of a relevant algorithm in the implementation.

The results in this chapter were published in [184].

Chapter 7: A Socio-technical Security Analysis Framework (Q2,Q3) A follow-up direction of the analysis in Chapter 6 emerges from speculating if and how the user experience affects the impression that users acquire of a system with which they interact. In particular, defining security properties that consider such user impressions allows investigating possible scenarios of misalignment between the factual security of a system and how secure the user perceives that system to be. For instance, we introduce the property "false sense of insecurity" to capture situations in which a secure system injects uncertainty in users, and its analogous "false sense of security". Both of these situations leave room for attacks. In this chapter, we propose a formal framework for reasoning about such socio-technical discrepancies. The resulting models can be automatically verified using existing model checkers. We exemplify the approach by analyzing this socio-technical notion of security within p≡p and discuss how the presented approach provides a broader understanding of a system's security.

This chapter is largely based on our results in [168].

Part II. Improvements to secure email and secure messaging The security analysis in Part I reveals potential areas for development in the p≡p protocols; we identified that secure email and secure messaging solutions would benefit as well from improvements in those areas. In this part we elaborate on ideas for relaxing the security assumptions that depend on users and propose improvements in various directions.

Chapter 8: Protecting Weak Passwords in Password-based Authentication (Q4) Here, reflecting on the reality of how users choose passwords, we propose a password-based authentication protocol, which combined with a One-Time-Password (OTP) device, aims at strengthening the security of weak passwords at login and also in storage by hampering the execution of offline dictionary attacks. Similar to the protocol introduced for code-corruption of the Honeywords System, the core idea relies on masking the passwords with a random value extracted from the OTP. We analyze further theoretical limitations of the approach derived from the entropy of the OTP value, and propose an idea to overcome such shortcomings.

The ideas discussed in this chapter are inspired by the content published in [186], however, here we have revised the main ideas and added contributions which have not been published.

Chapter 9: Automating Entity Authentication from Low-entropy Secrets (Q4) We revise the entity authentication process in decentralized end-to-end encrypted email and secure messaging, which allows users to determine whether they are exchanging messages with the intended peer. Here,

we propose a cryptographic solution based on password-authenticated key exchange (PAKE), for allowing users to authenticate each other via shared low-entropy secrets—e.g., memorable words—without the need for a public key infrastructure or a trusted third party. By minimizing the impact of human error, this approach provides stronger security guarantees than prevailing techniques that rely on out-of-band fingerprint verification, and enables additional and entirely new cryptographic functionalities in email. Moreover, PAKE establishes a base for automation and further improvements in several other aspects of secure email. The solution can be adapted for secure messaging solutions.

This chapter is largely based on the contributions published in [183] and we are currently working on an extended version.

Chapter 10: Conclusions In the last chapter we bring together overall conclusions regarding security protocols for email but also regarding the assessment of $p \equiv p$. We also discuss open problems and future directions in this area of research.

2 Background

Due to its ubiquitous deployment, securing the whole email environment requires considering problems in multiple dimensions, among them, the need for securing architectures, security in each layer of the Internet models (e.g., transport and application layer in the OSI model), usability issues, adoption obstacles and also social engineering attacks.

This thesis largely focuses on the assurance of end-to-end private communication along with the proof of association of a message with the owning entity. In this regard, email systems should essentially fulfill the three elementary security properties, all of which are required to effectively guarantee secure private communications[1]:

Confidentiality (secrecy) prevents the unauthorized acquisition of information by making the content of a message incomprehensible to everyone except to the intended recipient.

Integrity guarantees the receiver of a message from a genuine sender that such a message has not been altered in transit.

Authentication guarantees the recipient of a message that such a message comes from the claimed sender.

Cryptographic protocols are the fundamental element allowing the achievement of such properties. However, before getting into details, we provide some context for how the exchange of messages takes place in an email system, and the security of the communication channels.

2.1 Email Architecture and Communication Protocols

The basic architecture of an email system is illustrated in Figure 2.1. Email clients allow users to compose, read, send messages and manage their inbox. Mail servers manage users' mailboxes—i.e., the server's storage space allocated per user for the received emails—and deliver email messages between users.

Typically, three standard communication protocols are involved in the exchange of emails: Simple Mail Transfer Protocol (SMTP) [108], Post Office Protocol (POP) [130] and Internet Mail Access Protocol (IMAP) [58]. When an email is sent, the sender's client uses SMTP—or alternatively HTTP(S) if the client is web-based—to submit the message to the mail server of the corresponding provider. The sender's mail server locates the receiver's mail server using the domain name system (DNS), and then executes SMTP[2] to transfer the message accordingly. Finally, the client of the receiver executes POP3 (the current version) or IMAP to retrieve the message from the corresponding mail server—or HTTP(S) in the case of webmail.

SMTP, POP and IMAP are part of the application layer of the Internet Protocol Suite (TCP/IP), which includes protocols and interface methods used by hosts to exchange application data over

[1] There is a slight difference between the notions of privacy and secrecy which is rather at the conceptual level. Privacy refers to the individual's right to share specific information only with intended people, free from the observation of anyone else. Secrecy concerns the intentional concealment of information from others' simple observation or analysis.

[2] In order to address security requirements, the version of SMTP used to submit a message from client to server differs from the version of SMTP used to transfer messages between servers [83].

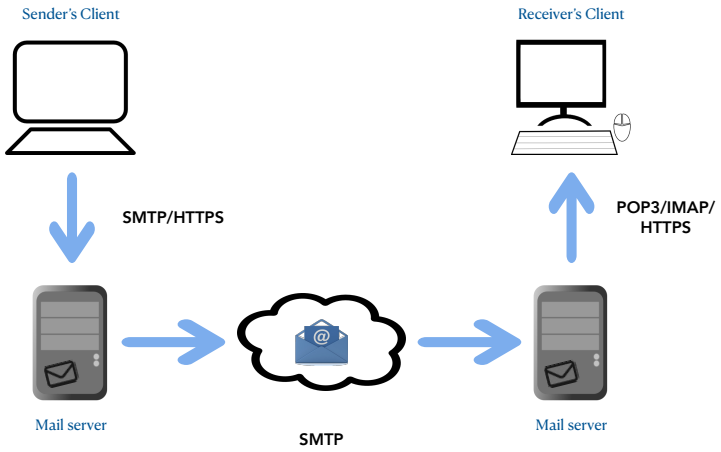

Figure 2.1: Email architecture and communication protocols simplified for a sender and a receiver.

network connections established by lower level protocols. One of such protocols is the well-known TLS, usually implemented for establishing secure channels between clients and mail servers.

Sanchez et al. [120] report a detailed security analysis of these protocols and provide technical and organizational recommendations to mitigate the privacy and security risks of email communications more effectively.

2.2 Threats and Vulnerabilities

The main vulnerabilities of email systems reside in their intrinsic lack of confidentiality, which permits emails (along with attached files) to be accessed by unauthorized third parties who can gain access to sensitive information, and in the lack of integrity and authentication, by which identity impersonation and tampering of email content are feasible.

A common impersonation attack is known as *man-in-the-middle (MITM)*. In this attack, an adversary manages to intercept and possibly alter email messages between sender and receiver while the legitimate participants still believe to be directly communicating with each other. Other well-known threats affecting different weaknesses of email include malware, spam, social engineering attacks (e.g. phishing), and massive eavesdropping.

We can also categorize attacks as per the infrastructure of email systems.

Compromise of server to server communication. The communication between mail servers is considered the most vulnerable element of the email system [120], given that it takes place through the Internet. In general, the identity of the SMTP servers (mail servers) is not mutually authenticated and the emails in transit are assumed to have originated from a genuine source.

The lack of a confidential communication channel allows a passive adversary to eavesdrop on the messages in transit. The lack of authentication allows spoofing, e.g., an adversary can deliver a prepared message to an SMTP server, pretending to be another SMTP server.

Compromise of client to server communication. In the absence of a channel implementing SSL or TLS (e.g. a public Wi-Fi hotspot), an adversary can learn the messages when they are retrieved from the server (via for instance IMAP), and furthermore learn a username and password submitted, hence fully compromising the user's account. This can be achieved for instance using a sniffer, which is a program that monitors and analyzes traffic passing over (part of) a network. A

sniffer can decode, according to the corresponding specifications, the data of the packets captured and consequently show the values of various fields in the packet.

Compromise of email data storage By compromising the mail server, an adversary gets access to all the users' emails. This attack can be executed either remotely or locally—for instance by a malicious administrator. Furthermore, if the password database of the server gets compromised, the attacker would be able to fully impersonate legitimate users, sending emails from their identities and reading all their emails.

2.3 Protocols for Securing Email

While email providers usually focus on preventing spam, filtering malware, and stopping passive eavesdropping, several cryptographic (a.k.a security) protocols and tools have been proposed to ensure the authenticity of the participants in the communication (users and mail servers), and the confidentiality and integrity of the messages.

Some of these protocols have the purpose of securing the transport of the communication by implementing secure channels to transmit the messages; this is the case of TLS. In turn, *end-to-end* security protocols focus on protecting the content of the messages themselves all the way from the sender to the receiver, regardless of the communication channel.

As stated in the Introduction, in this thesis we focus on the second category, i.e., end-to-end security protocols. Next, we review some fundamental cryptographic concepts underlying security protocols for end-to-end private communication in email and messaging. We intend this to be a brief overview, interested readers can refer to standard books on cryptography, such as [106, 127], for further details on the topic.

2.4 Cryptographic Background

Encryption schemes enable two parties to communicate confidentially over insecure channels, where an adversary can eavesdrop on the communication. As confidentiality is essential for privacy, encryption is typically at the core of security protocols achieving privacy.

While encryption schemes provide secrecy for the messages, they do not provide guarantees on the identity of the sender. Moreover, encrypted messages could be tampered by specific attacks without the honest parties noticing the modification—e.g. when malleable encryption schemes, such as ElGamal, or unauthenticated modes of encryption are used. To protect from MITM attacks, protocols that guarantee integrity and authentication must be implemented.

In modern cryptography, encryption techniques are typically classified according to the concepts of symmetric (private-key) and asymmetric (public-key) cryptography.

2.4.1 Private-key cryptography

In this setting, the sender and the receiver share a key k which is known to no one else and which they agree on before the communication that they wish to secure.

Encryption. A symmetric encryption scheme consists of three algorithms: key generation, encryption and decryption. Given a sender and a receiver, the sender uses k to encrypt a message m, also known as *plaintext*, thus obtaining c, a *ciphertext* which is transmitted to the receiver. Then, the receiver recovers the original m from the received ciphertext by decrypting c using k.

Symmetric encryption schemes satisfy the correctness requirement: $\mathtt{Dec}_k(\mathtt{Enc}_k(m)) = m$, where m is the original plaintext and $\mathtt{Enc}$ and $\mathtt{Dec}$ are respectively encryption and decryption algorithms

that take k as an input parameter. The Advanced Encryption Standard (AES)[166] is the current standard for symmetric encryption schemes.

Message Authentication Codes (MACs). Message authentication codes allow to detect if a received message has been modified in transit or did not originate from the intended party.

A MAC consists of a key generation algorithm, a tag generation algorithm $\texttt{Mac}$ and a verification algorithm. The sender computes a tag t using the message m and the shared key k, i.e., $t = \texttt{Mac}_k(m)$, and then sends (m,t) to the receiver. Upon reception, the second party runs the verification algorithm that takes as input k, m and t, to determine whether t is a valid tag of m.

Private-key cryptography has two important disadvantages. First, users have to share a different key with each peer, which demands complex key management schemes. Second, two users need to find a secure way for establishing a secret key, which turns out to be challenging for parties who have never interacted before the communication or those whose single communication channel is the one that they want to secure.

2.4.2 Public-key cryptography

To deal with the issues affecting symmetric cryptography, Diffie and Hellman [64] introduced in 1976 the notion of public key cryptography, proposing fundamental and revolutionary ideas that set the ground for moving cryptography into the public domain.

Public-key cryptography enables two parties to communicate privately without having agreed on any secret information in advance. In this setting, each user has two different mathematically related keys: a public key which can be (as the name suggests) publicly disclosed, and a private key which must remain secret and whose computation from the public key is computationally intractable.

Encryption. A sender uses the public key of the receiver pk to encrypt a message, generating the corresponding ciphertext c. The receiver then uses the own private key sk to recover the original message by decrypting c.

Asymmetric encryption schemes satisfy the correctness requirement: $\texttt{Dec}_{sk}(\texttt{Enc}_{pk}(m)) = m$, where m is the original plaintext and $\texttt{Enc}$ and $\texttt{Dec}$ are respectively encryption and decryption algorithms that take the public and private key as input respectively.

One of the most well-known and widely used asymmetric encryption schemes is the RSA cryptosystem [154], by Rivest, Shamir and Adleman.

Digital signatures. Similar to MACs, digital signatures are schemes for demonstrating that a message comes from a legitimate sender and that it was not altered while in transit, but here in addition, the sender cannot deny having sent the message.

In a signature scheme, the signer uses the owned private key sk (here called a signing key) to generate a tag—the signature—computed over a cryptographic hash of the message (we introduce hash functions later in this section). The corresponding public key pk acts as a verification key, allowing anyone who knows it to verify that a signed message was issued by the sender and that it has not been modified from the original.

Besides the primary difference regarding that digital signatures are asymmetric crypto schemes, whereas message authentication code (MAC)s are in the symmetric setting, a difference between digital signatures and MACs resides in the properties of *non-repudiation* vs *deniability*. Roughly,

digital signatures achieve non-repudiation, as the holder of a signing key cannot claim not having issued a signed message, assuming that the secret key has not been compromised. Contrariwise, in MACs all the parties sharing the secret key could have issued such signature, and thus, they only protect the integrity of the messages. Depending on the intention, one or the other scheme might be preferred; for instance, messaging solutions aim at emulating face-to-face private conversations, thus they seek for deniability, while in email usually non-repudiation is the option, for instance to certify that messages were issued by official sources.

2.4.3 Key exchange protocols

Along with public-key cryptography, Diffie and Hellman proposed secure interactive protocols that allow two parties to jointly derive a secret key k using information that can be exchanged over a public communication channel, and from which an adversary cannot benefit to derive k. These are known as key exchange protocols and enable the use of private-key schemes bootstrapped by the use of public-key ones; more specifically, the derived k can be used as a symmetric key to encrypt and authenticate subsequent communication between the two parties involved.

The security of most of the widely used key exchange protocols depends crucially on the difficulty of the underlying mathematical problem used to derive the shared key. In particular, the Diffie-Hellman key exchange is based on the conjectured difficulty of computing discrete logarithms over a finite field with a prime number of elements [64]. Note however that this protocol is susceptible to MITM attacks, since it does not consider authentication of the parties involved in the exchange.

The Diffie–Hellman protocol introduced the idea of using asymmetric techniques, along with number-theoretic hard problems, to address the key distribution problem. Indeed, closely related variants of such a protocol remain at the core of standardized key-exchange protocols resilient to MITM attacks, such as TLS.

2.4.4 Cryptographic hash functions

A hash function $\mathtt{H}$ takes as input a string s of arbitrary length and outputs a short fixed-length string $\mathtt{H}(s)$, also known as a *digest*. Cryptographic hash functions are typically expected to have three properties:

- *Collision resistance*: it is infeasible for any probabilistic polynomial-time (PPT) algorithm to find two values x and y such that $x \neq y$ and $\mathtt{H}(x) = \mathtt{H}(y)$. Informally, this property avoids the mapping of two different inputs to the same digest[3].
- *Second-preimage resistance* (weak collision resistance): given a uniformly distributed value x it is infeasible for a PPT adversary to find a value y such that $y \neq x$ and $\mathtt{H}(y) = \mathtt{H}(x)$.
- *Preimage resistance*: given a uniformly distributed value y, it is infeasible for a PPT algorithm to find a value s such that $\mathtt{H}(s) = y$. This property indicates that $\mathtt{H}$ is a one-way function, i.e., it is infeasible to be inverted.

By these definitions, collision resistant hash functions are also second preimage resistant; and hash functions that are second preimage resistant are also preimage resistant.

Hash functions are particularly important for authentication, for instance, they are used in digital signatures and in message authentication codes (MACs). Another frequent use concerns the generation of checksums for verifying integrity of data.

[3]Note that if the hash function is defined for input strings shorter than the output length (i.e., the domain is smaller than the range), collision-resistance can be trivially satisfied.

2.5 Privacy and Authentication in Secure Email

2.5.1 Securing the communication channel

Based on symmetric and asymmetric cryptography, TLS is at the core of modern communications securing the channel that transports the exchanges, independently of protocols used by applications.

In short, the client and the server first engage in a handshake protocol in order to agree on an encryption algorithm and on the symmetric key (recommended to be of at least 256 bit-long) that will be used for the rest of the exchanges in the session. The generation and the exchange of the key are typically achieved via asymmetric cryptography, with some variation of a Diffie-Hellman key exchange. The resulting session key is used for symmetric encryption of the data transmitted by one party, and for decryption of the data received at the other end. Once the session is over, the session key is discarded.

2.5.2 End-to-end security

Protocols for end-to-end security aim at maintaining the content of a message itself confidential even if the email is intercepted in transit, and to make unauthorized alterations detectable by the end points of the conversation (i.e., the users). These goals are respectively achieved by *end-to-end encryption*, in which an email message is encrypted at the sender's device before being released to the network and it can only be decrypted by the holder of the decryption key after being retrieved by the email client, and *entity* (a.k.a. identity) *authentication*, a process by which a user verifies the identity of the communication party to ensure that it corresponds to the intended peer, in order to prevent man-in-the-middle (MITM) attacks.

In the context of secure email, entity authentication is achieved via key validation (a.k.a. key verification or key authentication), which asserts that a certain public key belongs to a specific individual. A basic problem lies on how to disseminate or how to find public keys, and more importantly, how to validate the obtained keys [50].

Various approaches have been developed to tackle this problem, which follow either a centralized—e.g. certificate directories, trusted public key servers—or a decentralized trust model—e.g., web of trust, fingerprint comparisons.

A ***centralized*** public key infrastructure (PKI) relies exclusively on certificate authorities (CAs) to issue and manage certificates for keys. A public-key certificate is a digital document that contains the name of the user and the corresponding authorized public key (along with additional information about the key), signed using the key of the CA. Users can get the public keys of others directly (but not exclusively) from the central authorities and validate the certificates using the public key of the CA to check that they have a valid signature.

A main drawback of this approach is the inherent requirement for trusted entities, which turns the latter into target of attacks. For instance, a famous CA attack on DigiNotar, a Dutch certificate authority, resulted in the compromise of thousands of Gmail accounts in Iran as a result of a fraudulent wildcard certificate issued for Google on behalf of the CA [66]; furthermore, DigiNotar went bankrupt after the incident. Moreover, governments and authorities can exhaust legal means to ask server owners to hand over stored data, such as decryption keys, violating the privacy of users.

In a ***decentralized*** *(peer-to-peer)* setting, users directly verify that a public key received via a key distribution or a key exchange protocol belongs to the intended real-world entity, without the intervention of a third party. As opposed to centralized settings, in this approach, there is not a single party whose compromise would affect the security of the whole system.

An approach for implementing a decentralized PKI is the use of public ledgers, which are publicly verifiable logs of keys certificates. The fact that ledgers can be audited by anyone makes it possible to detect certificates that have been mistakenly or maliciously issued by a certificate authority (CA), exposing in consequence the malicious certificate authorities too. Although the underlying idea still considers the use of certificates and a PKI, the trustworthiness of the CAs can be determined based on publicly verifiable evidence. Examples of this approach are Certificate Transparency [113], a ledger system for web certificates based on the use of a Merkle hash tree, and ARPKI [19], which is resilient against an adversary capable of compromising a determined number of entities at any time and whose security properties have been formally verified.

Note that, regardless of the trust model, the approaches mainly rely on central repositories for storing keys, to ease their distribution among users. However, public-key servers used by decentralized approaches differ from PKIs. In a PKI, the trust relies on one or various central authorities who certify the keys, while public-key servers are just repositories aimed at supporting the dissemination of keys, but where the authenticity of the content has no guarantees and instead, is achieved by other methods.

End-to-end email security allows to effectively mitigate a wider variety of threats than protocols securing the transport of the messages (e.g. identity spoofing). However, unlike in the latter, end-to-end solutions require human intervention (even if minimally) for tasks that are not always straightforward, such as the installation of specific software, and as previously mentioned, the distribution and verification of cryptographic keys.

Those reasons are at the core of usability and scalability issues present in secure email solutions; as a result these approaches have faced limited adoption [50].

2.6 Securing Email

We consider three relevant approaches for securing email. Clark et al. elaborate on these approaches in [50], where they also provide an analysis of issues hindering further advances in securing email.

Privacy Enhanced Mail (PEM) Born in the late 1980s, PEM was the first completely architected and operational end-to-end secure email system, whose primary goals included confidentiality, data origin authentication, connectionless integrity, non-repudiation with proof of origin, and transparency to providers and to SMTP.

In particular, PEM defined file formats for storing and sending cryptographic keys and certificates, that were adopted by later solutions. This approach also identified and solved some of the difficulties still challenging secure email solutions, such as interoperability, hierarchical trust, certificate's validation and revocation, and anonymity.

Arguably, due to its slow evolution for Multipurpose Internet Mail Extensions (MIME) and to the dependence on a hierarchical public key infrastructure (PKI) with a single root CA, which was never deployed, PEM did not enjoy wide adoption.

Secure/Multipurpose Internet Mail Extensions (S/MIME) S/MIME [177] is a standard for public key encryption and signing of MIME data. Unlike in PEM, the trust model of S/MIME is based on certificates generally issued and managed (e.g. revoked) by independent third-party CAs (certified by centralized root CAs), therefore, internal communication of institutions fits in with this standard, as typically each organization acts as a trusted root for its own certificates.

Commercial solutions such as Microsoft Outlook and IBM Notes have an embedded implementation of the standard, which improves usability; these solutions are mostly adopted by enterprises and governments who usually have support teams to deal with configuration tasks and users assistance.

A drawback of this approach is the difficulty for users to decide which CAs to trust, and interoperability between key management servers. S/MIME has also limited adoption by individual users as they must execute all the key management task themselves (create own key pairs and certificates, search for the correct public key of a recipient, etc).

Pretty Good Privacy (PGP) OpenPGP [89] is arguably the most widely used email encryption standard. Derived from the PGP software, released in 1991 by Phil Zimmerman [196], the standard proposes the use of symmetric and asymmetric cryptography plus data compression to encrypt communication, and digital signatures for message authentication and integrity. OpenPGP includes also methods to protect data at rest, such as passwords or backups. Open source implementations include GnuPG (a.k.a GPG) [172] and Sequoia-PGP [189], the latter currently under development.

In this case, the trust model is implemented via a *web of trust*. The rough idea is that each user creates and publishes their own public key to public key servers; the authenticity of these keys is asserted by other individuals who have validated them in a trustworthy way. A popular way for doing this is at so called key signing parties; for this event, the public keys (and the associated fingerprints) of all the participants are distributed to them in advance, so that during the event, each fingerprint in the list is presented to the audience who can verify that it is correct. Then, the owner of each key presents an adequate proof of identity to certify the ownership. After the party, the assistants digitally sign the public keys that they have trusted as a result of the previous process, thus forming a chain of certifications [196].

Note that although in the web of trust model there is no dependence on trusted authorities (anyone can certify others' keys), the discovery and construction of certificate chains is supported by public key servers, to store certificates and respond to key searches.

As in the case of the previous approaches, severe usability drawbacks have been identified in PGP (e.g. [191]), principally, the need for users to understand at least general cryptographic concepts regarding public key cryptography—which inevitably narrows down the scope of the audience—and the need for verifying the ownership of public keys. Furthermore, the key authentication process described above clearly has scalability issues, although it is a good non-commercial solution for small closed groups.

2.7 Secure Email vs Secure Messaging

Despite the constant efforts put in securing email, secure email solutions have not reached the degree of maturity and adoption that secure messaging solutions present. Because of that, using the latter has been recommended by security experts and activists when privacy in the communication is relevant[4]. However, email and messaging were conceived for different purposes and although nowadays there is an overlap in functionalities, user's expectations from each of them differ.

We discuss some of the differences next.

Synchronous vs asynchronous communication. Initially, instant messaging solutions required both communicating parties to be online for the messages to be delivered (e.g. AOL Instant Messenger); due to this requirement, the communication was synchronous—i.e., in real-time or "at the same

[4] E.g. cites in `https://signal.org/`

time". Messaging solutions have considerably evolved in such a way that current applications permit the delivery of messages to users that are offline, thus emulating the asynchronicity of email.

The main functionalities provided by messaging solutions are inspired by face-to-face conversations; for instance we can see a "typing" status when a peer is replying (somehow emulating the presence of the peer), and users can immediately see the messages received (as opposed to the need of opening an email to have access to the content). In addition, users can hold several simultaneous conversations.

The use of email fits asynchronous communication, where users do not expect an immediate answer or an answer at all. Typical uses of email include formal conversations, sending long texts and file sharing. Among the expected functionalities are easy archiving and organization of messages and files, and search over previous messages and attachments.

Email is also preferred for ubiquitous communication, while in messaging, the exchanges tend to be among people that have met each other (although not necessarily). The interoperability of email is another differentiating factor; as opposed to messaging, where both parties must be registered to the same service, email is an open system, where an email address is all that is required to send a message to anyone regardless of the provider.

Security and usability Two protocols are at the core of end-to-end encryption. While PGP was designed for asynchronous, high-latency email communications, Off-the-record Messaging (OTR) [42], a protocol originally published in 2004 for providing end-to-end encryption, deniability and authentication, was designed for low-latency messaging environments (we elaborate on the OTR protocol in Chapter 9). Both protocols have also different security goals, but while security in both contexts reaches comparable levels, usability is a determining factor that has favored messaging solutions.

In this regard, secure messaging and email share two long-standing challenges, namely entity authentication and key management (e.g., see [157]). Mainly due to the use of closed architectures, messaging solutions have managed to automate key management tasks. Presumably, this is the main reason behind the successful adoption of such solutions, since the automation allows security to be embedded by default, unnoticeable to users, who are not required to perform tasks in order to have encryption, and require minimal intervention for entity authentication. As we saw in previous sections, this is not the case for secure email solutions.

2.8 Pretty Easy Privacy

As previously mentioned, we conducted this research in the context of a partnership with pEp Security SA. In consequence, throughout this thesis we consider p≡p, their software solution, as a case study.

Pretty Easy Privacy (p≡p) is a software solution that offers end-to-end privacy-by-default in digital written communications (e.g. email clients). The software is intended for a general audience, therefore, it was designed considering functionality and usability features, such as interoperability, minimal configuration required, accessibility, and simple usage.

A valuable aspect of p≡p is the focus on usable security; the solution builds upon well-established cryptographic protocols and secure email standards to provide confidentiality via end-to-end encryption, and largely automates tasks that would require specific knowledge from non-expert users, e.g., key management. As such, the key contributions of p≡p include:

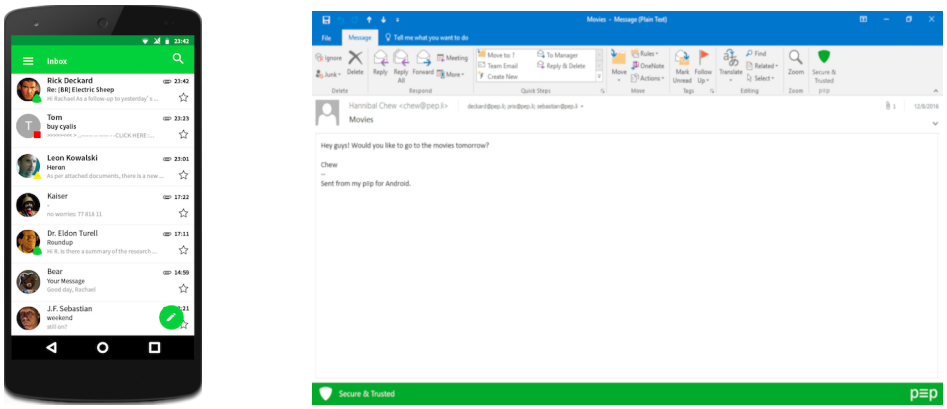

Figure 2.2: p≡p for Android and Outlook (from https://www.pep.security).

- Automatic generation of a user's key pair, key management and key discovery of user and contact's keys.
- A function to calculate so called *trustwords* (words in a natural language) from fingerprints, intended to ease out-of-band authentication.
- An algorithm to determine the highest privacy rating that can be assigned to a message from/for a specific partner, and the corresponding interface to inform the user about such a rating.
- Whenever possible, automatic end-to-end encryption of messages (*opportunistic encryption*).
- A protocol for the synchronization of keys among multiple devices of the same user, to make encryption and decryption of messages possible in all of them (still under development).

2.8.1 Architecture of p≡p

The architecture of p≡p consists of three layers: *Engine*, *Adapter* and *Application* (Figure 2.3). The Engine layer implements the logic and core functionalities of the system; the core component is pEpEngine, where cryptographic functionalities are implemented upon existing standards for secure end-to-end encrypted communications (e.g., OpenPGP, S/MIME) and well-known implementations of such standards (e.g., GnuPG); pEpEngine is written in C99 and is released as an open source library.

Adapters for interoperability of pEpEngine with different programming languages (e.g. Python and ObjC) and the corresponding user interfaces, are implemented in the Adapter and Application layers respectively.

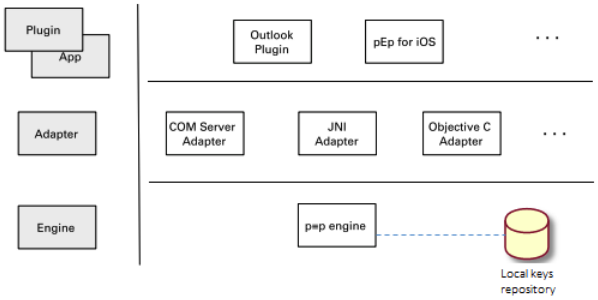

Figure 2.3: Overview of p≡p's architecture.

Distribution specifications. Each installation of p≡p creates a local database of contacts, to manage cryptographic keys and privacy ratings. Additionally, it creates a database from which the trustwords for mutual authentication are retrieved; the trustwords database contains the exact same data in all the distributions.

For key distribution and storage, p≡p relies on a fully decentralized model; private and public keys are generated locally, and securely stored in the devices relying on GnuPG[5]. No p≡p central servers are needed for the distribution of messages either, since p≡p acts as a service on top of email services, and thus, the email exchanges are managed by the email providers. A detailed description of p≡p can be found in [32].

The p≡p software is distributed as a mobile application or as plugins for desktop installations of some existing email clients, such as Outlook and Enigmail. Although p≡p has already been released for several platforms and the main functionalities of the product are implemented, the software is still under active development. Current developments include coverage for other platforms (e.g. iOS, macOS, browser add-ons), and synchronization between devices.

2.8.2 Privacy-by-default

The development of p≡p is in agreement with two principles introduced as regulations in the General Data Protection Regulation (GDPR) [75].

Privacy-by-design: privacy must be approached from the earliest stages (design) and throughout the complete development life-cycle of a service or product—not just in reaction to breaches.

Privacy-by-default: by default, personal data should be processed with the highest privacy protection settings—regarding the amount of data collected, the scope of their processing, the period of their storage and their accessibility—so that it becomes the user's choice what information to make accessible for others.

In this case, privacy-by-default is achieved via trust-on-first-use, opportunistic encryption and out-of-band (OOB) authentication: upon installation, p≡p generates a public-private key pair and automatically stores and manages contacts. The public key of a user is attached to outgoing emails when a key of the recipient has not been stored. Received keys are automatically stored for future use (*trust-on-first-use*) and outgoing emails are automatically encrypted when a public key of the intended receiver is available (*opportunistic encryption*).

For the entity authentication process, p≡p introduces the so called *trustwords*, which are natural language words that users have to compare via an out-of-band channel—e.g., in person, video-call. This approach requires neither a PKI nor a trusted third party (TTP).

2.8.3 p≡p Trustwords

Keys in p≡p are identified by the full fingerprint of the public key. Manual key-fingerprint comparison is a well-established method for entity authentication; yet, the approach has been shown to perform poorly for the intended goal (e.g., [60]). As a solution, in addition to the hexadecimal representation, PGP allows fingerprints to appear as a series of so-called "biometric words", which are phonetically different English aimed at simplifying the comparison for humans and to make it less prone to misunderstandings [102].

Trustwords in p≡p follow the same idea; they are natural language words mapping hexadecimal strings intended to allow two peers to authenticate each other after having exchanged public keys

[5] https://www.gnupg.org/

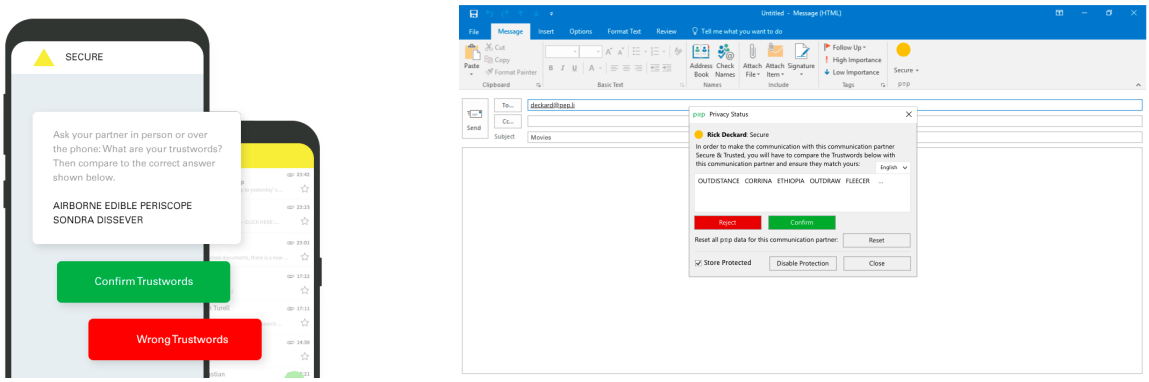

Figure 2.4: Trustwords (from https://www.pep.security).

in an opportunistic manner. In short, such hexadecimal strings represent a combined fingerprint obtained by applying an XOR operation to the fingerprints associated to the public keys being authenticated. Each block of 4 hex characters of the combined fingerprint is mapped to a word in a predefined trustwords dictionary. For instance, the extract `F482 E952 2F48 618B 31DC 5428` could be mapped to `kite temporal brother juice situation broken`.

The main differences with the "biometric words" are the availability of trustwords in different languages, which improves the security for non-English speakers, and the use of longer words, which allows the creation of a larger trustwords dictionary in which the likelihood of phonetic collisions among the elements is decreased. Considerations regarding the number of words in the dictionaries and the length of the words themselves are further discussed in [31].

2.8.4 Trust Rating and Visual Indicators

According to the security of the communication channel, the cryptographic settings available and some criteria regarding the key, p≡p assigns a numeric privacy (or trust) rating per message. To inform users of this rating, the assigned number is first mapped into one of four user-friendly categories, and then displayed in the message along with a colored icon which mimics the semantics of a traffic light [124]. The user ratings are described in Table 2.1.

General design choices and principles of p≡p are detailed in [32].

2.9 Formal Verification of Secure Systems

Security and correctness of systems are typically assured by meticulous functional testing and by the application of good development practices. However, although these pragmatic approaches ensure the correct behavior of systems, they lack mathematical foundations and a formalism for describing processes, expressing properties and proving theorems that concern the security of such systems.

As security protocols are at the core of most of the current distributed communication systems, to achieve secure communications and secure information processing, proofs of security are fundamental in all the stages of the development process (design, implementation, compilation, etc.) to ensure specific guarantees, or to detect flaws in a protocol.

In fact, implementations of security protocols that aim to comply with certification standards for Information Technology Security[6] must be built upon a formally verified design to achieve the highest

[6] E.g., the "Evaluation Criteria for Information Technology Security" (ISO/IEC 15408-1:2009). `https://www.iso.org/standard/50341.html`

Rating	Description	Message format
Mistrusted 	The system has evidence of the communication partner not being the expected person. E.g., when the user explicitly mistrusts a peer.	Plaintext
Unknown/ Unsecure/ Unreliable (Unsecure)	Encryption/decryption of a message cannot be properly executed. E.g., when the recipient does not use any secure email solution, or does not have cryptographic keys (unsecure), or the message has no recipient yet, hence, the privacy level cannot be determined (unknown).	Plaintext
Secure 	The user has a valid public key for the recipient, however the key has not been authenticated.	Encrypted and signed
Trusted 	The user has the public key of the recipient and both peers have authenticated their respective keys.	Encrypted and signed

Table 2.1: p≡p trust ratings

certification levels (EAL7, EAL7+) [81], which however, in practice are only awarded to applications controlling high-risk situations.

A proof of security for the design of a protocol guarantees that the theoretical model grants the proven properties in the presence of a specific adversary; however, flaws may still be introduced when the protocol is implemented or furthermore, at compilation time. Therefore, ideally end-user applications should be formally verified, but security proofs of implementations are very challenging.

Two main approaches prevail for attaining a verified implementation; in both, we start from an informal description of the protocol and then, either

a) build a formal model in a language FL accepted by a verifier; then, iteratively verify and refine the model, assisted by automatic tools. When the model satisfies the required properties, generate the corresponding source code in a programming language P; or
b) program the protocol directly in P and then, from the existing source code generate a formal model in FL, based on predefined rules. FL can be used as input for a verifier.

The first approach is the ideal as per good practices in software development, because errors are detected in early stages. Such an approach can be implemented for instance using F*, a relatively recent programming language aimed at program verification based on dependent types, which uses an SMT (Satisfiability Modulo Theories) solver as a reasoning engine. Code in other languages, such as OCaml, C and WASM can be extracted once a program has been successfully verified; this allows to have running applications with verified security. F* is for instance being used for building a verified replacement for the whole HTTPS stack in the Project Everest [152].

The second approach is suitable for already existing implementations; relevant work on this approach includes [8], [6] and [187].

The use of verified compilers complements the formal verification at the source code level (e.g. by program proof or model checking). As the term suggests, a verified compiler has been proved to be exempt from miscompilation issues. The corresponding proof of correctness guarantees that

the produced executable code behaves as specified by the semantics of the source code, hence, properties formally verified on the source code automatically hold for the compiled executable. A relevant verified compiler for the C programming language is CompCert [115], which is largely programmed in Coq.

In this thesis, we focus on the verification of theoretical models, i.e., designs of protocols. The main reason behind is that, by considering the study of p≡p's design instead of the implementation, we can apply techniques from a mature area of research, namely symbolic verification (for which provide a more detailed introduction in Chapter 4), while being able to focus in exploring further problems related to securing email communications.

3 A Hidden Requirement: Extracting Specifications of Security Protocols

Implementation of cryptographic algorithms has for long time been known to be a delicate and complex task. As the security of systems highly depends on such components, a recommended practice is to make use of cryptographic libraries that have been proved secure (e.g. [147, 197]) or those whose continued successful use serves as evidence of their robustness (e.g. [28]).

Nowadays, we can find cryptographic libraries for many popular languages such as Java, Phython and C (see [192]). Implementations in the latter prevail largely due to the efficient runtime performance of programs in C—given that C is a relatively low-level programming language with simple data structures, the translation of source into machine code is quite straightforward—which is a valuable feature considering that cryptographic functions involve mathematical operations inherently expensive for the CPU [72].

Yet, the subset of existing cryptographic libraries that have been proven secure via mathematical-based techniques is very limited; hence, more than increasing the number of new implementations, we believe important to prove secure those already in use.

To analyze cryptographic protocols implemented in libraries we require a clear definition of the protocol's logic, along with the parties involved, the messages exchanged and the cryptographic primitives used. Ideally, software producers should offer insights on their solutions; however, it is rather common to encounter that documentation is missing, outdated or even restricted—e.g. for proprietary solutions as in the case of WhatsApp.

Non-adherence to good development practices or the adherence to specific software development methodologies are some of the causes behind the lack of documentation. In *agile methods*, for instance, one of the principles states: "Working software over comprehensive documentation" [23]. The documents to be generated (if any) are usually determined by the needs and experience of each development team; in fact, even commented code could be considered as documentation.

Therefore, to understand a piece of source code, analysts often resort to software reverse engineering (RE)[1], a technique for scrutinizing a system with the purpose of representing it at higher levels of abstraction. The characteristics of the software to be reversed limit the approaches and tools that can be applied, and the kind of diagrams that can be retrieved.

Today, there are stable and good-performing tools for automating the creation of diagrams—mostly compliant with the UML/ISO standard for modeling systems—from code. Motivated by the need to understand how a system works, such RE tools usually take as input a running application. However, this input does not always exist. As previously discussed, security protocols are often encapsulated in libraries without executable files.

[1] Remark that in the context of security protocols' analysis, RE frequently refers to reversing executable binaries or programs to source code, for instance through system-monitoring tools, disassemblers, debuggers and decompilers. Here, we refer to RE at the code level, i.e., reversing source code to a conceptual model.

When we started studying p≡p, there was relevant technical documentation, neither of the protocols implemented nor of the source code. User manuals and a white paper [79] were available, but technical documentation consisted mainly on comments in the code. Still, such specifications were essential for studying the protocols presented on this thesis.

The content of this chapter results from our experience towards defining an abstraction of the security protocols implemented in p≡p. We present a methodical approach that combines the use of existing RE tools with manual tasks, for obtaining a concrete high-level definition of a security protocol whose core algorithms are services provided by a library. As the steps are independent of the implementation, this approach is generic enough to be applied to any C library.

The technique that we propose is based on software reverse engineering and undertakes the following assumptions: **(i)** open source code in C available, **(ii)** unknown specifications of the protocol, **(iii)** unavailability or scarcity of documentation, and **(iv)** inaccessibility to any application requesting the library's services to fully set up the protocol.

We also intend this chapter to bring insights on the state-of-the-art, capabilities, and limitations of existent RE tools. As a result of surveying, selecting, and experimenting with tools that could be appropriate for our specific problem, in Section 3.2 we compile a list of state-of-the-art tools relevant for each step of the approach.

3.1 Retrieving a Protocol from a Library in C

We started by searching for an automatic tool fulfilling all of the following requirements: performing RE of code without an executable file, supporting the language C, and being capable to generate at least one kind of interaction diagram (e.g., sequence, message). Our quest included internet searches, IT blogs and forums, academic papers, and informal consultations with experienced software developers and architects working in industry. We considered both, commercial and non-commercial tools, although for the last ones we only tested the demo versions, which in general provide full features for a limited period of time.

Despite the search scope, we did not find a tool meeting all of the constraints, thus, we devised a streamline process where we combine different tools (see Table 3.1) to increasingly extract and gather pieces of information until there is enough knowledge for abstracting a model of the protocols. The process has five steps.

Step (1): Extract the API specification of the library
Input: Compiled Libraries **Output**: The APIs

Libraries offer functionalities through well-defined application programming interfaces(APIs), which are the access point to the services provided. This encapsulation of code eases, among others, modularity, portability, and extension of functionality.

The first step to get an overview of the services that a library provides—and possibly a hunch of what they do according to their names—is to export the API specification from the compiled binary. A couple of tools for performing this step are DLL EXPORT VIEWER [165] for *dll* files or the regular export option for *jar* files. An MS-DOS command-line option is `dumpbin <dll_file>`.

Once we learn the (names of the) functionalities that can be invoked, the next goal is to understand what exactly they do, what are their constituent steps and how the protocol's components interact when calling them. *Interaction diagrams* are particularly suitable to capture this behavior as they show sequences of messages sent among components of a system (e.g., files, hardware), including

passed parameter values and events occurring in between [70]. Unfortunately, since a library does not run by itself, there is no easy way to gather this information.

An intuitive solution would be to implement a simple application with the purpose of invoking functions from the library; the resulting executable file could then serve as input for existing RE tools to automatically generate the corresponding diagrams. However, the problem becomes circular: having no documentation, we do not know how to invoke the functions and, moreover, we do not know yet what they do.

Note also that, although in general open source code compilation is feasible, this task is not necessarily easy; take for instance projects that depend on several third-party libraries, require hierarchical compilations, or need additional tasks that prevent to automate the process.

The next steps aim at understanding how the library works.

Step (2): Create an overview of the system architecture
Input: Library's source code **Output**: graphs and diagrams

This step's goal is to create a global picture of the library's components and of the way in which they are related. *Dependency graphs* and *collaboration diagrams* accomplish this purpose. A convenient tool capable of automatically generating them is DOXYGEN [96], the standard tool for creating documentation by annotating C++ sources, but it supports more programming languages, C among them. The documentation is by default generated in HTML.

A similar tool is OOVAIDE [135], capable of generating classes and sequence diagrams. This tool, however, was conceived for the Object Oriented approach and thus the functionality with C is incomplete. Auxiliary documents that can be created are UML *class diagrams* or the analogous *file diagrams* for C code [70]; they depict the relation among .c and .h files composing the implementation. *Call graphs* provide another useful visualization of the dependency among functions; in call graphs the nodes represent functions and incoming directed edges indicate that the function in such node is called by the function on the opposite side of the edge (see Fig. 3.1).

Using the output from this step and keywords in the API functions' names (step 1), the analyst can identify the files and functions that contain code related to the logic of the protocol to be reversed, and discard code irrelevant for the protocol's description (e.g., files implementing system's IO operations or interaction with databases).

Step (3): Retrieve the algorithms of relevant functions
Input: Library's source code **Output**: Flowcharts

Call graphs from the previous step provide already an idea of a function's intricacy in terms of the number of sub-functions invoked, but the order in which those sub-functions are called is missing. The goal here is to retrieve the algorithm that an identified relevant function follows. CODE VISUAL TO FLOWCHART (CVF) [77] converts procedural code into flowcharts and works with both, executable and non-executable declarations. The flowchart provides an easier way to follow the workflow of an algorithm, but since all the instructions are mapped into the diagram, it still needs to be refined by extracting only those instructions directly concerning the main purpose of the algorithm. C programs in particular contain a lot of low-level instructions, for instance to handle memory and pointers; code related to exceptions handling is also irrelevant in the normal flow of a protocol. After performing this step, the analyst has a flowchart with instructions related only to the protocol's logic or containing important assignments (Fig. 3.2).

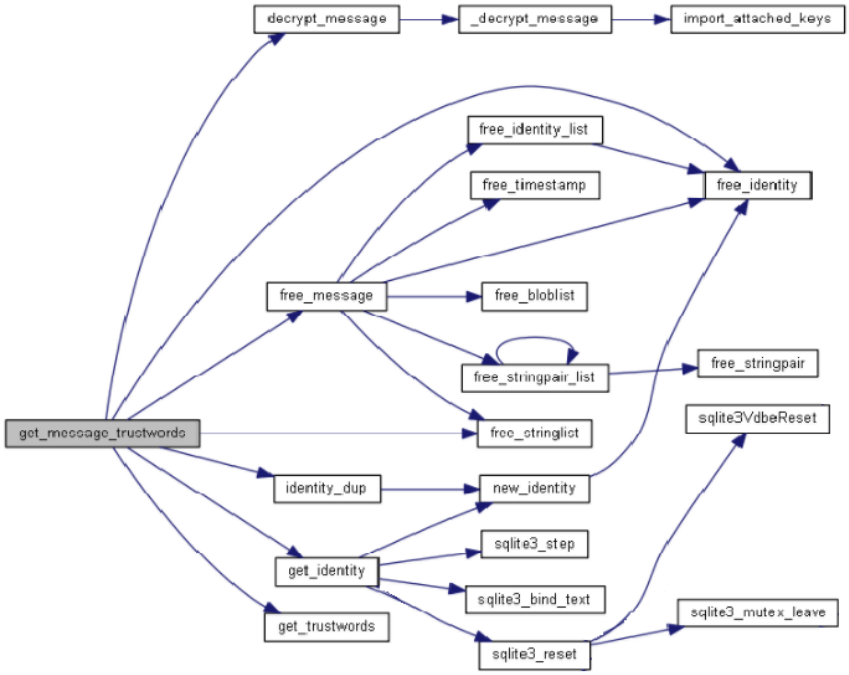

Figure 3.1: Call graph obtained by DOXYGEN for the function get_message_trustwords.

Step (4): Identify relevant calls, and group instructions
Input: Flowchart/Code **Output**: Flowchart/Code

At the end of step (3) we have a sketch of the algorithm performed by a function but it is still too detailed. The objective in this step is to abstract the instructions into a higher-level conceptual model; the representation is still a flowchart/code but instead of direct programming instructions, the content will be in terms of tasks. For that purpose, we introduce two rules which are similar to the *extract* and *inline* refactoring methods, used principally to reorganize code for better reuse and readability [80]:

1. Detect instructions linked to a single more general task and replace them all with a newly defined method with a meaningful name related to the purpose of the task.
2. For each function call, review the implementation details and if they are relevant to define the protocol, mark the function (e.g. with '*') to be treated later.

These rules might be applied recursively on the functions marked in 2. It is the analyst's duty to determine when an adequate level of abstraction has been reached. An example of the application of these rules is shown in Figure 3.3.

Step 4 requires navigation through the files and the code itself, thus, any IDE would be in principle enough. We worked with VISUAL STUDIO 2015 [128]; a feature worth mentioning of this tool is the "call hierarchy" view of a function, since observing the references to and from a function helps to determine whether it has a principal or an auxiliary role.

Step (5): Create a diagram of the Protocol
Input: Flowcharts **Output**: Message Sequence Chart

At this point, the analyst has a deep understanding of the examined functions. The last step consists in interpreting the previously obtained flowcharts to reconstruct a representation of the protocol in an appropriate notation. There are many accepted notations; here we adopted Message

```
DYNAMIC_API ...

  get_message_trustwords(...){
     // verify that the variables exist
     assert(keylist);
     assert(msg);
     assert(received_by);

     if (!(keylist && msg && received_by))
        return ILLEGAL_VALUE;

     if (keylist == NULL) {
        message *dst = NULL;
        stringlist t * keylist
          ...
     }
     ...
  }
```

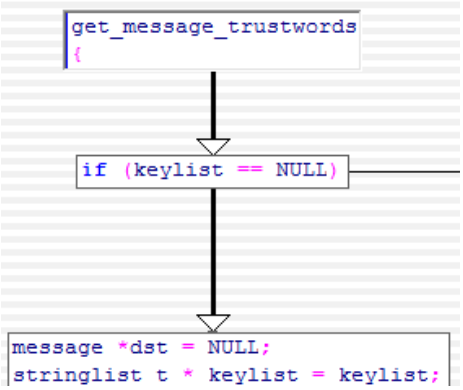

Figure 3.2: Left: extract of the function get_message_trustwords for retrieving words to authenticate a peer. Right: flowchart after removal of instructions related to parameters checking.

Before step (4)

```
char *source1 = id1->fpr;
char *source2 = id2->fpr;

int source1_len = strlen(source1);
int source2_len = strlen(source2);
int max_len;

*words = NULL;
*wsize = 0;

max_len = (source1_len > source2_len?source1_len:source2_len);

char* XORed_fpr = (char*)(calloc(1,max_len + 1));
*(XORed_fpr + max_len) = '\0';
char* result_curr = XORed_fpr + max_len - 1;
char* source1_curr = source1 + source1_len - 1;
char* source2_curr = source2 + source2_len - 1;

// create a hex string by xoring char by char the given fingerprints
while (source1 <= source1_curr && source2 <= source2_curr) {
  ...
  *result_curr = xor_hex;
  result_curr--; source1_curr--; source2_curr--;
}
...
// retrieve a list of words corresponding to the string XORed_fpr
status = trustwords(session, XORed_fpr, lang, &the_words);
```

After step (4)

```
XORed_fpr = createSharedFingerprint(fpr_id1, fpr_id2)
status = trustwords(session, XORed_fpr, lang, &the_words);
```

Figure 3.3: By rule 1, we create the function createSharedFingerprint to abstract the block of code, excluding the last instruction. By 2, we leave the call to trustwords as a function.

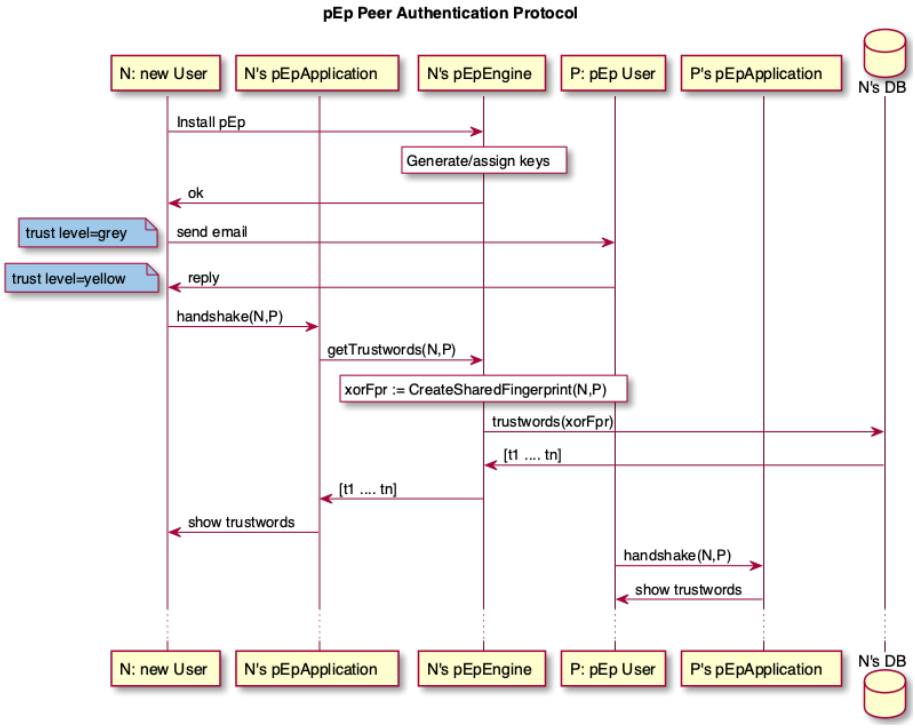

Figure 3.4: Extract of the MSC representation of the abstracted protocol, created with PlantUML.

Sequence Charts (MSC), defined in SDL[2] which is a language for unambiguously describe the behavior of distributed systems.

This step is completely manual since it requires the analyst not only to connect information from the flowcharts, but also to use any domain knowledge to come up with the complete scenario of the protocols, including the actors involved. The participants can be for instance derived from the input parameters of the functions; similarly, calls to databases imply the existence of a repository, which might need to be considered as an actor in the protocol model. The analyst's domain knowledge can come from different sources: a general notion about the intention of the protocol can be learned in documents regarding final products that make use of the library; often implementations include test files, which can provide examples of functions calls; user manuals can also contain hints about the components of the protocol and their interaction.

The offer of modeling tools for MSC is vast, however, we consider more efficient the use of tools that automatically generate diagrams from scripts, given that the scripting languages' syntax tend to be simple and intuitive for programmers. A good performing tool in such category is PLANTUML [141], which supports all kinds of UML and some non-UML diagrams, and provides an online application as well as plugins to be integrated with many different IDEs. ZENUML [138] is another tool in this category; it has a syntax closer to C code an can be used online or as a Chrome extension, but it is very limited in terms of the diagram that it produces as well as in the conversion to image formats.

3.2 Evaluation and Discussion

The following outcomes report our experience applying the reverse engineering method presented in Section 3.1, for creating MSC diagrams corresponding to the protocols implemented in the core

[2]The Specification and Description Language (SDL): `http://www.sdl-forum.org/SDL/`.

Table 3.1: Summary of RE tools reviewed in our methodology. (*)These tools do output sequence diagrams from non-executable code but only from Object Oriented (OO) languages. Although they do not work with our requirements, we include them as a reference.

Step	Tool	Availability	Supported Languages	Output Diagrams	Output Format	O.S.
1	DLL Export Viewer	Free	dll	N/A	File Explorer	Windows
2	Doxygen	Free	C++, C, C#, PHP, Java, Python, ...	Call graphs, Dependency graphs, Inheritance diagrams	HTML, Latex	Windows, Linux, MacOS
	Oovaide	Free	C++, CLang related, Java	Zone, Class, Portion diagrams	SVG	Linux, Windows
2,3,4	Rational Rhapsody for Architect (RRA)	Free: 1 month, full functionality / Commercial	C, C++, Java, C#	Object model, Flowcharts	Several image formats	Linux, Windows
	Enterprise Architect	Free: 1 month, full functionality / Commercial	Ada, C, C++, Java, PHP, Python, ...	Several structural and behavioural diagrams	HTML, Several image formats	Windows, Mac/Linux (WINE)
3,4	Code Visual to Flowchart	Free for C, C++, C#: limited functionality/ Commercial	C, C++, Java, PHP, PL/SQL, T-SQL, Perl, JavaScript ...	Flowcharts	PNG, only visual in the free version	Windows
*	Visual Paradigm Enterprise	Free: 1 month, full functionality / Commercial	Java, C++	Class, Sequence, Communication, Component, Package, Object, Interaction Overview diagrams	Several image formats	Windows, Linux, MacOS
	Architexa	Free / Prototype for C on request	Java, Prototype for C++, C	Layered, class and sequence diagrams	PNG, own format	MacOS free. Plugin for Eclipse IDE

component of p≡p, namely pEpEngine (introduced in Section 2.8.1), which provides the main functionalities upon which the p≡p protocols are built—e.g., key management, encryption and trust assignment.

The process was entirely executed by the author of this thesis, who has approximately 4 years of professional experience in software design and development. Steps (1) and (2) can be executed relatively fast: one day was enough to generate the documents, scan them and identify the relevant functions, using a DLL file provided by our industrial partner. Step (3) requires the analyst to be careful not to discard meaningful information; 1 day per function was invested in this task. Step (4) is the most time-consuming because here is where the real understanding of a program occurs; the analyst has to decipher the control flow statements—such as conditions and loops—and try to reconstruct the logic behind. Reconstructing the algorithm of a function containing around 500 lines of code took up to two weeks of work, having no previous experience using the tools—but we believe that this time can be considerably shortened with practice and familiarity with the tools. The final transition from step (4) to step (5) was achieved in one week. Certainly, these times are relative and depend significantly on the complexity of the system; small files can contain very complex logic, requiring hence longer time to be understood. The time we have reported for the work is only meant to give a relative estimation to whoever wants to contemplate the possibility of applying this technique in a project.

An appropriate selection of the tool to be used at each step is important, as feeling more confident or understanding better the output of a specific tool could improve the analyst's performance. We included two well-known commercial tools in this research: RATIONAL RHAPSODY FOR ARCHITECT (RRA) [98] and ENTERPRISE ARCHITECT [169]. In general, with C libraries they do a good job for class diagrams and flowcharts; RRA's flowcharts are slightly the most comprehensive from all the tools. For sequence diagrams though, both commercial tools rely on exploring execution traces from running applications, thus, they are useless for automation in our case.

According to our experience, this method is suitable to be applied in small and medium size projects since manual parts would become exhausting with very lengthy code. We consider pEpEngine to be medium-big size given that it consists of approximately 70 files, from which the longest has around 2524 lines of code while the shortest contains around 40.

The proposed technique is straightforward and suggests the use of automatic tools to perform the main tasks; still, it unavoidably requires human interaction to connect the steps. Nonetheless, the approach presents clear advantages over purely manual RE: at the end of each step there is already a document concerning the source code which, in a situation where documentation is still missing, supplies the analyst with means to promote discussion and knowledge exchange with the development team; those intermediate diagrams also help analysts to contextualize faster their mental process after periods of interrupted analysis (in case that they occur). Another important advantage is that, even before starting the code inspection, the analyst has an organized and complete overview of the library's structure and of the relation among functions; this is beneficial to address the RE efforts in the adequate direction. Moreover, as a result of the code inspection analysts become more aware of the vulnerable points of the system and might detect security properties to be considered in the analysis.

3.3 Related Tools and Approaches

The most general approach to automatically reconstruct protocol specifications aims at describing the protocol from an application perspective and consists in observing the code at runtime to record sequences of the executed actions. A survey by Tiwari and Prasad [174] reports on tools following this approach. A close problem to ours is treated by Tonella et al. [175], who work on reverse engineering code in C++; however, their algorithm relies on the number of objects instances and so it is not directly applicable in our case, which is not object oriented.

A related technique to infer application-level protocol specifications relies on network traffic observation. It concerns network protocols (e.g., HTTP, RPC and CIFS/SMB) implemented in any layer of the OSI model[3]. The central idea is to automatically reverse engineer the protocol message formats of an application from its network trace. This method is especially useful when a file to execute the protocol is unavailable because it only requires a system implementing the protocol to be running, even if such a system is not running locally. Narayan et al. [132] worked on a survey of automatic protocol RE tools using this approach. Those tools seem to be mostly academic; a commercial one is VISUALETHER PROTOCOL ANALYZER [76], which uses the output of Wireshark—a network protocol analyzer—to generate sequence and call-flow diagrams.

On the research side, Avalle et al. [11] survey state-of-the-art techniques for automatically getting security proofs of formal models closely depicting the logic of protocol implementations. They

[3] https://en.wikipedia.org/wiki/OSI_model

comment extensively on work that extracts models from code to further validate widely deployed implementations written in C (rather than libraries as in our case), limited to a subset of the language. In particular, they conclude that approaches for model extraction do not target arbitrary legacy code, instead they introduce some restrictions on how the code should be written, for instance by adding annotated semantic information. A relevant technique in this realm was developed by Aizatulin et al. [8]. To automatically generate a formal model from an implementation in C, analysts need to add annotations on the original source code; note therefore, that they require knowledge on the C language and to have at least a broad knowledge of the protocol.

Finally, Wikipedia offers a compilation of available tools for automatic reverse engineering into UML diagrams [193]. Unfortunately, the information sources are poorly referenced; nonetheless, at least the information regarding the tools mentioned in this chapter is aligned with the information presented there.

3.4 Concluding Remarks

We realized that, although there exists mature and efficient software for reverse engineering executable applications—mostly for the Object Oriented paradigm—current solutions for non-executable implementations still require a lot of manual effort, which could be minimized by having access to people knowing about the implementation or to functionality-related documents.

Adhering to the approach presented in this chapter, we abstracted specifications of the p≡p key distribution and p≡p authentication protocols implemented in pEpEngine, along with the algorithms of relevant functions. These models provide the base of our formal study of the protocols, which is the subject of future chapters.

As a final note, we realized that the lack of specifications is much more common than it seems. A considerable amount of research on formal verification of protocols reports having performed RE related tasks in order to obtain the models used in their work (e.g. [52]). Although we presented the method by addressing libraries in C, the underlying steps are also applicable to structure a manual RE process or to optimize RE tasks with similar conditions. We therefore hope that this chapter serves as support to others for reducing the time and effort invested in required tasks which are not necessarily linked to security analysis.

Part I

Formal methods for the analysis of security protocols and ceremonies

4 Formal Verification of Security Protocols

In this part of the thesis we focus on the formal analysis of secure email protocols. This short chapter is aimed at providing a brief introduction to both, the area of research and the principal technique in which we rely.

4.1 Proofs of Security: Approaches

Security protocols are a finite number of communicating processes running in parallel, that rely on the use of cryptographic primitives for achieving various security goals in insecure environments. Security protocols are critical components of any distributed security architecture; in particular, they underlie current secure digital communications. However, designing distributed systems is already error-prone; moreover, functional testing cannot be used to asses the security of a protocol as flaws appear only in the presence of an adversary and might be caused by extremely subtle things that testing cannot hope to detect, e.g., a mistake in the order and way in which a message is exchanged. Yet, in a security protocol execution all the participants exchange messages through a network typically controlled by the adversary. Thus, providing strong guarantees about the achievement of the security goals is essential.

Two approaches have been developed for modeling security protocols within a mathematical framework, which enables the derivation of rigorous proofs of security: the *computational* (or cryptographic) approach and the *symbolic* (or formal methods) approach (interested readers can refer to [38] for expanding on the topic).

Computational models typically capture the cryptographic games used for standard security definitions in the literature (e.g. see [106] for an introduction on cryptographic schemes and proofs). In these models, messages are represented with bitstrings, primitives are functions on bitstrings, and the attacker is a probabilistic polynomial-time (PPT) Turing machine. Proofs of security rely on so called standard cryptographic assumptions, which refer to the computational hardness of solving certain problems in number theory—such as the integer factorization problem. These proofs are either completely manual or they require a considerable degree of interaction with dedicated tools (e.g. CryptoVerif [35]).

Symbolic models describe the interaction among participants and the exchange of messages. Here, messages are modeled as terms and cryptographic primitives are assumed to work as perfect black boxes operating with those terms; the attacker is restricted to use only the primitives modeled. Proofs of security are based on formal verification methods, which have simplified the development of automated tools for protocol verification (e.g., ProVerif [36], Tamarin [20]).

Note that while symbolic proofs are relatively easy but at a high level of abstraction, computational proofs provide strong security guarantees at the cost of being tedious and highly error prone. Therefore, an area of research attempts to combine both approaches, to simplify computational

proofs. Under additional assumptions, the goal is to prove security guarantees in the computational model by hinging on guarantees proven in the symbolic one [54].

Given their characteristics, computational models are more suitable for security proofs related to the cryptographic primitives involved in a protocol, such as signatures and encryption, while symbolic models allow to seek flaws in the logic of a protocol.

Our work resides mainly in the symbolic model, given that we study security protocols built on top of proven cryptographic primitives.

4.2 The Symbolic Model

Formal verification of security protocols has been successfully applied in diverse domains, from the commonly cited Needham-Schroeder key transport protocol, found vulnerable to a specific adversary almost 20 years after its publication [118], to authentication standards (e.g., [18, 57]) and TLS 1.3 [91], to name a few. The results of this approach evidence that security protocols can be vulnerable even without breaking the underlying cryptography.

Multiple techniques have been developed for modeling and analyzing security protocols (e.g. [46, 126, 159, 137, 2]), which are extensively covered by the literature. Some comprehensive texts on foundations of symbolic protocol verification and related techniques include [125, 53] and references therein.

Invariably, in order to determine whether a protocol is secure against an adversary with specific capabilities, three elements need to be formally defined: the model of the protocol, the set of security properties to be proved, and the threat model in which the verification is framed.

Then, rules in a deduction system usually model what an adversary can compute.

Threat model. The standard adversarial model is determined by three assumptions, introduced by Dolev and Yao [69] in 1881:

1. Cryptography is perfect. So, for instance, a message can be decrypted only by someone knowing the decryption key.
2. Messages are abstract terms, and thus, either the adversary learns the complete message or nothing.
3. The network is under full control of the adversary, who can read, modify, insert, delete and transmit messages.

Protocol model Two elements need to be considered: the messages and the actions of the participants. There are different formalisms for this task. In this thesis, we use the applied pi calculus, which we introduce in Section 4.4.

4.3 Security Properties

Most security properties can be classified into two categories: trace and equivalence properties.

Trace or reachability properties: they can be defined on each run of the protocol and usually are used to ensure that unwanted behavior is not reachable (hence the name) in any of the execution traces (possible runs) of a protocol. A property is satisfied when it holds for all traces in the model. Weak secrecy and authentication are usually modeled as trace properties.

Equivalence or indistinguishability properties: they are defined in terms of the notion of the adversary's incapability of distinguishing two given protocols. For instance, given the models of two

protocols: a protocol P under study, and its specification S, if an adversary interacting with one of them cannot decide whether the interaction occurs with P or with S, then P is indistinguishable from S, which implies that the protocol meets the given specification. Strong secrecy and anonymity are examples of equivalence properties. Indistinguishably is also known as observational equivalence.

Automating proofs of equivalence properties is more difficult than automating proofs of trace properties, since they need to be expressed in terms of relations between traces.

The properties that concern our analysis fall in the first category. For further reference on equivalence properties, there are several sources; [47] provides an informative coverage on related work, in addition to introducing a procedure for verifying equivalence properties

The fundamental properties that concern secure communications are related to privacy and authentication. Informally, they refer to the following:

- Syntactic (weak) secrecy: the adversary $\mathcal{E}$ does not learn the content of a message s. This is usually expressed as a trace property: a state in which the knowledge of $\mathcal{E}$ allows the adversary to deduce s, is never reached.
- Strong secrecy (non-interference): an adversary $\mathcal{E}$ cannot detect any change in a message s—as opposed to weak secrecy where $\mathcal{E}$ could have partial knowledge of a message, and yet, not know the final value of s. Even if the content of s belongs to a publicly known set of values (e.g., the name of candidates in an election), $\mathcal{E}$ cannot distinguish the particular value of s (e.g., the vote of a person). Strong secrecy is an equivalence property.
- Authentication: generally, this property ensures two participants of a protocol that they are interacting with the participant that they believe, i.e., a participant A runs the protocol apparently with a participant B, if and only if B runs the protocol apparently with A. There are many variants of this definition, we will formalize some of them in the corresponding upcoming chapters.

4.4 ProVerif and the Applied-pi Calculus

ProVerif [36] is an automatic verifier for cryptographic protocols under the Dolev-Yao model (introduced above). The tool's proof derivation strategy consists in abstractly representing a given protocol by a set of Horn clauses, and then applying a resolution algorithm on those clauses, to determine whether a given fact can be derived from the clauses.

Protocols in ProVerif are modeled in a dialect of the applied pi calculus [1], a process calculus extended with cryptographic formalizations, which enables specifying and reasoning about security protocols. Participants are represented as *processes*, whose input parameters symbolize the initial knowledge possessed by the participant. The exchanged messages are represented by *terms* sent over public or private channels. A so called *equational theory* allows to define an equivalence relation over terms, cryptographic primitives, and other functions occurring in the protocol; this equations are translated by ProVerif into inference rules for the resolution algorithm.

The core syntax of ProVerif, which we define next, largely corresponds to the applied pi calculus language. The enriched grammar of ProVerif is fully specified in [45] and further details on the verification process can be found in [38].

Terms. Given the sets $\mathcal{V}$ and $\mathcal{N}$ of variables and names respectively, and $\mathcal{F}$ a set of function symbols (known as a *signature*) each with an arity, the set of terms $T(\mathcal{V},\mathcal{N},\mathcal{F})$ is defined as:

$$M,N ::= x \in \mathcal{V} \mid a \in \mathcal{N} \mid f(M_1,\ldots,M_n) \in \mathcal{F}$$

Typically, variables represent unspecified parts of a message and can be substituted by terms, names model channels and atomic data—e.g. keys or nonces—and functions represent cryptographic primitives, such as encryption, decryption, and hash functions.

As a remark, terms in ProVerif are typed; while this helps to detect mistakes in the protocol specifications, to expand the scope of detected attacks typing is not considered in the verification process. The definition above is closer to applied pi calculus, which is untyped, however, the code that we developed for this thesis and which is included in Appendix A, is typed.

Processes. Processes represent programs which implement the actions and interactions of the participants in the protocol. For $M,N \in T$, the grammar of processes is as follows:

$$\begin{aligned} P,Q ::= \quad & 0 \mid \texttt{out}(N,M);P \mid \texttt{in}(N,x);P \mid P|Q \mid !P \mid \texttt{new}\,a;P \mid \\ & \texttt{if}\ M\ \texttt{then}\ P\ \texttt{else}\ Q \mid \texttt{let}\ x = M\ \texttt{in}\ P\ \texttt{else}\ Q \end{aligned}$$

Roughly, the process 0 (*nil*) is interpreted as a process that does nothing. The output process (`out`) outputs the message M on the channel N and then executes P. The input process (`in`) waits for a message on channel N, and upon reception, binds x to the input message and executes P. $P|Q$ is the parallel composition of P and Q. Replication (!) behaves as an infinite number of copies of P running in parallel. The restriction process (`new`) creates a new private name a and then executes P, which allows the introduction of fresh values. The conditional construct behaves as usual. Finally, if M is a valid term, `let` binds x to M and then executes P; if the evaluation of M fails, then Q is executed.

Equational theories. An equational theory E is a set of equations $u = v$, where u and v are terms in $T(\mathcal{V},\mathcal{N},\mathcal{F})$. The equivalence relation $=_E$ is closed w.r.t. reflexivity, transitivity and substitutions. Equational theories allow to express properties of cryptographic primitives, functions and data structures, in the term algebra.

For instance, an equational theory defining the properties of asymmetric encryption is:

$$\texttt{adec}(\texttt{aenc}(m,\texttt{pubKey}(sk)),sk) = m$$

which models that, given a pair of secret and public keys (sk, $\texttt{pubKey}(sk)$), a message m encrypted with $\texttt{pubKey}(sk)$ can only be obtained by decrypting the ciphertext with sk.

Events. Events are constructs of the form $\texttt{event}\ e(t).P$, where e is taken from a set of function symbols $\mathcal{E}v \neq \mathcal{F}$. Events allow to annotate or mark important states reached by the protocol and do not affect the protocol's behavior; they are similar to breakpoints or flags used in software development for debugging purposes.

Correspondence properties Correspondence properties (originally called assertions) [194] are expressions of the form $e \Longrightarrow e_1 \wedge \ldots \wedge e_i$, with $e, e_i \in \mathcal{E}v$. They are interpreted as: *if an event e is executed, then events $e_1,\ldots,e_n$ have been previously executed*.

An *injective* correspondence requires each occurrence of e to be preceded by a unique set of occurrences of determined events. This is denoted in ProVerif as $\texttt{inj} - \texttt{event}(e_i)$, for each e_i required to have a unique instance.

Authentication and secrecy can be defined as correspondence properties, formalized as "correspondences" for the applied pi calculus by Blanchet [37]. ProVerif can verify complex non-injective and injective correspondence properties, and a limited class of equivalence properties, e.g. strong secrecy and guessing-attack resistance in password-based protocols.

ProVerif is sound, i.e., given P the model of a protocol, if the verifier proves that a property is satisfied in P, then that property really holds in P (the proof is correct); however, ProVerif is not complete, which means that the tool is not able to prove all the properties that hold in a model.

5 The Honeywords System under a Code-corrupted Login Server

When we think about email security in general, one of the first concerns that arises refers to the user's access to a mailbox. As we stressed in the Preliminaries chapter, the compromise of email data in storage, in particular of a password database, is an identified vulnerability of email systems (and indeed more generally, of password-based authentication systems). Therefore, before looking deeper into end-to-end security protocols, we look into a measure to protect the privacy of email from the most outer layer: detection of illegitimate access to mailboxes.

Due to its easy-operation, scalability, compatibility and low-cost advantages, the most widely used method for authenticating users in email services (and more generally in IT systems) is based on passwords [41]. Many prevailing email clients, such as Gmail, have implemented two-factor authentication; this feature is however optional to the user and the default authentication method still requires only a password.

Password-based authentication in a client-server setting simply requires users to submit a username and a password through an interface, which then servers check against a database of legitimate username-password pairs, granting access if the values are found.

The security of the authentication process lies on the assumption of passwords being only known to the corresponding users. To achieve so, passwords are expected to be transmitted over encrypted channels (e.g. TLS), users are in charge of keeping them secret, and servers are expected to securely store them—e.g. hashed with an appropriate function, together with some random data known as "salt"—in a so called *password file*.

Yet, there are well-known ways in which adversaries can learn valid passwords, thereby, illegitimately gaining "authorized" access to the system, which makes the password leakage unnoticeable. For instance, users are prone (intentionally or not) to reveal their credentials; or password files can be stolen from servers, and hence hashed passwords exposed to offline dictionary attacks.

Password database breaches have actually been so common in the last years, that at the beginning of 2019, a collection of 2.2 billion unique usernames and associated passwords composed of data from multiple previous leakages (e.g., from Dropbox, Yahoo and Linkedin) started to be freely distributed on hacker forums and torrents[88], and other collections have followed[1].

Besides the huge number of passwords exposed in each breach, an alarming aspect is the large delay that passed between the attacks and their detection (e.g. 3 years in the case of Yahoo [133]); and failing to detect a password compromise on time, worsens the problem as it delays the application of countermeasures to limit the damage.

Although some systems have developed solutions to improve the detection of a password theft—e.g. Google monitors suspicious activities and invites users to review the devices and locations from which they have accessed their account—raising awareness on a password file compromise is more

[1] The increasing number of password databases compromise in the last years has motivated the creation of websites, such as `https://haveibeenpwned.com/` and `https://www.avast.com/hackcheck`, which allow users to verify if their accounts have been compromised in a data breach.

critical as the whole database of users is compromised at once.

To detect unauthorized logging attempts derived from a password file compromise, in 2013 Juels and Rivest [101] proposed an alternative password-based authentication scheme, which they called the *Honeywords System*. In this chapter, we study the security of the Honeywords System against an active adversary who has successfully modified the code of the component that processes the login data. This problem was left open in Juels and Rivest's work.

After introducing the Honeywords System in Section 5.1 and formalizing the problem setting in Section 5.2, we show in Section 5.3 an attack to the Honeywords System within the defined threat model. Then, in Section 5.5 we describe a new cryptographic protocol that arguably removes the weakness, which we prove secure with a formal analysis in Section 5.6. Finally, in Section 5.7 we discuss this solution in a wider perspective.

5.1 Framework: the Honeywords System

A Honeywords System stores a user (hashed) password together with a list of decoy words, called *honeywords*. The honeywords and the password together are called *sweetwords* and they provide each user account with a list of possible passwords, only one of which is genuine.

Honeywords should be generated in a way that the real password remains indistinguishable from the generated words; for instance, "redsun3" is a good honeyword for "whitemoon5". *Flatness* is a property that measures the probability of an adversary guessing which is the correct password from the sweetwords [101, 74]. Thus, flat honeyword generation algorithms—which create words that have the same probability to be selected as the real password—are assumed.

Since it is very unlikely for a user to type a honeyword purely by chance, any attempt to log in with a honeyword instead of the password might be an indicator of a password file leakage, in which case, the system flags the event and contingency actions can be taken (e.g., system administrators are alerted, monitors are activated, user's execution rights are reduced, etc).

Note however that the Honeywords System does not impede a password file theft nor it avoids impersonation; an adversary who has cracked the hashed sweetwords can still succeed in guessing the correct password of a user by random choice.

Authentication in the Honeywords System. The Honeywords System consists of (i) a computer system that grants access to a user who enters valid login credentials, which for concreteness, we will consider to be formed by a *User Interface (U)* and a *Login Server (LS)* that processes login requests from U; and (ii) a hardened secure server that assists in the password verification, known as the *Honeychecker (HC)*.

LS keeps the password file, in which for each registered user u, there is an ordered list of hashed sweetwords $sw_u = [h(w_1), \ldots, h(w_k)]$, where k is the number of sweetwords fixed in the system. In turn, HC stores c_u, the index of u's password in the list of sweetwords.

At authentication, the system runs a simple protocol, depicted in Figure 5.1: U sends a username and a password (u, w) to LS; then, LS searches u's sweetwords for the hashed version of w. If no match is found, access is denied. Otherwise, the LS sends to the HC the username together with j, the index of w; this communication occurs over a dedicated or an encrypted and authenticated channel. The HC checks whether j corresponds to the stored index c_u, in which case the access is granted. If the test fails, the HC acts according to the implemented policy.

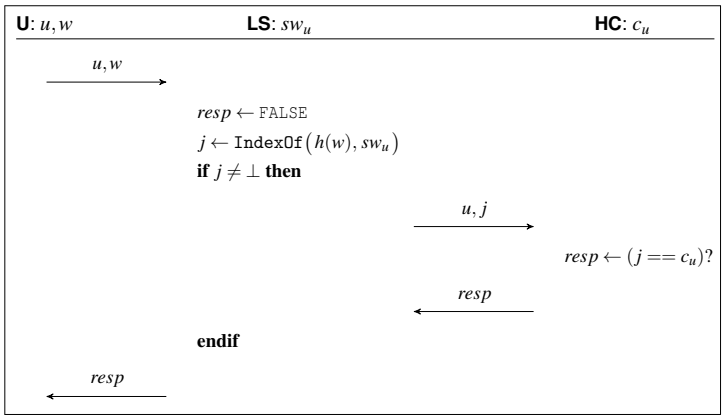

Figure 5.1: Honeywords System's authentication protocol with a responsive HC

5.2 Adversary Model

Were the Honeychecker (HC) compromised, the security of the Honeywords System would be downgraded to the level that classical authentication systems achieve, as the sole index of the password does not have any use for an adversary $\mathcal{E}$—the password file would still be needed. Several interesting threats arise if instead we consider a compromised Login Server (LS).

Here, we focus on *code corruption*, which invariably involves malicious modification of a specific piece of code, in this case, the one implementing the steps executed by LS in the authentication protocol. To give a precise definition of the code corruption thread model that we consider, we first introduce our assumptions.

Assumption 1. Before corrupting the compromised code of the LS, $\mathcal{E}$ has retrieved the sweetwords from a password file, however $\mathcal{E}$ does not know which of them are passwords.

Assumption 1 follows from the original paper and reflects the capability of $\mathcal{E}$ to steal the password file and retrieve the plaintext of the k sweetwords of any given user u; yet, $\mathcal{E}$ cannot distinguish which one among u's sweetwords is the password. Hence, if we call a *successful login* the event of a pair (id, pwd) being granted access to the system, with id a username and pwd a candidate password, then the probability of $\mathcal{E}$ having a successful login with a random guess of pwd for a fixed id is $1/k$.

Note that additional factors, such as public information or knowledge of social facts about a specific user, might be used to increase the probability of guessing the password. Here however, we focus on general attacks, where the adversary intends to gain access with any pair of valid credentials, without using additional information regarding a specific target.

Assumption 2. The goal of a *code corruption attack* is to increase the adversary's probability of a successful login, with respect to the probability of randomly guessing the password from the list of sweetwords.

There are many ways in which $\mathcal{E}$ could modify the file that implements LS's tasks in the authentication protocol, hereafter ls.c. For instance, $\mathcal{E}$ could reprogram the LS to completely change its behavior, e.g. to make it play chess[2]. We are not interested in such attacks as they do not help $\mathcal{E}$ to increase its probability of successfully logging in. For a similar reason, we exclude attacks that shut-down the systems or cause Denial-of-Service.

[2] In 2006, R. Gonggrijp did so to prove insecure a Dutch electronic voting machine.

Another possibility is the modification of ls.c so that access is always granted, independently from the HC's answer; but administrators could spot this attack by analyzing login statistics. Similarly, a back door for establishing an alternative channel between $\mathcal{E}$ and the LS to leak information outside the protocol's message flow, can eventually be detected (e.g., by monitoring the network traffic), leading to have a safe version of the ls.c reinstalled.

Perhaps for some adversaries leaving a trail might not be a concern, as long as they manage to log in and e.g. extract sensitive data. Here, we are concerned about a corruption that allows $\mathcal{E}$ to have long-term continuous access to the system in an unnoticeable way.

More precisely, we focus on corruption of the file ls.c in such a way that LS's behavior remains the same from the perspective of the components with which it interacts—e.g. the HC, system administrators and users—to avoid raising alarms that would result in restoring the original ls.c. Thus, our threat model consists of a scenario where the LS is code corrupted as per definition 5.2.1, and the HC is secure.

Definition 5.2.1. Let ls.c be a file in LS which implements the role of LS in the Honeywords System authentication protocol. *Code corruption of LS* occurs if the adversary modifies ls.c in such a way that LS does not change its observable behavior.

According to the previous discussion, definition 5.2.1 says that if $\mathcal{E}$ wants to communicate with the corrupted LS, it must be through the same channels from which legitimate users log in, and respecting the message flow of the honest protocol. The definition however does not prevent $\mathcal{E}$ from using the knowledge obtained by cracking the password file; for instance, learned user's IDs and sweetwords could be hard-coded in the corrupted ls.c for further use.

Note also that $\mathcal{E}$ could still retrieve information from legitimate channels, for example, via the *resp* message in Figure 5.1. And, as we will discuss in Section 5.3, LS communicating back to $\mathcal{E}$ enables a powerful attack that breaks the original Honeywords System. Yet, strictly speaking, this modification could be considered "visible" as, depending on the implementation, a careful user or an administrator could spot the information leakage.

Hereafter, by default we interpret definition 5.2.1 strictly, thus excluding the possibility of this retroactive communication; but were relevant, we also discuss what happens if we relax this constraint in the definition and allow LS to leak information to the adversary, since this might be a strong incentive for code corrupting the LS.

5.3 A Code-corruption Attack

For illustration purposes, consider Algorithm 1 to be the pseudo-code of ls.c, where Passwords is the password file, sweetw_u is the list of sweetwords of user u in Passwords, and H is an appropriate hash function[3].

In turn, Algorithm 2 shows a corrupted ls.c compliant with Definition 5.2.1. The attack relies on $\mathcal{E}$ hard-coding a username—here `eve`—in ls.c, which serves as a flag to trigger malicious actions. LS stores the first credentials successfully authenticated by the HC (16), so that in the future, every time that `eve` attempts to login (5), the malicious code will load the stored credentials (6), allowing $\mathcal{E}$ to

[3]Remark that approaches to secure passwords in storage keep constantly upgrading, as computing resources become more achievable to adversaries; withal, there is not yet an approach proven secure and dictionary attacks still succeed, specially finding weak passwords. We expand on this topic in Chapter 8; here, we assume for simplicity that a "decent" approach is used, such as using a password-based hashing function to mask the password together with a salt.

Algorithm 1 Login Server Authentication

```
1:  procedure ls.c(Passwords)
2:      while TRUE do;
3:          ReceiveFrom(U, (u, w));
4:          j ← IndexOf(H(w), sweetw_u);
5:          if j < 0 then
6:              resp ← FALSE;
7:          else
8:              SendTo(HC, (u, j));
9:              ReceiveFrom(HC, resp);
10:         SendTo(U, resp);
```

Algorithm 2 Code Corrupted LS

```
1:  procedure ls.c(Passwords)
2:      (u', w') ← (⊥, ⊥)
3:      while TRUE do;
4:          ReceiveFrom(U, (u, w));
5:          if (u' ≠ ⊥) ∧ (u == "eve") then
6:              (u, w) ← (u', w')
7:          j ← IndexOf(H(w), sweetw_u);
8:          if j < 0 then
9:              resp ← FALSE;
10:             if u == "eve" then
11:                 (u', w') ← (⊥, ⊥)
12:         else
13:             SendTo(HC, (u, j));
14:             ReceiveFrom(HC, resp);
15:             if resp == TRUE then
16:                 (u', w') ← (u, w)
17:         SendTo(U, resp);
```

log in with them regardless the submitted password. As long as the credentials are valid, $\mathcal{E}$ will have access to the system. If the stored credentials become invalid (e.g., u changed password), they are deleted (11) and a new valid pair is stored the next time that a user logs in successfully (16).

Alternatively, the corrupted file could store only the valid j (in step 16) and, in a next round, skip the search in sweetw_u (7), sending the stored j to the HC (13) (however, possible variations in the execution time might rise the alert of a suspicious behavior in a statistical analysis). In any case, LS gets knowledge of a user's valid password, even though $\mathcal{E}$ gains access to the system without the need to learn it.

For the attack to work, $\mathcal{E}$ would only need to wait a reasonable delay after corrupting ls.c, to allow for a legitimate login attempt to have place. To verify that valid credentials have been stored, $\mathcal{E}$ could try to log in with a dummy password; since the username `eve` is not registered, in the worst case the access would be denied. With this attack, $\mathcal{E}$ raises the probability of a successful login from $1/k$ to 1, as the password has been identified.

This attack is possible because LS can learn the correct w for u. Only hashing the password in U's side before sending it to LS would not solve the problem, since the LS can still search for the received hash value in sweetw_u, hence learning the index. Moreover, under a relaxed Definition 5.2.1, LS can send the hash back to $\mathcal{E}$, who by Assumption 1, is able to obtain the corresponding password.

5.4 Sketching a Solution

We exclude pragmatic fixes that deal with code corruption, such as regular integrity checks of ls.c or forcing users to frequently change passwords, as they do not tackle the real weakness of the system.

The main problem allowing the attack in the previous section seems to reside in the simultaneous occurrence of three facts, which a solution that aims to make LS resilient to code corruption of ls.c can prevent, by satisfying all of the following security requirements:

R1 LS should not receive usernames and (candidate) passwords in clear

R2 LS should not be able to store a password that has been validated by the HC

R3 LS should not be able to retrieve the index of a pair of credentials that were not submitted through the running session's legitimate channel.

To address these requirements, we propose the following countermeasures:

(i) Shuffling sweetw_u after each time that LS submits a valid password HC: this stops LS from associating a valid password with an index;

(ii) Blinding each element in sweetw_u after each time that LS submits a valid password HC: this prevents LS from associating a valid password with a username;

(iii) Allowing LS to learn the string to be searched in sweetw_u only when user u submits it, and never in plaintext: this prevents LS from storing passwords in clear and from reusing them at will.

Countermeasures (i) and (ii) deal with R2, while (iii) addresses R1 and partially R3. We focus first on R2 and later on elaborate on R1 and R3.

To implement (i) and (ii) we take inspiration from re-encryption mixnets, initially proposed by Chaum [48]. This scheme is designed to transform a list of ciphertexts $c_1, ..., c_n$ (usually encrypted with ElGamal) into another list of ciphertexts $e'_1, ..., e'_n$, with the same plaintexts in permuted order. Some uses of mixnets include making email untraceable, as in the original paper, and e-voting (e.g., [92]).

Here, we propose the use of a one-node mixnet, played by the HC. We do not require encryption in the strict sense given that there is no decryption associated; instead we want to blind the password to the LS, therefore, we call this step *blinding*.

Shuffling consists on applying a random permutation π over the elements of a given row $[w_1, \ldots, w_k]$, thus obtaining $[w_{\pi_1}, \ldots, w_{\pi_k}]$. After shuffling the row, the corresponding index c stored by the HC needs to be updated to the new position $\pi(w_c)$.

Blinding could be implemented by modular exponentiation. At user u's registration, HC appropriately selects a generator g of a cyclic multiplicative subgroup $\mathbb{G}$ of order q, and generates a list of sweetwords $\overline{w} = [w_1, ..., w_k]$. Then, HC blinds each w_i in $\overline{w}$ by applying $f_{r_0}(w_i) = g^{r_0 \cdot w_i}$, where r_0 is an element of $\{1, \cdots, q-1\}$ sampled at random. Thus, the initial row of u in Passwords is $f_{r_0}(\overline{w}) = [f_{r_0}(w_1), \ldots, f_{r_0}(w_k)]$. [4]

At the n^{th} time that LS submits a valid password to HC, HC chooses a new random number $r_{n+1} \in \{1, \cdots, q-1\}$ for blinding the current row $f_{r_n}(\overline{w})$, as follows: for each $w_i \in \overline{w}$,

$$f_{r_{n+1}}(w_i) = g^{r_{n+1} \cdot w_i} = g^{r_{n+1} \cdot \frac{r_n}{r_n} \cdot w_i} = (g^{r_n \cdot w_i})^{\frac{r_{n+1}}{r_n}} = f_{r_n}(w_i)^{\frac{r_{n+1}}{r_n}}.$$

Shuffling and blinding can be implemented as an atomic block within the HC, as shown in Figure 5.2. The idea is for HC to receive $f_{r_n}(\overline{w})$ along with each query from LS, so that the row gets shuffled and blinded with a freshly generated r_{n+1}; then, HC returns $f_{r_{n+1}}(\overline{w})$ to LS, together with the index's verification result.

[4] A more accurate notation of the elements involved in shuffling and blinding, would require the use of the subscript u (e.g. c_u, g_u, $f_{r_0}(\overline{w}_u)$), to stress that the processes are user based. To lighten the notation we omit the index u, but ask the reader to keep this distinction in mind.

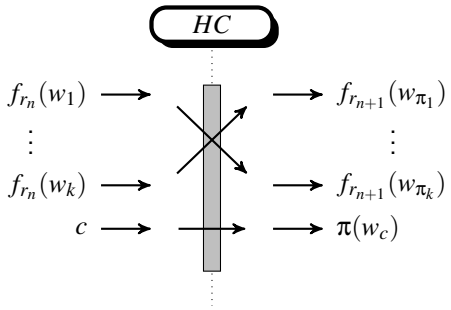

Figure 5.2: Shuffling sweetw_u, blinding $\overline{w}$, and updating the index c.

Note however that, each n^{th} time that a user u submits a password x, LS needs to calculate $f_{r_n}(x) = g^{r_n \cdot x}$ before being able to search for x's index in $f_{r_n}(\overline{w})$. For doing so, LS needs to have received r_{n-1} (or $g^{r_{n-1}}$) too. This means that, at the end of round n, LS knows x and r_{n+1}, which allows it to further query HC with this information, disregarding the user's input.

To prevent this situation, we need to implement countermeasure (iii), for which we propose the use of a One-Time-Password (OTP) device. We briefly justify this decision in the next section, but readers interested only in the solution, can skip it and restart reading from Section 5.5.

OTP against code-corruption. As we saw, LS can use HC as an oracle because LS has access to all the elements needed to generate a valid query, namely, a candidate password—which is still in plaintext—and the exponent used for blinding. Thus, to deal with the problem we would need a separation of duties, so that the component that can check the index, is not the same as the one that blinds the submitted word.

A natural idea regards the addition of a component, which we call Key Register (KR), who would store r_n. At login time, KR receives (u, y) from User Interface (U) and calculates $f_{r_n}(y)$; then, KR forwards this value to the LS, who has also received u from U. The rest of the process remains the same except that the HC returns $f_{r_{n+1}}(\overline{w})$ to the LS, but r_{n+1} only to the KR.

This solution prevents LS from receiving passwords in plaintext; the problem now is that, if we comply with the original Honeywords System's design [101], the only component assumed to be hardened secure is the HC. And considering that the LS can be corrupted but the KR cannot, would be unjustified. So, an adversary compromising both components would once again have the power of obtaining the blinded value of a word w, and then determining whether w is a valid password. By exchanging the adequate information between KR and LS, $\mathcal{E}$ would be able to store a valid pair of credentials for future use.

This reasoning applies to any other component that were to have a role between U and HC. Alternative ways for implementing (iii), such as using timestamps from the U's side as a proof of freshness, do not work either since LS stands in the middle and can compromise those messages.

Therefore, in addition to the previous requirements, we need the U to be the only component able to blind the password, and we need to enforce the result of such a blinding to be the value submitted to the HC. So, the intuition suggests a "synchronization" between the U and the HC, occurring over a channel which is controlled neither by any component of the Honeywords System nor by the adversary, to avoid man-in-the-middle (MITM) attacks.

5.5 A Code-corruption Resilient Protocol ($\mathcal{P}$)

The solution that we propose requires the user to posses an OTP device, which is employed to generate a random nonce r_n per use. The OTP device and the HC share the initial seed and the algorithm to generate r_n; this information enables U (e.g., the user's browser) to blind the candidate password using the same r_n that is generated by the HC—somehow mimicking the role of the KR described above—thereby addressing R1.

The OTP device serves as pseudo-random generator and the OTP as proof of freshness, since this value is synchronized with the HC's output; this serves to address R3. Here we assume an event-based device (a.k.a. HOTP for HMAC-based One-Time Password [129]), i.e., one that generates a new code at the press of a button, which remains valid until it is used by the application[5].

Figure 5.3 describes the protocol, which for simplicity, we will denote as $\mathcal{P}$. $\texttt{OTP}(n)$ represents the action of generating the n^{th} OTP value (step 1), which U uses to compute the blinded password $f_{r_n}(w)$, which in turn is sent to the LS along with the username u (step 2).

Then, $\mathcal{P}$ follows as discussed in Section 5.4: the LS searches for $f_{r_n}(w)$ in the row of user u (step 3), which the HC has reshuffled and blinded in a previous session using the same OTP number that U has used now to blind the password; if an index j is found, it is submitted together with the username and u's row of blinded sweetwords (step 4). The HC decides whether the submitted index corresponds to the password (step 5); in affirmative case, the HC shuffles and blinds the row, and updates the index of u, c_u, according to the new order (step 6). The shuffled and blinded file is returned to the LS (step 7), who notifies the decision to the user (step 8).

Informal Security Analysis. $\mathcal{P}$ satisfies R1, R2 and R3: the LS cannot observe the submitted passwords in plaintext because it does not have r_n; the shuffling and blinding processes prevent the LS from associating, with probability more than $1/k$, a blinded password with its correct index; and even if the LS learns that a particular $f_{r_n}(w)$ is a valid password, blinding prevents LS from reusing that string at any time, as it needs the randomness inserted by U in order to calculate the new index. Additionally,

- even if two users choose the same password, it is unlikely that the blinding values are the same, since the initial seeds of the OTP devices differ;
- in the case that the LS submits the correct index by chance, access will be granted once but the LS will not be able to reuse the index, given that the HC changes c_u. This situation would actually invalidate future legitimate login attempts, as the r_i from U and HC will be no longer synchronized. While this counts as a Denial-of-Service (DoS), it is not an attack according to Assumption 2 as it does not increase the probability of $\mathcal{E}$ to gain access;
- the protocol is secure under a relaxed Definition 5.2.1 too, since even if the LS sends back to U a particular $f_{r_n}(w)$ learned to be a valid password, and $\mathcal{E}$ retrieves w, neither w nor $f_{r_n}(w)$ provide enough information to gain access. The value of $f_{r_{n+1}}(w)$ would still be needed, which $\mathcal{E}$ cannot generate without the OTP device.

In consequence, we argue that the protocol is resilient to a code-corruption of LS attack, aligned with Definition 5.2.1, which we prove with a formal analysis in the next section.

As a remark, the actions in step 6 need to be *executed atomically*, otherwise and if $\mathcal{E}$ learns the result of the validation before, LS could take advantage of the HC being in an inconsistent state and

[5] E.g. `https://www.microcosm.com/products/oath-otp-authentication-tokens`

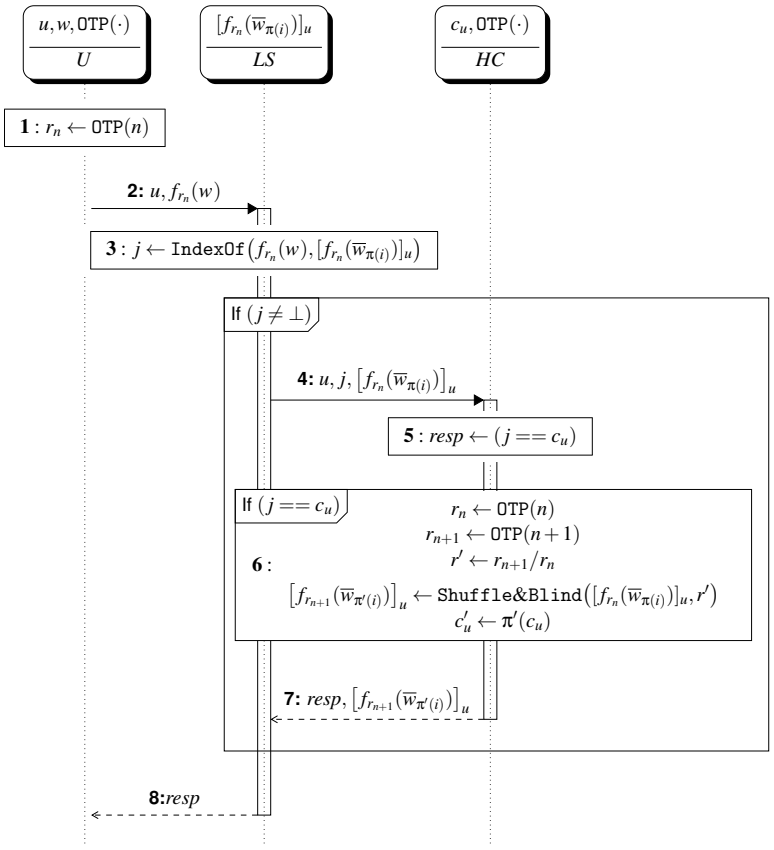

Figure 5.3: Authentication Protocol ($\mathcal{P}$) resilient to code corruption of LS

perform an attack (see Section 5.6).

Note also that the use of an event-based OTP device is a relevant assumption. Were a time-based device used instead, $\mathcal{E}$ could submit an index learned to be correct, during the time-window before the OTP gets regenerated. The time-interval for $\mathcal{E}$ to succeed would be limited as opposed to the original system, but $\mathcal{P}$ would still be vulnerable.

5.6 Formal Security Analysis

We used ProVerif to verify the security of both, the original protocol (Figure 5.1) and $\mathcal{P}$ (Figure 5.3), against the defined threat model of code corruption. The models and relevant traces obtained from the verification are included in Appendix A.1; here, we elaborate on the modeling decisions.

5.6.1 Analysis of the Honeywords System

As we already found the original Honeywords System vulnerable to code corruption of LS, this verification is mainly aimed at understanding how to capture the attacker model correctly—since the default assumption of ProVerif is a Dolev-Yao attacker—and how to correctly interpret the results, discarding attacks that do not correspond to our defined threat model.

Our protocol model consists of three processes: `user`, `loginService` and `honeychecker`, representing respectively the U, the LS and the HC. The LS is an active attacker since it is able to read and send messages from and to the HC; the channel between LS and HC is thus *public*. In contrast, the channel between U and LS is *private*, as this communication is assumed to happen over TLS or similarly secured channels.

ProVerif adds all information sent in public channels to the adversary's knowledge, hence, the latter use of a private channel prevents the attacker from learning the submitted password at any time. It also rules out the obvious guessing attack—which is always possible since Honeywords System is not designed to avoid it—allowing the verifier to find other attacks related to the protocol's flow.

Our security property expresses that, whenever the HC sends a positive answer to the LS for a submitted pair of user and index (u, j), all of these three actions had previously occurred: (1) the user u logged in with password p, (2) LS found p in the list of u's sweetwords at index j, and (3) the value stored in HC for u is equal to j. This is formulated with the correspondence:

$$correctIndex(u, j) \quad \Longrightarrow \quad inj-event\big(indexFound(u, p, j)\big) \;\&\&\; inj-event\big(usrLogged(u, p)\big).$$

Injectivity in the expression (*inj-event*) models the HC processing only once each request received from LS, preventing interaction between LS and HC in the absence of a user, since this communication is assumed to occur over a secure channel (see Section 5.1).

Result. As expected, the verification indicates that the property does not hold. The attack found corresponds to the attack described in Section 5.3, and shows how once the attacker (in this case the LS) gets a positive answer from the HC, it is able to send a new validation request to HC using the correct index, and hence, gaining access to the system. The attack trace contradicts injectivity, because a fresh event $usrLogged(u, p)$ did not have place before that second request. These observations support our model design for code-corruption and provide formal evidence that a Honeywords System resilient to the flaw must satisfy requirements R1, R2 and R3.

5.6.2 Analysis of $\mathcal{P}$

For protocol $\mathcal{P}$, we model all the channels as *public*, to allow a corrupted LS to interact with HC at any time and learn the inputs from U and HC. Conversely, the LS's function that retrieves the index of a sweetword, `indexOfHw`, is *private*, to exclude the possibility of the LS sending a random guess to the HC, as we are not interested in that attack.

Each instance of U is initialized along with an instance of HC, with a shared $seed$; this represents that both parts generate the same OTP at the beginning of a round. The HC knows as well the index of the password in the password file. To allow LS to attempt attacks using the knowledge gained during the run of the protocol, we model the fact that HC can be queried after the end of its role (somehow simulating an open session).

The LS is almost as in the original protocol, except that this time it receives a blinded password parametrized by the OTP, instead of a plain password. An index is a term determined by a blinded word and the row of sweetwords where it is searched. We represent it in ProVerif as

$$indexOfHw\big(\, blindWord(w, getOTP(n)),\; shuffleNblind(u, n)\,\big)$$

where $blindWord$ is the blinded value of w calculated with the seed n, and $shuffleNblind$ is the sweetwords' row for user u shuffled and blinded with seed n.

Our equational theory relies on the $checkEqual$ function in the HC, which returns `TRUE` only when all the parameters in the indexes under comparison are equal. After a successful match, the index hold by the HC is affected by the next seed value, becoming

$$indexOfHw\,\big(\, blindWord(w, getOTP(\textbf{next(n)})),\; shuffleNblind(u, \textbf{next(n)})\,\big).$$

Hence, after this point the evaluation of $checkEqual$ will be $false$ for any submitted index not obtained

with the new seed.

We define a property analogous to the one we verified for the original Honeywords System:

$$correctIndex(u,j) \implies inj-event\big(\, usrLogged(u,p)\,\big) \;\&\&\; inj-event\big(\, indexFound(j, blindWord(p,x), shuffleNblind(u,y))\,\big).$$

The expression states that whenever an index j is equal to the one in the HC's database for u, then (a) a user logged in with username u and password p, and (b) j corresponds to the index of the blinded value of p in the sweetwords row for u. The conjunction ensures the execution of every step in the protocol; the injectivity ensures that each is executed only once.

In addition, we introduce the property $event(unreachable)$ to verify that LS cannot retrieve a sweetword's index of a word not submitted by a user; the event $unreachable$ is triggered if the HC's check function returns $true$ after shuffling and rehashing, when applied to a previously submitted hashed password.

The model also assumes, as we stated in Section 5.5, that the shuffling, blinding and updating instructions are executed *atomically* before sending the response to the LS. Failing to implement HC in this way, leads to an attack that we explain next in the results, which proves that atomicity must in fact be a requirement.

Results. ProVerif confirmed the properties to be `TRUE` in negligible time, from which we conclude that, even knowing that a certain $blindWord(p, getOTP(n))$ is a valid password, LS cannot use it to anticipate the new correct index, since it depends on the seed value possessed only by U and HC.

The analysis also proves that event $unreachable$ is actually unreachable, which implies that LS cannot get any advantage from using HC as an oracle, in combination with messages obtained from previous runs with U and HC.

We also verified the need for the atomic execution of the Shuffle&Blind and the index updating instructions. For instance, sending the response to the LS once the new u's row is obtained but before updating u's index, gives place to the following attack: let HC_1 and HC_2 be concurrent runs of the HC; then,

1. LS sends (u,i) to be verified
2. HC_1 verifies that i is correct, computes the new row, and sends the answer to LS
3. LS submits again i, knowing that it is a correct index
4. HC_2 validates that i is correct—since HC_1 has not updated it—then, computes the new row, updates and grants access
5. HC_1 continues its execution and grants access as well.

5.7 Concluding Remarks and Further Directions

Summing up, in this chapter we proposed a protocol that relies on the use of one-time-passwords, to improve the security of the original Honeywords System, when we consider the component that processes login credentials to be compromised. We provided a formal model of a non-standard adversary, who can steal a password file and run offline dictionary attacks, however, cannot modify the code of the authentication protocol in a way that its behavior changes from the perspective of the other components in the system.

Implementations of the proposed protocol $\mathcal{P}$ and of the original Honeywords System are reported in our original publication [84], where we compare the performance of $\mathcal{P}$ with that of the original

system, concluding that the time complexity in both is linear with respect to the number of sweetwords, and that considering its adoption would be reasonable.

A debatable aspect is that, although the OTP device here is used as a pseudo-random generator and not as a proof of possession, both ideas are somehow tied, as an adversary who manages to get the device will be able to generate the next seeds used for blinding. Besides, from the user's perspective, the approach appears just as a regular two-factor authentication. This observation highlights the difficulty to find implementable and effective approaches that fulfill simultaneously requirements R1, R2 and R3.

An alternative idea worth exploring looks into homomorphic encryption, a relatively recent research area based on public key cryptography, which allows untrusted parties to perform operations on encrypted data [86]. Quite broadly, the intuitive idea is for the User Interface (U) and the Honeychecker (HC) to share a key for encryption and decryption. Then, the U sends the encrypted password to Login Server (LS), who searches for it in an encrypted row of encrypted sweetwords, retrieving an index in encrypted form too. LS is unable to decrypt the index that it sends to HC, but the latter can decrypt it and send the answer to LS. Note that some re-encryption of the passwords would still be needed, so that LS cannot reuse a stored valid password. Feasibility of the implementation of these schemes is however an aspect to take into consideration.

We also remark that code corruption might be interpreted as the LS being turned into an active man-in-the-middle (MITM), slightly restricted in the way in which it can modify messages. This suggests that authentication mechanisms between the U and the HC would be a solution for circumventing this adversarial model. An idea based on the use of password-authenticated key exchange protocols is explored in [22]. In the second part of this thesis, we research further on authentication techniques that could also be considered as the basis of alternative solutions.

Finally, we consider code corruption to be a strong adversary, whose implementation in real life might require substantially more effort than just stealing a password file. Still, the theoretical setting posses an interesting problem and evidences the difficulty of finding adequate and efficient cryptographic tools to deal with such an adversary. Furthermore, the formal verification performed in this chapter served as a kickoff into the formal analysis of security protocols, which we will develop further in the next chapter.

We will move now into protocols that deal with the security of the communication itself rather than the security of the environment.

SECURITY ANALYSIS OF p≡p PROTOCOLS

6

In Chapter 2, we discussed about the importance of encryption and authentication for achieving private secure communications in hostile environments (such as Internet). PGP was the earliest widely adopted solution bringing those security properties to email. However, its design targets a specific audience, assumed to understand the concept of public-key cryptography.

Attempting to reach a wider audience, recent secure email solutions tend to automate tasks that would require average users to be familiar (at least) with basic cryptographic concepts. Such is the case of ProtonMail [148] and Tutanota [178], which offer privacy and strong security guarantees coupled with "easy" usage. However, a drawback of those solutions is that they work optimally in closed environments—i.e., between clients of the same service—but when they interact with other solutions, the security guarantees are in general not preserved. Actually, interoperability has been identified as one of the principal open problems for secure email [50].

In addition to providing end-to-end encryption (E2EE), the p≡p software introduced in Section 2.8 offers to tackle usability and interoperability issues. Given its combination of security and usability features, p≡p appears to be a serious private email solution with possibilities for a broader adoption. Nevertheless, this solution lacks from a rigorous study and proofs of security of the underlying protocols, which is the problem that we address here.

In this chapter, we present a symbolic formal analysis of the two core security protocols implemented in p≡p, that lead to the achievement of authenticated private communication. We start by defining the p≡p key distribution and the trust establishment protocols (Section 6.1), along with the security properties that concern our analysis (Section 6.2). Then, in Section 6.3 we go into the details of the formalization in the applied pi calculus. The results of the protocol verification with ProVerif are reported and further analyzed in Section 6.3.4. We also discuss limitations of the analysis in Section 6.3.5, and research directions therefrom.

6.1 Key distribution and Authentication Protocols of p≡p

The abstract protocols that we introduce in this section were created according to the approach presented in Chapter 3. We used the last version of pEpEngine available at the moment[1] (see page 18), downloaded from the open source code repository of p≡p [144], and online user manuals [146]. These models, presented in [184] as Message Sequence Charts (MSC), constitute the first detailed technical documentation of the key distribution and authentication protocols in p≡p. Some IETF internet drafts [31, 32, 123] released later on, have been useful to reinforce our models.

Remark that the analysis concerns the logic of the mentioned protocols, hence, the messages that we model are exchanges occurring in cryptographic protocols rather than email exchanges. For an overview of email protocols, readers can review Chapter 2.

In the rest of the chapter, we will use sk_x and pk_x to refer to secret and public keys owned by agent x, respectively. We will also use $\mathcal{A}$ and $\mathcal{B}$ to refer to honest participants and $\mathcal{E}$ for the malicious agent trying to prevent the honest parties from achieving the security goals. We denote the p≡p

[1] https://pep.foundation/dev/repos/pEpEngine/file/b7f6df848795 (date:24-02-2018)

instances running in $\mathcal{A}$'s and $\mathcal{B}$'s devices as $\mathtt{pEp}_\mathtt{A}$ and $\mathtt{pEp}_\mathtt{B}$ respectively.

We will also refer to the privacy rating assigned by p≡p per message and sender. These ratings are detailed in Section 2.8.4 of Chapter 2, where we introduced p≡p.

Setting. Let $\mathcal{A}$ and $\mathcal{B}$ be two communication partners that do not share any cryptographic information about each other—e.g. public keys. $\mathcal{A}$ installs p≡p from scratch in a device free of any cryptographic material. $\mathcal{B}$ is already a p≡p user owning a pair of keys (sk_B, pk_B). First, we describe the protocol that allows $\mathcal{A}$ to establish end-to-end encrypted communication with $\mathcal{B}$, and afterwards the protocol for mutual authentication.

6.1.1 Public key Distribution and Encrypted Communication

So that the key distribution protocol (Fig. 6.1) can take place, when p≡p is installed, $\mathtt{pEp}_\mathtt{A}$ generates a pair of keys (sk_A, pk_A) for $\mathcal{A}$ (step 1). The protocol starts when $\mathcal{A}$ sends a message m to $\mathcal{B}$; $\mathtt{pEp}_\mathtt{A}$ creates an identity for $\mathcal{B}$ (2) and stores his contact details (3); then, $\mathtt{pEp}_\mathtt{A}$ sends m in cleartext along with pk_A (4). At reception, $\mathtt{pEp}_\mathtt{B}$ displays m to $\mathcal{B}$ with the privacy rating UNSECURE (5); additionally, $\mathtt{pEp}_\mathtt{B}$ creates an identity for $\mathcal{A}$ (6) and stores her email address and pk_A (7); finally, following the trust-on-first-use (TOFU) approach, $\mathtt{pEp}_\mathtt{B}$ assigns the privacy rating SECURE to $\mathcal{A}$'s identity (8).

When $\mathcal{B}$ replies to $\mathcal{A}$, $\mathtt{pEp}_\mathtt{B}$ attaches pk_B to his response $resp$; this message is then signed with $\mathcal{B}$'s secret key sk_B (9) and encrypted using pk_A (10). The signed and encrypted message is sent to $\mathcal{A}$ (11), and $\mathtt{pEp}_\mathtt{B}$ shows to $\mathcal{B}$ that this message is sent SECURE. At reception, $\mathtt{pEp}_\mathtt{A}$ decrypts $\mathcal{B}$'s message using sk_A (12); then, again by TOFU, it stores pk_B as the public key of $\mathcal{B}$ (13) and assigns to his identity the SECURE rating (14). $\mathcal{B}$'s response is finally shown as SECURE to $\mathcal{A}$. From this point, all the messages between $\mathcal{A}$ and $\mathcal{B}$ are sent encrypted and signed with the stored keys.

Note that the identifiers created for $\mathcal{A}$ in $\mathtt{pEp}_\mathtt{A}$ and $\mathtt{pEp}_\mathtt{B}$—i.e., idA_A and idA_B respectively— are internal and independent for each instance, although they refer to the same entity. The same holds for $\mathcal{B}$. Also, pk_A and pk_B sent in steps (4) and (11) are only attached to the first communication between $\mathcal{A}$ and $\mathcal{B}$ or whenever they are updated.

The key distribution protocol allows $\mathcal{A}$ to send private messages to the owner of pk_B, however, it does not guarantee that the owner is $\mathcal{B}$. Man-in-the-middle (MITM) attacks are possible, as we show in Section 6.3.4. To establish a secure communication, $\mathcal{A}$ and $\mathcal{B}$ still need proof that they are actually communicating with the parties they intend.

6.1.2 Authentication and p≡p Privacy Rating Assignment

Trust is established once $\mathcal{A}$ and $\mathcal{B}$ have verified that the public keys previously distributed are owned by the intended peers. In p≡p, this mutual authentication is achieved via a so called Handshake (Fig. 6.2), which consists in $\mathcal{A}$ and $\mathcal{B}$ comparing a list of trustwords through an assumed to be secure out-of-band (OOB) channel, which is needed only once.

When $\mathcal{A}$ selects the option to perform a handshake with $\mathcal{B}$ (1), $\mathtt{pEp}_\mathtt{A}$ generates a combined fingerprint based on applying an XOR function to the fingerprints of $\mathcal{A}$ and $\mathcal{B}$ (2). The resulting hexadecimal string is mapped onto words in the selected language from the trustwords database (3) and displayed to $\mathcal{A}$ (4). The analogous actions occur in $\mathtt{pEp}_\mathtt{B}$ when $\mathcal{B}$ selects the handshake option. Given that the trustwords database is the same in all p≡p distributions, if $\mathtt{pEp}_\mathtt{A}$ and $\mathtt{pEp}_\mathtt{B}$ use the same input parameters, i.e., the same public keys and thus the same fingerprints, the list of trustwords generated by each p≡p instance must be the same (we revisit this claim in Section 6.4).

The next step is the authentication, where $\mathcal{A}$ and $\mathcal{B}$ contact each other in a way that they are

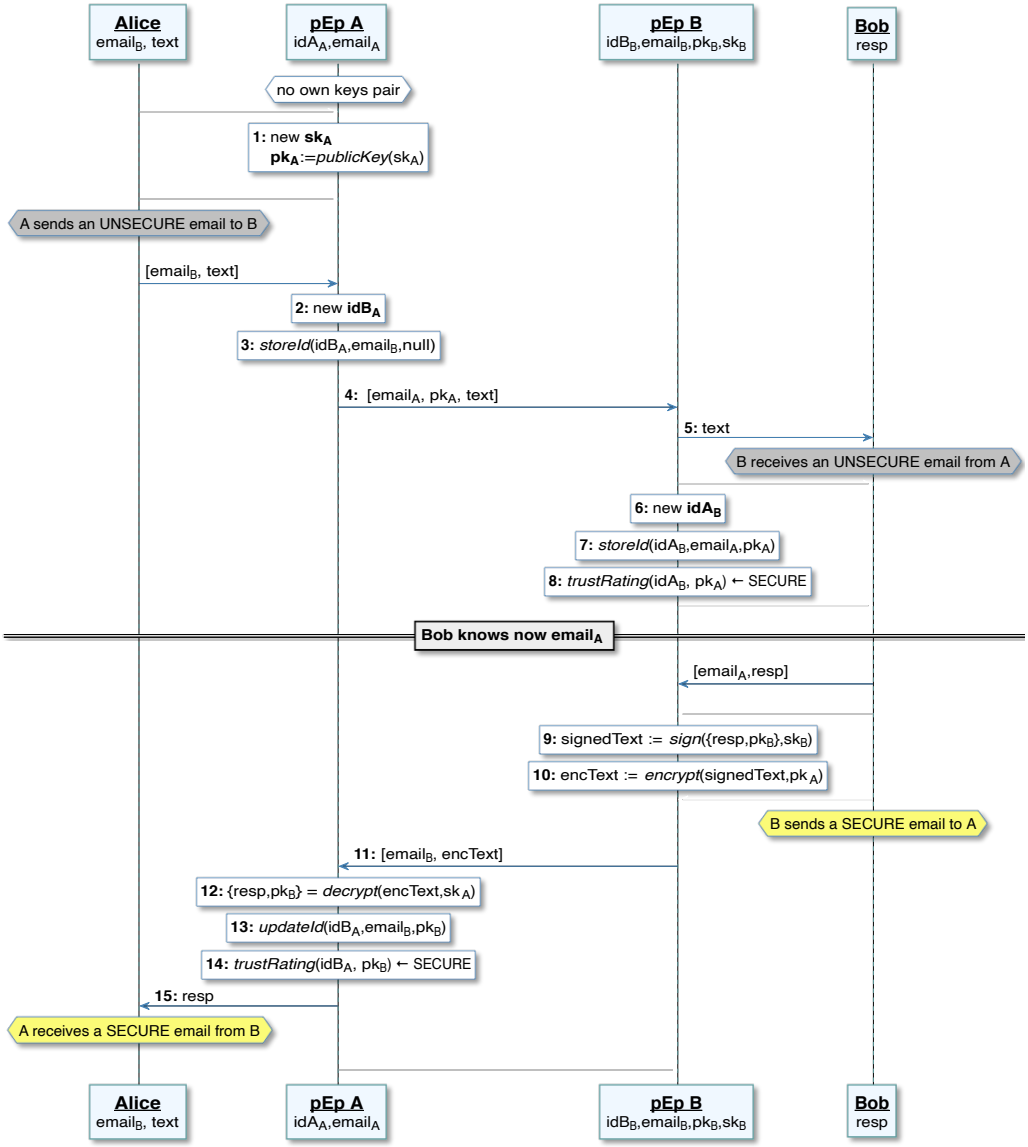

Figure 6.1: p≡p Key Distribution Protocol. The initial privacy assignment is based on TOFU.

sure to be talking with the real person—e.g., they meet in person—and compare the list of trustwords displayed for each (5). If $\mathrm{pEp_B}$ receives a confirmation of trustwords from $\mathcal{B}$, $\mathcal{A}$'s privacy rating is set to TRUSTED (6); we call this case a *successful handshake*. Conversely, in an *unsuccessful handshake* $\mathcal{B}$ rejects the words and $\mathcal{A}$'s rating is downgraded from SECURE to MISTRUSTED (7). The analogous occurs in $\mathcal{A}$'s device with respect to $\mathcal{B}$.

The privacy rating assigned after a handshake remains for all future exchanges with the communication partner. Thus, after a successful handshake, messages between the identities involved are always sent encrypted and signed, therefore making the communication private and authenticated.

Remark that p≡p does not force users to perform a handshake. The email messages are always sent regardless of the security level, which is decided per message and per recipient according to the

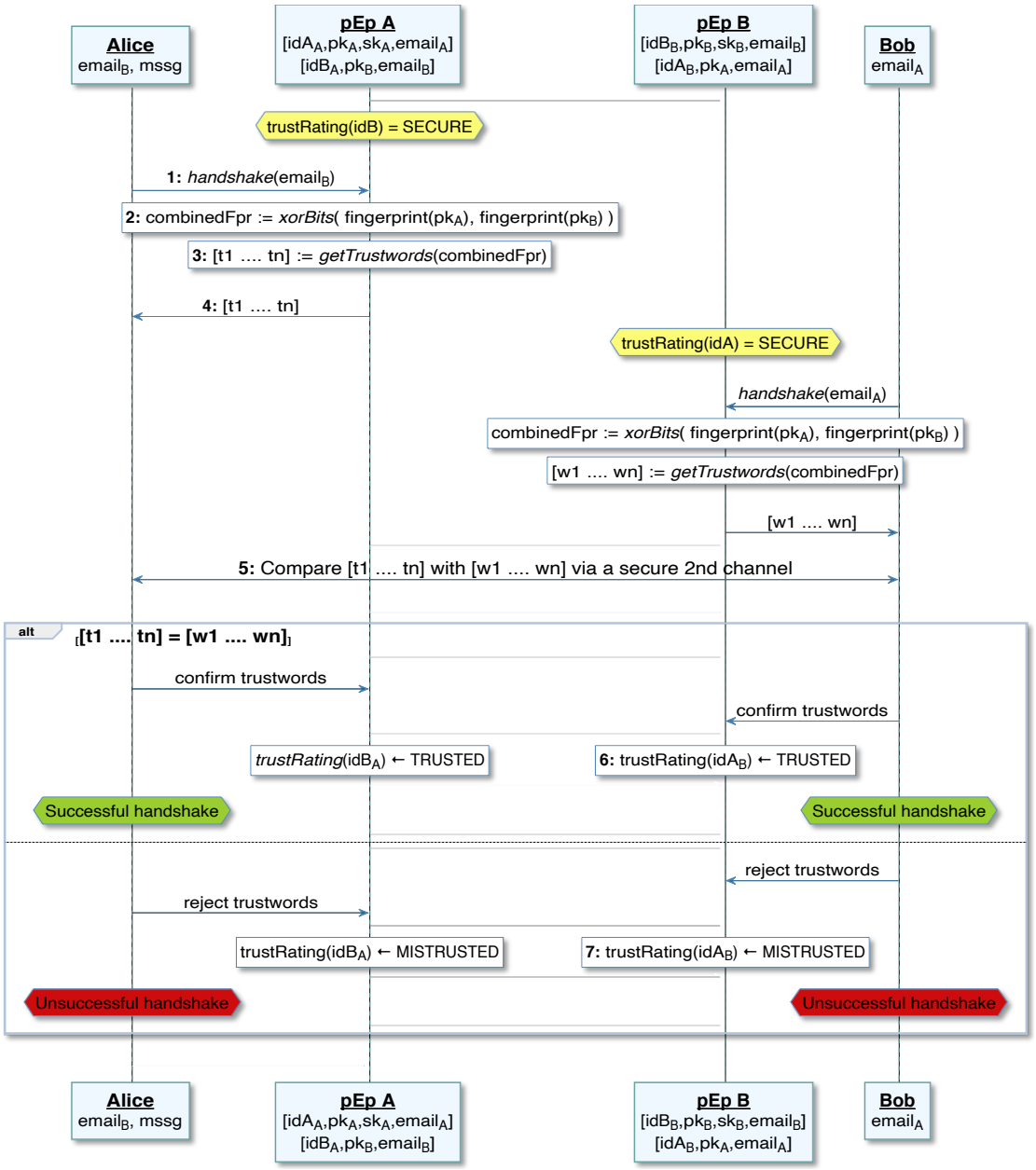

Figure 6.2: p≡p Handshake for mutual authentication

recipient's data available.

Note also that the handshake is essentially a human protocol, as the actions taken by a p≡p instance depend completely on the user's input. Models of security protocols that consider also the humans and their interaction with the system are known as *security ceremonies* [73]. We will expand on this topic in the next chapter.

6.2 Security Properties

Our requirements for authentication match the definition of *full agreement* given by Lowe in [117]. This definition subsumes aliveness, weak agreement, non-injective agreement and injective agreement as defined in the same reference. Broadly, full agreement requires that at the end of a protocol run, all the essential data known by the two participants coincides; in the case that concerns us, this data would be the public keys and the email addresses.

Definition 6.2.1 (Full agreement, from [117])**.** A protocol guarantees to an initiator A *full agreement* with a responder B on a set of data items ds if, whenever A completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the terms in ds, and each such run of A corresponds to a unique run of B. Additionally, ds contains all the atomic data items used in the protocol run.

Here we redefine this property in the context of $\mathsf{p} \equiv \mathsf{p}$, and introduce informally other security properties which also refer to the correctness of the trust level assignment.

Property 1 (Full agreement)**.** A full agreement between $\mathcal{A}$ and $\mathcal{B}$ holds on the data set conformed by pk_A, pk_B, $email_A$ and $email_B$, if, whenever $\mathcal{A}$ completes a successful handshake with $\mathcal{B}$, then: $\mathcal{B}$ has previously been running the protocol with $\mathcal{A}$, the identity data of $\mathcal{A}$ is $(email_A, pk_A)$, and the identity data of $\mathcal{B}$ is $(email_B, pk_B)$. In addition, by the definition of a successful handshake, $\mathcal{A}$'s and $\mathcal{B}$'s trustwords are equal.

Property 2 (Trust-by-handshake)**.** Trust-by-handshake holds for $\mathcal{B}$ if whenever $\mathcal{B}$ receives a message with privacy rating TRUSTED from $\mathcal{A}$, then previously $\mathcal{B}$ executed a successful handhsake with $\mathcal{A}$.

Property 3 (Privacy-from-trusted)**.** Privacy-from-trusted holds for $\mathcal{B}$ if, whenever $\mathcal{B}$ receives a message m with a privacy rating TRUSTED from $\mathcal{A}$, then $\mathcal{A}$ sent m to $\mathcal{B}$ and m is encrypted with $\mathcal{B}$'s public key.

Property 4 (Integrity-from-trusted)**.** Integrity-from-trusted holds for $\mathcal{B}$ if, whenever $\mathcal{B}$ receives a message m with a privacy rating TRUSTED from$\mathcal{A}$, then $\mathcal{A}$ sent m to $\mathcal{B}$ and m is signed with a valid signature of $\mathcal{A}$.

Note that "trust-by-handshake" guarantees all messages that $\mathcal{B}$ receives as TRUSTED to come from a peer with whom $\mathcal{B}$ has successfully run the protocol; hence, when this property holds, the definitions of "privacy-from-trusted" and "integrity-from-trusted" imply that $\mathcal{A}$ is rated as TRUSTED too.

Property 5 (MITM-detection)**.** MITM-detection holds if whenever an unsuccessful handshake between $\mathcal{A}$ and $\mathcal{B}$ occurs, then $\mathcal{A}$ had previously registered a key for $\mathcal{B}$ that does not belong to him, vice versa, or both.

Property 6 (Confidentiality)**.** Confidentiality holds if $\mathcal{E}$ cannot learn the content of any message intended to be private between $\mathcal{A}$ and $\mathcal{B}$. This property refers to the notion of syntactic secrecy introduced in Chapter 4.

6.3 Formal Security Analysis

Let us now formalize in the applied pi calculus all the elements. The source code for ProVerif of the full model that we discuss here, is included in Appendix A.2.

6.3.1 Threat Model

We start by assuming that the participants have a legitimate and correct distribution of the p≡p software (free of implementation flaws). Now, to determine a relevant attacker model we need to reflect about the architecture of p≡p.

To an attacker with access to the user's device, not only the code but also the application databases and the keys repository are available. $\mathcal{E}$ can thus have $\mathcal{B}$ trusting her by simply modifying the corresponding record in the privacy ratings database, even if a handshake was never performed. Modifications to the trustwords database would also result in an attack, which although not threatening privacy, could prevent $\mathcal{A}$ and $\mathcal{B}$ from establishing a valid trusted communication as TRUSTED. Therefore, we restrict the threat model with the following assumptions:

1. p≡p users are honest participants holding a legitimate p≡p installation in a secure device
2. The adversary cannot modify exchanges over the trustwords channel
3. The adversary has complete control over the network used to exchange emails (Dolev-Yao)

These assumptions allow $\mathcal{E}$ to eavesdrop, remove, and modify emails exchanged between $\mathcal{A}$ and $\mathcal{B}$, as well as to send them self-created messages; this includes learning their public keys exchanged by email. $\mathcal{E}$ cannot however interfere with the channel used to corroborate trustwords. Recall that this is a secondary channel (such as the phone or in-person) intended to be used only once and not to replace the email communication channel.

6.3.2 Modeling the p≡p Protocol

We will refer to the sequential execution of the key distribution protocol and the handshake, from Section 6.1, as the p≡p protocol.

An identity in the p≡p system is defined by a user identifier associated to an (email) address; each identity has a key pair associated [32]. Actually, each identity can have a set of key pairs spread across different devices, from which one is selected as the default key. In this model we disregard details of the key management implementation, as they are not relevant for the analysis; in particular, we assume that a user who configures p≡p in multiple devices with the same email address—and thus, same identity—uses by default the same key pair in all of them. A formal study of the p≡p protocol in charge of securely synchronizing secret information among devices is still ongoing work.

Then, we represent an email address as an identity of type `userId`, and model the identity as a function of the secret key, $\mathtt{id}(\mathtt{skey}) : \mathtt{userId}$. The roles $\mathcal{A}$ and $\mathcal{B}$ are represented by two processes: `senderA` and `receiverB`. To communicate with $\mathcal{B}$, $\mathcal{A}$ needs to know $\mathcal{B}$'s email, i.e., the identity $\mathtt{id}(\mathtt{sk_B})$; in turn, $\mathcal{B}$ only needs to know the own id and secret key. The actions for each participant model those in the diagrams in Figures 6.1 and 6.2. We run multiple instances of $\mathcal{A}$ as well as of $\mathcal{B}$, to simulate communication with multiple peers.

For the exchange of emails we use a public channel; on the contrary, a private channel models the trustwords' validation channel. In order to prove confidentiality of encrypted and authenticated communication, we introduce a private message $mssg$ representing a message whose content is initially unknown to the adversary $\mathcal{E}$; then, we model $\mathcal{A}$ sending $mssg$ to $\mathcal{B}$ via the public channel after a successful handshake between them. Since $\mathcal{B}$ is trusted, $mssg$ is sent signed and encrypted, and thus, expected to remain unreadable by $\mathcal{E}$ at the end of the protocol.

In agreement with the symbolic model assumption, our equational theory models a perfect behavior of asymmetric encryption and digital signatures. These equations capture the relationships

allowed among the cryptographic primitives involved, determining the ways in which any participant, the attacker included, can reduce terms. Then, for M a message and SK a secret key:

$$adec(aenc(M, pubKey(SK)),\ SK) = M \tag{1}$$
$$verifSign(sign(M, SK),\ pubKey(SK)) = M \tag{2}$$
$$getMssg(sign(M, SK)) = M \tag{3}$$

Equation (1) expresses that a message M encrypted with a certain public key can be decrypted with the corresponding secret key; moreover, this is the only way to obtain M from a ciphertext since there is no other equation involving the $aenc$ primitive. Analogously, equation (2) returns M only if it was signed with the secret key associated to the public key used for the verification. Equation (3) allows the recovery of a message without verification of a digital signature and we introduce it to model the capability of $\mathcal{E}$ for learning messages without the need of verifying the signature.

Additionally, we assume that users execute the comparison of trustwords correctly, i.e., they confirm the trustwords in the system only when they match in the real world and they reject them only in the contrary case; by modeling this, we implicitly model correctness of the trustwords generation function. Since a PGP fingerprint is uniquely derived from a public key, we abstract fingerprints simply as public keys. Then, for two public keys PK_1, PK_2, two trustwords lists W_1, W_2 and the trustwords generation function `trustwords`:

$$trustwordsMatch(trustwords(PK_1, PK_2), trustwords(PK_1, PK_2)) = true$$
$$trustwordsMatch(trustwords(PK_1, PK_2), trustwords(PK_2, PK_1)) = true$$
$$otherwise\ trustwordsMatch(W_1, W_2) = false.$$

During its computations, $\mathcal{E}$ is allowed to apply all and only these primitives. Furthermore, $\mathcal{E}$ has access to all the messages exchanged via the public channels and to any information declared as public. These rules allow for instance to capture $\mathcal{E}$'s real-life capability of generating the trustwords, which is possible because all the elements are public knowledge: the source code of the function, the trustwords database, $\mathcal{B}$'s public key and $\mathcal{A}$'s public key.

6.3.3 Privacy and Authentication Properties of p≡p

We formalize the properties introduced in Section 6.2, as correspondence assertions and reachability properties (review these concepts on page 37). The formalization is given in terms of the events defined in Table 6.1, where s and r represent two p≡p users.

Then, for a message m, and for all p≡p users a and b, and public keys pk_A, pk_B:

Full Agreement.

$$endHandshakeOk(a, b, pk_A, pk_B) \implies startHandshake(a,b) \wedge startHandshake(b,a)$$
$$\wedge\ userKey(a, pk_A)\ \wedge\ userKey(b, pk_B)$$
$$\wedge\ confirmTrust(a, b)\ \wedge\ confirmTrust(b, a)$$

This is the formalization of Property 1. As explained in 6.3.2, in our model the email address is abstracted as the identity itself, hence, verifying that an email address is associated to an identity (e.g. $email_A$ belongs to a) is implicit. In contrast, we explicitly add the condition for a handshake to be successful, which implies agreement on the trustwords.

EVENT	DESCRIPTION
`attacker`(m)	the adversary knows the content of the message m
`decryptionFails`(r,s,m)	r cannot decrypt a message m from a trusted peer s
`endHandshakeUnsucc` (s,r,pk_s,pk_r)	s and r completed an unsuccessful handshake with the public keys pk_s and pk_r respectively
`endHandshakeOk` (s,r,pk_s,pk_r)	s and r completed a successful handshake with the public keys pk_s and pk_r, respectively
`receiveGreen`(r,s,m)	r received the message m from s as TRUSTED
`confirmTrust`(r,s)	the peer r sets the privacy rating of s as TRUSTED after verifying via a secure channel that their trustwords match
`sendGreen`(s,r,m)	s sent the message m to r as TRUSTED
`signVerifFails`(r,s,m)	r cannot verify the signature attached to m as a valid signature of s
`startHandshake`(s,r)	s starts a handshake via a second-channel with r
`userKey`(s,pk_s)	the agent s is the owner of the key pk_S

Table 6.1: Events in the model of the p≡p protocol

Trust-by-handshake. $receiveGreen(b,a,m) \implies confirmTrust(b,a)$

This formula matches exactly the definition of Property 2.

Privacy-from-trusted. For messages z and y,

$$\begin{aligned}\big(receiveGreen(b,a,z) \quad &\implies \quad sendGreen(a,b,z) \ \wedge \ z = aenc(y,pk_B)\\ &\quad\ \wedge userKey(b,pk_B)\big) \quad \wedge\\ \big(decryptionFails(b,a,m) \quad &\implies \quad \neg sendGreen(a,b,m)\big)\end{aligned}$$

This formula is the conjunction of two correspondence assertions. The first one expresses Property 3; the second correspondence enforces the first by saying that it cannot be otherwise, i.e., when b receives a message m from a which for any reason cannot be decrypted—e.g. m is not encrypted—then a did not send a TRUSTED message m to b.

Integrity-from-trusted. For a message z, a secret key sk_A, and a public key kb,

$$\begin{aligned}\big(receiveGreen(b,a,z) \quad &\implies \quad sendGreen(a,b,z) \ \wedge z = aenc(sign(m,sk_A),kb)\\ &\quad\ \wedge userKey(a,sk_A)\big) \quad \wedge\\ \big(signVerifFails(b,a,m) \quad &\implies \quad \neg sendGreen(a,b,m)\big)\end{aligned}$$

Analogous to the previous formula, in this one we express Property 4 and reinforce it by proving that whenever the verification of the signature fails in message m, then a did not send m.

MITM-detection. For public keys ka and kb,

$$\begin{aligned}endHandshakeUnsucc(a,b,ka,kb) \quad \implies \quad &(userKey(a,pk_A) \ \wedge \ pk_A \neq ka) \ \vee\\ &(userKey(b,pk_B) \ \wedge \ pk_B \neq kb)\end{aligned}$$

This formula matches exactly the definition of Property 5.

Confidentiality. `attacker(x)` is a built in predicate in ProVerif, which evaluates to `TRUE` if by applying the derivation rules to the knowledge of the adversary, there exists a derivation that results in `x`. Therefore, the protocol achieves confidentiality if, for a private message $mssg$,

$$\neg attacker(mssg)$$

6.3.4 Verification Results and Analysis

The verification was executed using ProVerif 2.0 on a standard PC (Intel i7 2.7GHz, 8GB RAM), and the response time was immediate. We analyzed three different models: the key distribution protocol, the handshake protocol, and the full trust establishment protocol (the p≡p protocol).

The results confirmed the vulnerability of the key distribution protocol to MITM attacks. Inherent to public key cryptography, the weakness resides in the exchange of keys via a channel where $\mathcal{E}$ has complete access. An attack proceeds as follows: $\mathcal{E}$ intercepts the initial message from $\mathcal{A}$ to $\mathcal{B}$ and sends instead a new message replacing pk_A with $\mathcal{E}$'s own public key, pk_E; thus, pEp_B stores pk_E as the key associated to $\mathcal{A}$'s email (step (7) of Fig. 6.1). When $\mathcal{B}$ replies, the response is encrypted with pk_E (step (10)), and thus $\mathcal{E}$ can intercept and decrypt it with the corresponding secret key, obtaining in addition pk_B attached. From this point, $\mathcal{E}$ can send encrypted emails to $\mathcal{B}$ using $\mathcal{A}$'s email address as the sender, and decrypt the responses sent by $\mathcal{B}$. In an analogous way, $\mathcal{E}$ can have $\mathcal{A}$ associating pk_E to $\mathcal{B}$'s identity, to gain control on both sides of the communication.

Regarding the handshake, encryption and authentication hold since the trustwords comparison never mismatches due to the assumptions of the peer devices being secure and of a previous key distribution successfully executed.

Finally, the analysis of the p≡p protocol, where the key distribution is followed by a handshake, determined that the six properties: full agreement, trust-by-handshake, privacy-from-trusted, integrity-from-trusted, MITM-detection and confidentiality, are satisfied.

We conclude that the execution of the p≡p protocol fulfills the claimed security goals, i.e., after a successful handshake there is no undetectable way for $\mathcal{E}$ to modify the exchanges between $\mathcal{A}$ and $\mathcal{B}$, given that every message between them is always sent encrypted and signed with the corresponding keys. Consequently, secrecy, authentication and integrity of the messages are preserved. These results depend on the assumptions of p≡p residing in a secure environment, of the use of a secure OOB channel for the trustwords comparison, and of p≡p identities uniquely associating each email account with a key pair.

Additional findings. A side observation refers to unsuccessful handshakes; in particular to the case where $\mathcal{A}$ has the correct public key of $\mathcal{B}$, but $\mathcal{B}$ has registered an incorrect public key for $\mathcal{A}$. A handshake between $\mathcal{A}$ and $\mathcal{B}$ would fail and both partners would reject each other's trustwords.

In the version of pEpEngine that we analyzed, by design a MISTRUSTED privacy rating could not be reverted. This situation presumably represented an issue, since in any handshake with the correct keys executed after a failed attempt, $\mathcal{A}$ was not able to trust $\mathcal{B}$'s key as she had mistrusted this identity—the tuple $(\mathcal{B}, email_B, pk_B)$—before.

However, $\mathcal{E}$ misleading $\mathcal{A}$ to mistrust the intended partner resembles a Denial of Service (DoS) attack but does not represent a threat to privacy. In any case, we discussed the issue with p≡p developers and in recent versions they have implemented a mechanism that allows reverting this action.

6.3.5 Limitations

The analysis focuses solely on the technical specification of the key distribution and handshake protocols; social attacks such as impersonation or phishing are not detected. For instance, $\mathcal{E}$ can create a fake email account that appears to be from $\mathcal{A}$ and use it to send $\mathcal{B}$ an email with $\mathcal{E}$'s public key and contact details for the handshake; if $\mathcal{B}$ has never met $\mathcal{A}$, a trustwords comparison with $\mathcal{E}$ would succeed given that $\mathcal{A}$ and $\mathcal{E}$ are indeed executing the protocol; however, $\mathcal{B}$ thinks to be

interacting with $\mathcal{A}$. This attack is not spotted in our verification due to the secure OOB channel assumption, by which $\mathcal{A}$ and $\mathcal{B}$ always contact the correct real entity.

The assumption of perfect cryptography implies that we consider the libraries implementing cryptographic operations to be correct, we did not verify such implementations. Side-channel attacks are not considered either.

6.4 The Trustwords Generation Function

Following the black box assumption in the symbolic approach, we modeled a correct function that generates the trustwords in each p≡p client. This function is however central for the handshake since peer-to-peer authentication is based on the trustwords shown to the user; therefore, we consider necessary a proof of correctness and security.

Algorithm 3, from the up-to-date source code version of pEpEngine [145], shows the trustwords function in pseudocode, where $\texttt{getFingerprint}(\cdot)$ computes a cryptographic hash $hFpr(\cdot)$ (usually expressed as hexadecimal) of the public key given as input parameter; and $\texttt{trustwordOf}(\cdot)$ is a deterministic one-to-one mapping from a hexadecimal 4-digit value into a word in natural language (see Section 2.8.3 for details).

In fact, the original algorithm takes as input two p≡p identities, from which it extracts the associated key. In order to simplify the proof of correctness, here we pass already the public keys as initial parameters to Algorithm 3. Afterwards, we discuss security vulnerabilities of the original version, which however do not affect our correctness claims below.

Security and correctness. We want to assert that (1) if two p≡p instances, $\texttt{pEp}_\texttt{A}$ and $\texttt{pEp}_\texttt{B}$, carry out a trustwords generation process with the same public keys as input, then they output the same list of words. Furthermore, (2) two identical lists of words should only be generated when the public keys with which $\texttt{pEp}_\texttt{A}$ runs the algorithm, are the same as the input parameters used by $\texttt{pEp}_\texttt{B}$.

For formalizing (2) we need to consider the property of the XOR operation by which a bitstring xored with itself is equal to 0. We elaborate on this point in the proof. Then, we formalize (1) and (2) as Claim 1 and Claim 2 respectively, and prove them considering the adversarial model described in Section 6.3.1. Let:

- $\texttt{pEp}_\texttt{A}$ and $\texttt{pEp}_\texttt{B}$ be p≡p instances installed in secure devices;
- pk_A and pk_B be distinct public keys;
- $\texttt{pEp}_\texttt{A}$ store pk_A as the self key and store pk_B as the key of an identity obtained via a channel controlled by the adversary;
- $\texttt{pEp}_\texttt{B}$ store pk_B as the self key and store pk_A as the key of an identity obtained via a channel controlled by the adversary;
- $\texttt{trustwords}_\texttt{A}$ and $\texttt{trustwords}_\texttt{B}$ be the lists of trustwords returned respectively by the execution of GET_TRUSTWORDS in $\texttt{pEp}_\texttt{A}$ and $\texttt{pEp}_\texttt{B}$;
- $\texttt{getTrustwords}_\texttt{A}(\texttt{pk}_\texttt{B},\texttt{pk}_\texttt{A})$ and $\texttt{getTrustwords}_\texttt{B}(\texttt{pk}_\texttt{A},\texttt{pk}_\texttt{B})$ be predicates respectively denoting the events of $\texttt{pEp}_\texttt{A}$ and $\texttt{pEp}_\texttt{B}$ running GET_TRUSTWORDS with the given parameters.

Claim 1.

$$\left(getTrustwords_A(\texttt{pk}_\texttt{B},\texttt{pk}_\texttt{A}) \wedge getTrustwords_B(\texttt{pk}_\texttt{A},\texttt{pk}_\texttt{B})\right) \implies trustwords_A = trustwords_B.$$

Proof. $getTrustwords_A(\texttt{pk}_\texttt{B},\texttt{pk}_\texttt{A})$ implies that $\texttt{pEp}_\texttt{A}$ runs $\text{GET\_TRUSTWORDS}_A(\texttt{pk}_\texttt{B},\texttt{pk}_\texttt{A})$. By assumption, $\texttt{pEp}_\texttt{A}$, $\texttt{pEp}_\texttt{B}$, $\texttt{pk}_\texttt{B}$ and $\texttt{pk}_\texttt{A}$ reside in secure devices, thus, Algorithm 3 is executed following the

Algorithm 3 Trustwords generation

```
1:  procedure GET_TRUSTWORDS(pk1, pk2)
2:      //obtain the fingerprint of the given public key
3:      fpr1 ← getFigerprint(pk1);
4:      fpr2 ← getFigerprint(pk2);
5:      i ← 0;
6:      // an empty string to store hexadecimal digits
7:      xorResult ← ε;
8:      //bitwise XOR of both fingerprints. If the fingerprints have different length,
9:      //a substring of the length of the shortest one is taken from the longest one
10:     while i < length(fpr1) and i < length(fpr2) do
11:         f1i ← character at i of fpr1;
12:         f2i ← character at i of fpr2;
13:         xoredChar ← f1i to binary ⊕ f2i to binary;
14:         xorResult ← xorResult + xoredChar to hex;
15:         INCREMENT i;
16:     if length(fpr1) ≠ length(fpr2) then
17:         xorResult ← xorResult+ the remainder of the longest fingerprint;
18:     k ← 0;
19:     //an empty string to store the words obtained from the mapping
20:     trustwords ← ε;
21:     while k < length(xorResult) do
22:         //obtain the next 4 hexadecimal digits
23:         if k + 4 ≤ length(xorResult) then
24:             hexVal ← substring(k, k + 4, xorResult);
25:         else
26:             hexVal ← substring(k, length(xorResult), xorResult)
27:                         + add '0' to complete a string of size 4;
28:         //retrieve the corresponding word given in the p ≡ p mapping
29:         w ← trustwordOf(hexVal);
30:         trustwords ← trustwords + w+ ' ';
31:         k ← k + 4;
    return trustwords;
```

specifications and with the given parameters.

By instructions 3 and 4, the algorithm obtains the fingerprints of pk_B and pk_A and uses them to compute $xorResult_A = fpr_B \oplus fpr_A$ (line 14). Analogously, $getTrustwords_B(\mathtt{pk_A}, \mathtt{pk_B})$ leads to computing $xorResult_B = fpr_A \oplus fpr_B$. Then,

$$xorResult_A = fpr_B \oplus fpr_A = fpr_A \oplus fpr_B = xorResult_B.$$

From line 16 on, the algorithm operates on $xorResult$ to deterministically compute the string of words to return. Since $xorResult_A = xorResult_B$, then $trustwords_A = trustwords_B$.

■

Remark that Claim 1 only implies the correct trustwords generation when the legitimate algorithm is executed. However, it does not imply mutual authentication between the owners of pk_A and pk_B,

as the public keys stored in the devices are assumed to have been exchanged through an insecure channel. For instance, $\mathtt{pEp_A}$ could represent a device legitimately owned by $\mathcal{E}$ and $pk_A = pk_E$; still, the claim holds. Mutual authentication is achieved only after a successful handshake, which in p≡p requires an OOB comparison.

Also, for readability we overload the $\oplus$ operator and use it on hexadecimal strings (the fingerprints) to denote the bitwise application of the XOR over the given values.

Claim 2.

$$trustwords_A = trustwords_B \implies \big(getTrustwords_A(\mathtt{pk_B}, \mathtt{pk_A}) \wedge getTrustwords_B(\mathtt{pk_A}, \mathtt{pk_B})\big) \vee \big(getTrustwords_A(\mathtt{pk_A}, \mathtt{pk_A}) \wedge getTrustwords_B(\mathtt{pk_B}, \mathtt{pk_B})\big).$$

Proof. By assumption, a genuine instance of p≡p resides in a secure device, therefore, $trustwords_A$ must be the output of $\textsc{get\_trustwords}_A(x, y)$. By construction, y corresponds to the self public key, thus, $y = pk_A$. Now, since it results from the application of $\mathtt{trustwordOf}(\cdot)$, by definition of the latter, the value of $trustwords_A$ is deterministically determined by the value of $xorResult_A = fpr_x \oplus fpr_A$. Likewise, $trustwords_B$ is the output of $\textsc{get\_trustwords}_B(z, pk_B)$ and is deterministically determined by the value of $xorResult_B = fpr_z \oplus fpr_B$. Then,

$trustwords_A = trustwords_B$

$\implies$ $xorResult_A = xorResult_B$, since $\mathtt{trustwordOf}(\cdot)$ is a bijection

$\implies$ $fpr_x \oplus fpr_A = fpr_z \oplus fpr_B$ by lines 10-17

$\implies$ $\mathtt{getFingerprint}(x) \oplus \mathtt{getFingerprint}(pk_A)$
$= \mathtt{getFingerprint}(z) \oplus \mathtt{getFingerprint}(pk_B)$ by lines 3,4

$\implies$ $hFpr(x) \oplus hFpr(pk_A) = hFpr(z) \oplus hFpr(pk_B)$ by def.

Since $hFpr(pk_A)$ and $hFpr(pk_B)$ are fixed respectively for $\mathtt{pEp_A}$ and $\mathtt{pEp_B}$, the equality holds in two cases:

1) $hFpr(x) = hFpr(pk_B)$ and $hFpr(z) = hFpr(pk_A)$
   Since $hFpr(\cdot)$ is a cryptographic hash function, by definition it is one-way and collision-resistant. This implies that there ia a negligible probability of finding $x \neq pk_B$ and $z \neq pk_A$ such that the condition above holds, as any adversarial attempt would have to be the result of a brute-force search over the public key space, which is computationally infeasible. Hence $x = pk_B$ and $z = pk_A$.
   Thus, $\textsc{get\_trustwords}_A(pk_B, pk_A)$ and $\textsc{get\_trustwords}_B(pk_A, pk_B)$ were executed, and therefore, $getTrustwords_A(\mathtt{pk_B}, \mathtt{pk_A}) \wedge getTrustwords_B(\mathtt{pk_A}, \mathtt{pk_B})$ is satisfied.
2) $hFpr(x) = hFpr(pk_A)$ and $hFpr(z) = hFpr(pk_B)$
   This is the case where both sides of the equality are 0, by the definition of XOR. And analogous to case 1, $x = pk_A$ and $z = pk_B$, which substituting the variables, implies that $getTrustwords_A(\mathtt{pk_A}, \mathtt{pk_A})$ and $getTrustwords_B(\mathtt{pk_B}, \mathtt{pk_B})$ are TRUE.

■

This proof shows that there are two ways for the lists of trustwords to be equal. Either if $\mathtt{pEp_A}$ and $\mathtt{pEp_B}$ used the same input parameters (case 1), or if each of them used twice the same public key as

input parameter (case 2). In the latter case, the list of trustwords will consist only of occurrences of the word to which the hexadecimal value 0 is mapped.

Since Algorithm 3 takes the input parameters from data locally stored and the devices are assumed secure, an alternative for the adversary to influence the output of the trustwords generation consists in sending prepared identities in advance, given that these are received via an insecure channel.

For instance, considering $id_A = (email_A, pk_A)$ to be the identity of the owner of $\mathtt{pEp_A}$, and $id_B = (email_B, pk_B)$ the self-identity in $\mathtt{pEp_B}$, $\mathcal{E}$ could trigger case 2 by sending $id_1 = (email_B, pk_A)$ to $\mathtt{pEp_A}$ and $id_2 = (email_A, pk_B)$ to $\mathtt{pEp_B}$. Thus, a handshake executed in $\mathtt{pEp_A}$ with the owner of $email_B$, would fetch pk_A from id_1, and then run GET_TRUSTWORDS(pk_A, pk_A). Similarly, in $\mathtt{pEp_B}$ GET_TRUSTWORDS(pk_B, pk_B) will run.

A handshake executed with such trustwords lists will be successful, however, the communication between the genuine owners of id_A and id_B will not be shown as TRUSTED, since these correct identities were not verified. This is an attack to authentication, given that the wrong keys are trusted; it does not compromise the privacy of the communication, as $\mathcal{E}$ does not learn any key that allow to decrypt messages exchanged.

In Algorithm 3 we omitted an implementation detail by which, unless otherwise specified by the user, only the first five words are retrieved (lines 21-31). This decision increases the chances of an adversary to find a pre-image. We will expand on this topic in Chapter 9.

Finally, we stress the need for the software to ensure that all the distributions contain exactly the same trustwords database, to prevent the occurrence of unsuccessful handshakes derived from inconsistent mappings.

6.5 Concluding Remarks and Further Directions

The analysis presented through this chapter suggests several further research directions beyond the scope of p≡p, whose study would contribute to enrich the areas of formal security analysis and secure email.

One direction concerns the development of theoretical frameworks to enrich the insights gained with a symbolic security analysis.

- As mentioned before, the handshake models a security ceremony, as it considers the interaction of users, capturing the scenario in which the protocol takes place. Our model however considers only the ideal behavior of users. An interesting follow-up refers to extending the model to characterize users with different behaviors. A starting point could be the work by Basin et al. [16], who propose a model of human errors in human-to-machine authentication protocols.
  A related question goes slightly beyond and conjectures whether the perception of a user while executing a security ceremony has repercussions in the overall security of the system. Defining a scheme to reason about such aspects poses another challenge. We explore this further in Chapter 7.
- Given the importance of the trustwords generation algorithm in the protocol, we provided a proof of security. As in this case, a security analysis of the black box functions in a symbolic model might be needed. Since Algorithm 3 is relatively straightforward, we provided a manual proof; however, in the general case we believe that creating a reference implementation of core functions in an appropriate language, such as F* (see Section 2.9), is an interesting option to automate proofs of

security, with the additional advantage that, if flaws are found, this implementation can be refined until the security properties are satisfied; then, F* can generate code in other languages—e.g. Ocaml, C—to replace the faulty original function.

Another direction branches from reconsidering some of our assumptions, which are generally shared with secure email and messaging solutions. The aim is to enhance security and usability aspects of p≡p and secure email solutions.

- We assumed that humans carry out the trustwords comparison faithfully; yet, this might not always be the case. Manual authentication via an OOB channel opens the possibility for users to introduce inaccurate inputs to the system—e.g., mistrusting a trusted peer or vice-versa—which might be the cause of security flaws. A research line to follow up invites to look into cryptographic techniques that could replace the authentication via OOB, in order to cut down the impact of the user's actions in this process, relying instead in approaches with provable security. We dig into this topic in Chapter 9.
- In the scope of usable security, understanding the causes and frequency of user's behavior that deviates from the expectations is also an aspect to explore. In fact, several works address related topics (e.g., [182, 104]), including a user study on authentication ceremonies for messaging applications.
- We also assumed that private information between devices of the same user is correctly synchronized, nonetheless, the corresponding p≡p module is still under implementation. Hence, a security analysis of the corresponding p≡p's protocol and consequently possible improvements is an important direction to pursue.

Finally, we consider some general topics to follow for securing email.

- State-of-the-art protocols for secure messaging in general provide stronger security guarantees than those for email [50, 179]. We speculate whether the adoption of some of these approaches would make sense in the context of email.
  For instance, the Signal protocol [122], mixes multiple Diffie-Hellman shared keys (X3DH[2]) and refreshes keys for every message exchange (double-ratchet), to provide secrecy and authentication of message keys, and forward secrecy of messages [52]. An auxiliary central server stores all the public keys involved, allowing an interactive exchange. The use of such a server is indeed an important aspect to consider in the context of p≡p, since it stands against the decentralized paradigm adopted per design.
- Another idea to consider is how to automatically inherit trust from contacts shared with peers already trusted; a sort of an automatic web of trust. While there are many important considerations, e.g., how to get knowledge of shared contacts without violating privacy, we believe that this could be a direction worth studying.

We explore further some of these directions in the upcoming chapters and hope that at least some readers find enough motivation to pursue some others.

[2] https://signal.org/docs/specifications/x3dh/

7 A Socio-technical Security Analysis Framework

So far, we have focused on the analysis of security protocols in the presence of an adversary, which leads to find vulnerabilities regarding the logic of the protocol. Nevertheless, security protocols ground services meant to be used by humans (e.g., online payments, email, subscriptions, etc.), which makes the latter key elements in the achievement of security goals.

The formalization of security protocols typically assumes an ideal user behavior, according to the specifications, and thus their actions are excluded from the model. Yet, multiple factors influence a deviation from such an ideal behavior: complex security-related tasks, ill-designed user interfaces, unusable security policies, or lack of motivation/understanding to make adequate security-related decisions [56], among others.

These situations might be the cause of security threats, coming from non-malicious users. Therefore, considering the humans and their environment in a security analysis could help in the detection of flaws from a socio-technical perspective, which looks at the user-system interaction.

Security ceremonies, introduced by Ellison [73], are extended security protocols where humans, and their interaction with other nodes in the system, are explicitly included in the models. The formal analysis of security ceremonies is however not straightforward. Considerable effort has been devoted to developing techniques that allow to comprehensively model the behavior of users, the human-system interaction, and subsequently the formal analysis of ceremonies and its automation (e.g., [73, 17, 59, 100] and references therein).

Motivated by a question left open in the previous chapter, we focus on an aspect of security ceremonies less studied in the domain of formal security verification. We are interested in understanding whether the perception of security that users develop while interacting with a system corresponds to the actual security of the system, and whether and how, misaligned beliefs have repercussions in the latter.

In fact, there is evidence of inaccurate users' beliefs having consequences on the security of systems. For instance, a recent study [5] points out that in general users are not troubled by using secure messaging tools without authentication; and identifies users not understanding the security concepts alluded by tools, and the difficulty for users to directly evaluate the security provided, among the principal obstacles for the adoption of secure communication tools.

In this chapter, we present a formal framework for modeling security ceremonies on the basis of observed user's interaction with a system. The models allow the inclusion of two evaluations at particular moments in time: the values assigned by the system to specific security-related elements and user beliefs associated to such elements. We also introduce definitions of what we call misaligned-security properties, which express a relation between the user's understanding of security features and the system's assessment of such features.

The proposed framework is suitable for automatic verification via model checking, to detect dissimilar evaluations and to examine whether they pose a threat for the socio-technical security.

We start by contextualizing the scenario that frames this analysis; in Section 7.1.1, we briefly

introduce a conceptual model that captures all the elements required for the formalization. In Section 7.2, we present the formal framework and formalize misaligned-security properties; we also discuss their analysis by the use of model checkers. Section 7.3 covers the application of our technique for the socio-technical misaligned-security analysis of $\mathrm{p} \equiv \mathrm{p}$. The analysis pinpoints potentially problematic areas that could be investigated further by conducting appropriate user studies, which we discuss in Section 7.4. Finally, we briefly comment on how this approach could promote a more comprehensive understanding of socio-technical system security.

7.1 Research context

Research has largely focused on the formal analysis of Human-Automation Interaction (HAI), mostly for evaluating safety and correctness of tasks that involve human operation in automated critical systems, such as flight controllers, infusion pumps or even medical devices' interfaces (e.g., [61, 59, 94]). One of the objectives is detecting errors in the design of human-machine interfaces, that can potentially decrease the system's usability, thus preventing the human operator from achieving a specific goal (see [39] for a review of approaches in this area).

The main interest of such approaches converges to modeling possible interactions occurring via the user interface, and the properties typically concern correctness of the system when we consider alternative user's behavior. Thus, these analyses regard a technical angle.

Differing from those works, the approach proposed here focuses in defining properties that search for security threats emerging from the user-system interaction in ideal conditions, i.e., when users are honest participants and the implementation is correct. Hence, unlike in regular security protocol analysis, here we do not consider an adversary model.

An example of such properties is what we denote as *False-SoI* (false sense of insecurity), which is satisfied when a secure system fails to transmit the adequate degree of protection that it actually provides; on the contrary, the property *False-SoS* (false sense of security) describes insecure systems that manage to get users to feel secure, without providing proven security guarantees.

Given the nature of the analysis, we introduce the following assumptions on the systems to be studied:

1. The implementation of the system meets the functional requirements of the technical specifications (functional correctness).
2. The security of the system is asserted by proofs of security of the underlying cryptographic protocols.
3. Users execute only actions that are valid in the ceremony.

These assumptions rule out discrepancies derived from a faulty implementation and from users intentionally sabotaging the system—e.g., changing system's values by undetected means.

As a starting point, we use a diagrammatic representation of a socio-technical system, which comprises all the elements required for our focus analysis, and which we introduce in the next section. Remark however that the proposed formal framework does not depend on the existence of a such a representation; all the elements to be formalized can be gathered independently.

7.1.1 Multi-layered User Journeys

A multi-layered user journey[168] (MUJ) is a visual model based on multi-layered representations of security ceremonies [25], user journeys [71], and technical specifications of a system. This model

captures the path followed by a user when trying to accomplish a certain goal in a system, along with the user's beliefs regarding specific security-related aspects of the system, experimented through the path.

Multi-layered (concertina) models [25] of security ceremonies split a socio-technical system into its different information and communication layers. Thus, there are layers related to the user—e.g. to capture their behavior outside the system, and their actions in the user interface—and layers that model system's processes and protocols, including external nodes with which the system interacts—e.g. cloud services.

Inspired by user journeys, which depict thoughts and emotions experienced by a user during the process of accomplishing a goal, multi-layered user journeys extend concertina models with a user layer that captures a user's perceptions in relation to selected aspects.

The layers modeling the system, and claims about its security guarantees need to be gathered from technical specifications and security analysis, respectively.

It is important to highlight that a multi-layered user journey always refers to the experience of a particular user with a specific application and a specific goal.

Note that, even though security is nowadays at the core of most systems, users are rarely aware or even concerned about their existence. MUJs are mainly intended to model socio-technical systems whose aim is to provide some sort of security—e.g. secure email, secure messaging— and so, the security goals are considered to go in parallel with the user's goal—e.g., wanting to send an email only to the intended recipient and unreadable to anyone else, is in line with authentication and encryption.

7.2 Formal Approach for Aligned Sense-of-Security Analysis

We start by proposing a formalization of the multi-layered user journey. Informally, our model represents all the possible states that a system can reach when a user performs any possible sequence of actions to achieve a specified goal; in each state we can model the evaluations determined by the implementation and by the user's perception.

Then, we introduce classes of misaligned-security properties, which must be expressed in terms of unambiguous and measurable aspects of the system. Finally, we describe how to use these elements to execute an analysis for detecting and reasoning about existing discrepancies between the objective and subjective security of a system, as explained before.

Perhaps with minor adaptations, this approach is suitable for detecting non-security related misalignment in a wide variety of systems.

7.2.1 Formalization of multi-layered user journeys

Roughly, *labeled transition systems (LTSs)* are directed graphs used to model the evolution of systems, where nodes distinguish states of the system and edges represent transitions among states, triggered by actions labeling the edges [15].

To formally represent the ceremony of a user pursuing a specified goal $\mathcal{G}$ in a given system, we adapt an LTS with elements for modeling knowledge concerning facts, system assignments and user perceptions about certain aspects of the system. We call this structure an s-t (for socio-technical) transition system.

Definition 7.2.1. An *s-t transition system* $\mathcal{M}$ is a tuple $(S, Act, Prop, \rightarrow, I, A, evSys, evUsr)$, where

- $S = \{s_1, ..., s_k\}$ is a finite set of states

- Act is a finite set of action names
- $Prop$ is a finite set of atomic propositions
- $\xrightarrow{x} \subseteq S \times (Act \cup Cond) \times S$ is a transition relation, where $Cond$ is the set of propositional logic formulas over $Prop$
- $I \subseteq S$ is a set of initial states
- $A = \{a_1, \ldots, a_n\}$ is a finite set of variable names. Each $a_i \in A$ has a corresponding domain D_{a_i} with a partial order defined. The value *undefined* ($\perp$) is in every D_{a_i}
- $evSys, evUsr : A \times S \to \bigcup_{i=1}^{n} D_{a_i}$ are evaluation functions that, in a state $s_j \in S$, assign to an $a_i \in A$ a value from its predefined domain D_{a_i}.

This structure can be used to formally represent a MUJ as described next.

The actions that a user can perform through the interface are elements in Act. The elements in $Prop$ model statements about the socio-technical context (reality), whose truth value might also be undefined, and known or unknown by the system or by the user at any state.

The relation $s_1 \xrightarrow{x} s_2$ denotes a transition from s_1 to s_2 when action x occurs or when formula x holds; the labels on the edges are determined by the user's actions allowed at the corresponding state or by statements whose satisfaction is a condition to reach the next state. S and I are determined by the system layers of the MUJ.

To study discrepancies, we need to identify aspects of the system relevant for specifying security properties, and either directly or indirectly measurable by both, users and the system itself. The variables a_i in A capture such aspects; since they represent different concepts, each a_i has its own domain D_{a_i} defined accordingly. In a messaging application for instance, we could select a_1:"trust level of a contact" and a_2:"message's sending mode" with $D_{a_1} = \{trusted, unknown, mistrusted, \perp\}$ and $D_{a_2} = \{encrypted, plaintext, \perp\}$.

Finally, the output of $evSys$ depicts the value assigned in the system to an a_i at a specific state. The evaluation of $evUsr$ reports the user's perception of a_i, mapped into the required D_{a_i}, that was observed at a specific state.

Note that an invariable aspect to be evaluated concerns the achievement of the goal itself, i.e., at each state we want to assess the system's and the user's evaluation regarding whether $\mathcal{G}$ has been reached, if it is still in progress, or if the conditions at that state prevent its future completion. Hereafter, we will use the variable $goal \in A$ to identify this aspect, for which $D_{goal} = \{reached, ongoing, failed, \perp\}$ and $failed < ongoing < reached$. Also, we will denote as *final states* those $s \in S$ in which $evSys(goal, s) \neq ongoing$.

7.2.2 Misaligned-security properties

Computation Tree Logic (CTL) is a branching temporal logic, originally proposed by Clarke and Emerson [51], sufficiently expressive for the formulation of properties that consider all possible futures from a certain point in time.

CTL formulas are interpreted over transition systems. The syntax of CTL is defined by the following grammar over a minimal set of connectives in propositional logic, where p is in the set of atomic propositions:

$$\Phi ::= \top \mid p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathrm{E}\varphi \mid \mathrm{A}\varphi$$

$$\varphi ::= \mathrm{G}\Phi \mid \mathrm{F}\Phi \mid \mathrm{X}\Phi \mid \Phi\mathrm{U}\Phi$$

The operators **A** and **E** are path quantifiers and **G, F, X** and **U** are state quantifiers. Intuitively, the operators **A** and **E** respectively express properties, over all paths of a model $\mathcal{M}$, and over at least one path in $\mathcal{M}$. Analogously, the operators **G, X** and **F** allow to express properties expected to hold in all the states of a path (G), in the next state (X), or at least in a state reachable in the future (F). The until operator, **U**, expresses that, at some state of the path, a property ϕ_2 holds and in all the previous states, ϕ_1 is true ($\phi_1 \mathrm{U} \phi_2$).

Satisfiability of CTL formulas is decided by a state-based semantics, i.e., formulas are evaluated in a given state of a model. Then, a model $\mathcal{M}$ satisfies a formula ϕ ($\mathcal{M} \models \phi$) iff ϕ holds in every state of the model.

CTL is a logic for which efficient and relatively simple model-checking algorithms do exist. Readers interested in the topic might refer to standard textbooks for logic in computer sciences or model checking (e.g. [15, 97]).

Representative misaligned-security properties

We use CTL to express the misaligned-security properties due to its interpretation of time over a tree-like structure, where at each state, the future value of propositions is not yet determined. The assessment of security in our scenario conforms to this interpretation, since user and system gradually set a value to the security aspects as the former executes the ceremony to achieve the specified goal.

Hereafter, we use a_i^{sys} to denote $evSys(a_i, x)$ and a_i^{usr} for $evUsr(a_i, x)$, where x is the state in which the property is being evaluated. Then, we define the following misaligned-security properties, which can be considered each representative of a class:

Aligned-AoS At every state during the ceremony, the assessment of security that the user has about aspect a_i corresponds to the system's evaluation of security of a_i:

$$\mathbf{AG} \quad a_i^{usr} = a_i^{sys}.$$

Lower-AoS There is a state in the ceremony, where the assessment of security that the user has about aspect a_i is less optimistic than the system's evaluation of security of a_i:

$$\mathbf{EF}\ a_i^{usr} < a_i^{sys}.$$

Higher-AoS There exists a state during the ceremony, where the assessment of security that the user has about a_i exceeds the system's evaluation of security of a_i:

$$\mathbf{EF}\ a_i^{usr} > a_i^{sys}.$$

Misaligned goal There exists a path where, even if the evaluation of all other aspects is aligned all the time, the understanding regarding the achievement of the goal differs:

$$\mathbf{E}\big(\left(a_1^{usr} = a_1^{sys} \wedge \cdots \wedge a_n^{usr} = a_n^{sys}\right)\ \mathbf{U}\ \left(goal^{usr} \neq goal^{sys}\right) \big)$$

The first three properties might provide better insights if analyzed as presented, for each a_i; however, they can also be verified considering all the aspects at once, by expressing a conjunction of comparison relations,

$$\bigwedge_{a_i \in A} a_i^{usr}\ R\ a_i^{sys}\ , \quad R \in \{<, =, >\}.$$

As a remark, recall that the model allows cases where the user cannot determine a value for a certain a_i ($a_i^{usr} = \perp$). For a more accurate analysis, the properties should exclude evaluations for

which the value is unknown, and so for instance, *Aligned-AoS* should read as

$$\mathbf{AG} \quad (a_i^u \neq \perp \rightarrow a_i^u = a_i^{sys}).$$

For readability, we omitted this check in the formulas above.

The analysis is certainly not constrained to the sole verification of the misaligned-security properties defined in this section; introducing the comparison relation '$\leq$' for instance, already gives different insights. Analysts might as well define properties specific for the socio-technical system in question.

Informal interpretation of the properties

Considering an s-t transition system that models the system in question:

Aligned-AoS holds if the user always has an accurate perception about the security provided by the system. This property represents an ideal case in which the user understands how the system behaves in every state, and which however rarely occurs.

Lower-AoS holds if at any step of the process, the user assessed the specific a_i less secure than it was in the system. An interpretation of this property is what we introduced in Section 7.1 as "False-SoI": assuming that $D_{a_i} = \{secure, insecure\}$, if at some point $a_i^{usr} = insecure$ and $a_i^{sys} = secure$, then the user perceived a degree of protection lower than what the system actually provides; this false sense of insecurity might for instance lead the user to stop utilizing the system.

Higher-AoS is the complement of the previous, and so, it holds if at any step of the process, the user assessed the specific a_i higher than the system. The analogous interpretation for this property is the previously mentioned "False-SoS", with $a_i^{sys} = insecure$ and $a_i^{usr} = secure$, which reflects an insecure system transmitting to the user a high but misleading sense of security; here, the unawareness of the risk puts the user in a vulnerable position.

Note that when we consider $a_i = goal$, the previous properties provide insights about the usability of the system, given the assumption that $\mathcal{G}$ expresses a security related aim.

Misaligned goal holds when the system's and the user's evaluations of all the a_is coincide, except for the one regarding the goal achievement, i.e., the user could not successfully notice when or if the system reached a final state. The satisfaction of this property might be an indicator of shortcomings in the user interface or in the user experience.

7.2.3 Formal verification of s-t transition systems

Satisfiability of misaligned-security properties can be automatically verified in s-t transition systems by using tools for model checking [51], a verification technique that explores in a systematic manner all the paths reachable from each state in a model, to determine whether a specified property holds in the said model[1].

Before performing the verification, the evaluation functions $evSys$ and $evUsr$ need to be defined. The system assignments required to specify $evSys$ might be obtained by analyzing traces or logs recorded during the system's execution. The evaluations concerning the user perspective, which define $evUsr$, are obtained from the multi-layered user journey.

These functions could also be mappings of predetermined evaluations, prepared to obtain insights

[1] With the help of adequate data structures, state-of-the-art algorithms for model checking are able to handle models with large state spaces (1020 up to even 10476 states), treating effectively the so called states explosion problem that is intrinsic to model checking.

on specific hypothetical settings. In such a case, selected values from the domain are assigned to the variables, depending on the hypothesis. A posterior study with actual users would validate/invalidate the hypothesis and consequently, the results obtained.

An analysis of the verification's results pinpoints states of the system where certain security aspects are potentially misunderstood by users. Detecting such states enables the improvement of localized components of the system. Moreover, the identified situations are subject to other types of analysis, such as those targeted at finding and fixing root causes of confusions related to deficiencies in the interface design [61].

An ideal application of the technique consists on a constant iteration of the process: model-verify-analyze-improve, until an adequate alignment between subjective and objective evaluation of security is achieved.

7.3 Sense-of-Security Analysis of p≡p

We find a use case for the presented technique, in the analysis of socio-technical security aspects of p≡p, which expands the findings of previous chapters.

There are opposed opinions with respect to whether hiding how a secure email system provides security can lead users to a lack of trust in the system [158], or not [10]. Some works agree that the security properties that encryption tools offer should be explained to the user so that they can evaluate security trade-offs [14], along with adequate documentation regarding the tools[10]. Furthermore, Lerner et al. [114] remark that encrypted email may have adverse effects, if it provides a false sense of security to certain groups that have special requirements, such as journalists willing to protect metadata about communication patterns.

With the analysis in this section, we want to learn whether the approach adopted by p≡p for providing encryption and authentication is understood and properly performed by users, and what the repercussions could be if this were not the case.

As a reminder, authentication in p≡p is achieved via a handshake (Chapter 6), after which, a privacy rating is assigned to the recipient and then shown to the user in all future communications, to indicate the level of privacy with which a message will be sent. These ratings (TRUSTED, SECURE, MISTRUSTED and UNSECURE) are explained in Section 2.8.4; in the rest of this chapter, we will respectively denote each of them as `T`, `S`, `I` and `N`.

7.3.1 MUJ for sending an encrypted and authenticated email

We based our formalization in the MUJ of Figure 7.1, presented originally in [168], where the emotions in the first sub-layer hypothetically represent emotions that could be retrieved from non-expert users; the system layers of the diagram are based on technical specifications that we extracted by the technique in Chapter 3, along with the security protocols studied in the previous chapter. Details regarding the construction of this representation can be consulted in the cited reference [168].

The user goal that the MUJ depicts is $\mathcal{G}$: *to send an e-mail whose content is confidential, exclusively to a specific person.*

7.3.2 Formalizing the MUJ and properties

Figure 7.2 shows our model of the MUJ described above, as an s-t transition system, $\mathcal{M}_{\text{p≡p}}$. S and $\xrightarrow{x}$ are respectively represented by the nodes and arrows in the diagram; the initial state is s_0; the green labels conform Act while the orange ones are elements in $Cond$. The propositions in $Prop$ are defined as follows: *pepReceiver*="the recipient is user of p≡p", *handshake*="the authentication

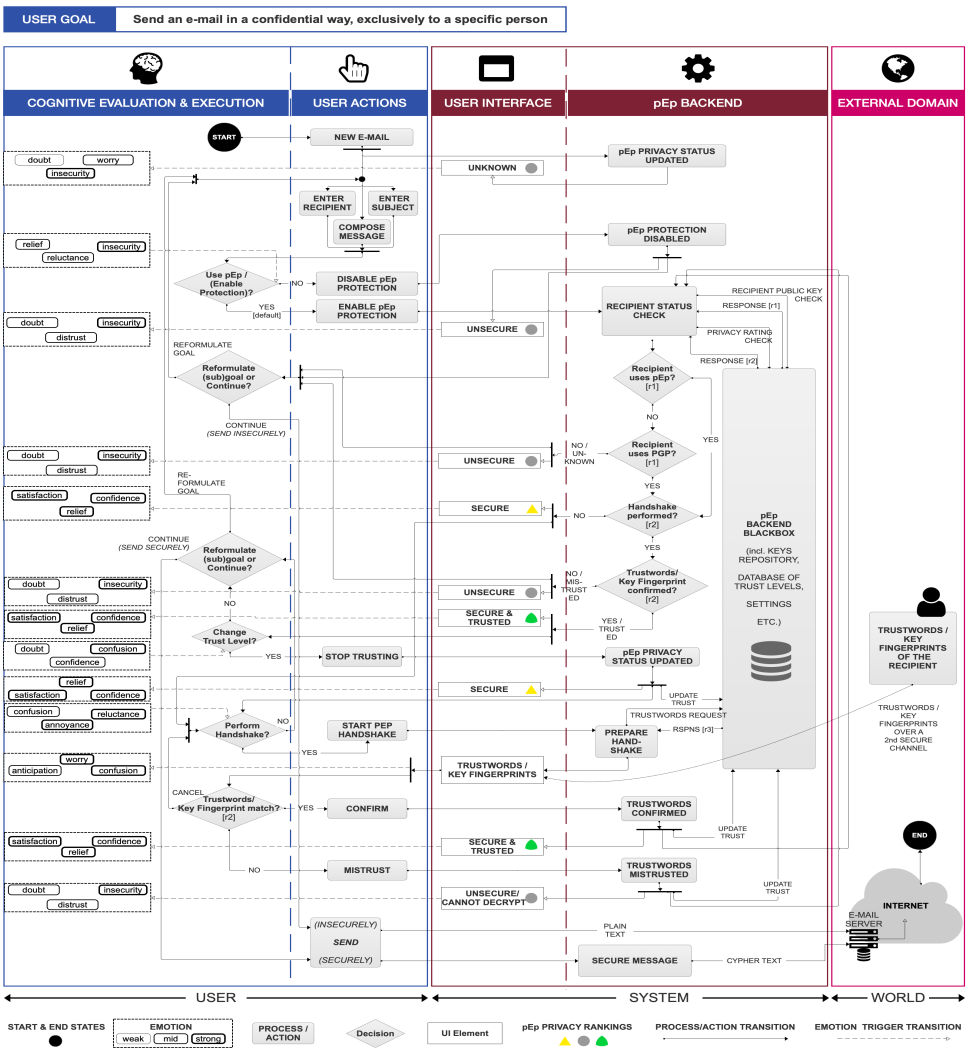

Figure 7.1: Multi-layered user journey depicting the process in p≡p of sending an e-mail whose content is confidential, exclusively to a specific person. Originally from [168].

protocol has been performed", *trustwordsMatch*="the compared fingerprints are equal in reality" and *confirmedTrustwords*="the user has confirmed in the system that the fingerprints match".

We define the set of security related aspects in which we are interested as $A = \{privacy, goal\}$, where $privacy$ represents the trust rating assigned to an email and $goal$ captures the status of $\mathcal{G}$, as defined in Section 7.2.1. The set of p≡p's trust ratings defines the domain $D_{privacy} = \{\bot, \mathtt{N}, \mathtt{I}, \mathtt{S}, \mathtt{T}\}$. In the graph, we use prv_{pEp} and prv_{usr} to model respectively $privacy^{sys}$ and $privacy^{usr}$ in each state; for example, $evSys(privacy, s_{14}) = \mathtt{S}$ is represented in the model by adding $prv_{pEp} = \mathtt{S}$ to the label of s_{14}. Similarly, $goal_{pEp}$ and $goal_{usr}$ express the evaluation of $goal$ returned respectively by $evSys$ and $evUsr$.

The evaluation function $evSys$ in each state is determined by the trust rating assignments as shown in the system layers in Figure 7.1; this definition relies on the assumptions from Section 7.1. Since our purpose here is to provide an analysis via the formal scheme presented, pragmatically, we determined the value assignments of $evUsr$ by mapping a subset of the emotional responses featured in the USER layer of Fig. 7.1 into $D_{privacy}$. The mapping, presented below, is subjective and merely illustrative; its validation and improvement by means of proper usability and User Experience (UX)

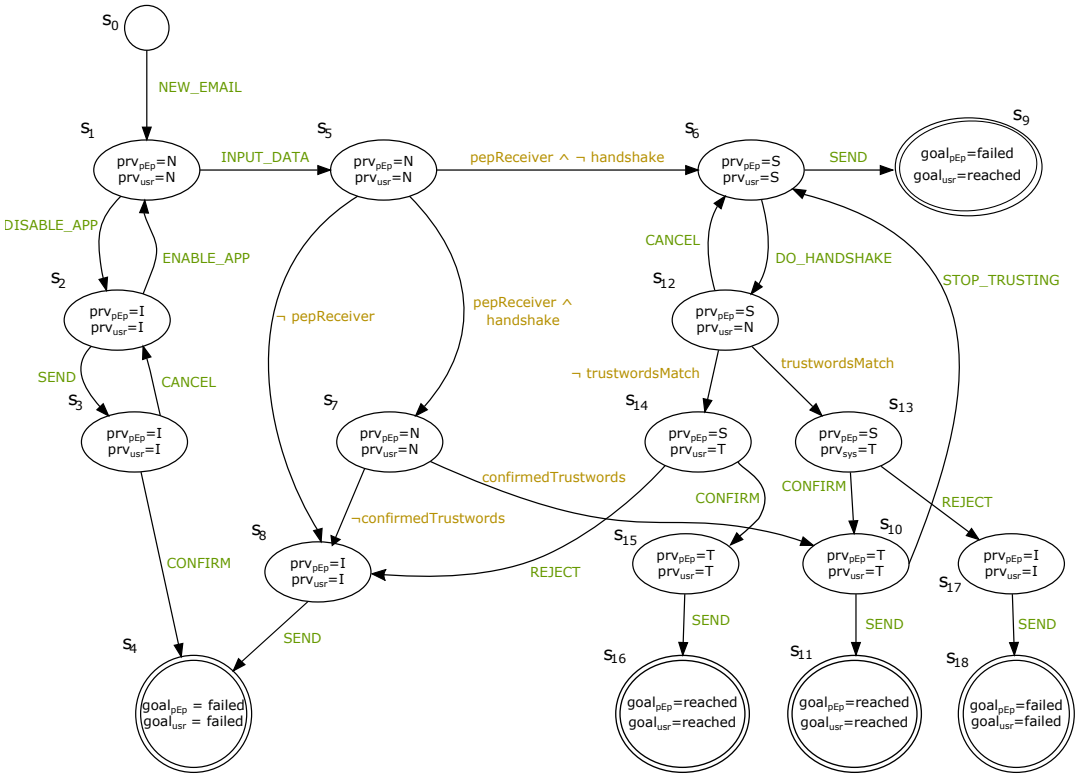

Figure 7.2: Model $\mathcal{M}_{\mathsf{p}\equiv\mathsf{p}}$ for the goal in p≡p: sending a confidential email exclusively to a specific peer. Where not specified, the values of prv_{pEp} and prv_{usr} remain as in the previous state, and the value of $goal_{pEp}$ and $goal_{usr}$ is *ongoing*.

related techniques would be insightful to validate the analysis.

Not reported by user $\Rightarrow \bot$
{doubt, confusion} $\Rightarrow$ N
{insecurity, worry, distrust} $\Rightarrow$ I
{security, reluctance, confidence} $\Rightarrow$ S
{trust, satisfaction} $\Rightarrow$ T

Then, to learn whether the user has an accurate understanding of the way in which the message is sent and how the trust rating changes during the process until the system sends it out to the world, we analyzed the misaligned-security properties:

P1. AG $(goal^{usr} \leq goal^{sys})$
P2. AG $(privacy^{usr} \leq privacy^{sys})$
P3. E $[(privacy^{usr} \leq privacy^{sys})$ U $(goal^{usr} \neq goal^{sys})]$
P4. AG $\left(\neg trustwordsMatch \rightarrow (\text{AG } privacy^{sys} \neq \mathtt{T})\right)$

P1, P2 and P3 are instances of the classes defined in Section 7.2.2. P4 is a property specifically of p≡p and expresses that, if at any state the trustwords lists (to be) compared are different, then p≡p's trust rating is never T in any of the future reachable states. This property can be customized to provide specific insights, for instance, to ensure that not only the system, but also the user never assesses the trust rating with the highest value if the trustwords do not match in reality.

7.3.3 Verification and Analysis

We used NuSMV [49], a model checker based on BDDs and SAT-solvers, to verify the misaligned-security properties in our model, and we found that only P3 is satisfied. The encoding of the model is included in Appendix A.3.

$\mathcal{M}_{\mathsf{p}\equiv\mathsf{p}} \not\models P1$. P1 does not hold when the user follows the trace $s_0 \to s_1 \to s_5 \to s_6 \to s_9$. This reflects the case of a user $\mathcal{A}$ wanting to send a confidential email to another p≡p user $\mathcal{B}$, yet, there is no record in the system of these peers having verified their identities (s_6).

As per p≡p's specifications, the system assigns the trust rating S to the message, thus, if $\mathcal{A}$ chooses to send the email (s_9), the achievement of $\mathcal{G}$ fails for the system because, even if the content is sent encrypted (confidentiality holds), p≡p does not have enough information to guarantee that the key used to encrypt the email belongs to $\mathcal{B}$ (authentication fails).

On the contrary, $\mathcal{A}$'s positive evaluation of $goal$ can be justified in several ways; for instance, $\mathcal{B}$ might have succeeded in reading a previous encrypted email from $\mathcal{A}$ and they might have talked about it; users might be reusing keys that they have authenticated somewhere else; or it could also be that $\mathcal{A}$ simply does not consider impersonation of $\mathcal{B}$ as a plausible risk.

$\mathcal{M}_{\mathsf{p}\equiv\mathsf{p}} \not\models P2$. P2 is not satisfied either, when the user goes through $s_0 \to s_1 \to s_5 \to s_6 \to s_{12} \to s_{13}$. The result is interpreted as follows: after the user, $\mathcal{A}$, selects to execute a handshake in s_6, p≡p shows the list of trustwords supposed to belong to the intended receiver $\mathcal{B}$ (s_{12}), which ideally, $\mathcal{A}$ is expected to validate with $\mathcal{B}$ through a second channel—e.g. via a phone call.

The state s_{13} represents the moment when the system generates for $\mathcal{A}$ the trustwords for a correct key of $\mathcal{B}$ and the peers determine that the trustwords are the same; at that point $\mathcal{A}$ is certain of being communicating with $\mathcal{B}$, but the value in the system has not changed because $\mathcal{A}$ has not pressed the confirmation button.

However, this violation to P2 occurs in a state that models a human action in the ceremony, where the system expects feedback from the user, thus, it does not represent a security problem as long as $\mathcal{A}$ submits the confirmation to p≡p in the next step (s_{10}).

$\mathcal{M}_{\mathsf{p}\equiv\mathsf{p}} \models P3$. For P3 to be satisfied there must exists a trace that satisfies the U condition. An instance is indeed the same trace found for P1, i.e. $s_0 \to s_1 \to s_5 \to s_6 \to s_9$, and the discussion for P1 explains this case too.

$\mathcal{M}_{\mathsf{p}\equiv\mathsf{p}} \not\models P4$. The satisfiability of P4 is falsified by the trace $s_0 \to s_1 \to s_5 \to s_6 \to s_{12} \to s_{14} \to s_{15}$, i.e., when the system generates for $\mathcal{A}$ a list of trustwords that differs from the list of $\mathcal{B}$ (s_{14}), and yet, $\mathcal{A}$ confirms the handshake in p≡p (s_{15}). Since the system can only rely on $\mathcal{A}$'s input regarding the words validation, it proceeds with the handshake, assigning the corresponding rating T to the message.

Furthermore, the model in Figure 7.2 shows that $\mathcal{A}$ indeed evaluates the message as T too (s_{15}). One explanation could be that $\mathcal{A}$ only started the handshake process but actually did not contact $\mathcal{B}$ to compare the words, and proceeded with the confirmation without noticing the disparity. $\mathcal{A}$ could have also confirmed even knowing that the trustwords were different, perhaps thinking that it was a minor mistake from p≡p.

In any case, the system and the user present an aligned assessment of $privacy$ and $goal$ (s_{16}) which however does not correspond to reality, and which is the cause of a security vulnerability as it nullifies the out-of-band (OOB) authentication process.

7.4 Contextualized Analysis and Related Work

The analysis shows that socio-technical vulnerabilities in p≡p are mostly related to the authentication process and confirms some known problems with the use of secure email identified by the usable security community.

For example, non-expert users might not be aware of the privacy threats present when authentication is missing. Such a scenario has in fact been observed in a user study where journalists and lawyers interact with a secure email solution [114]; the study reports only a few of the participants being concerned about the authenticity of the contacts. In the case of P1, the fact that p≡p shows SECURE as the trust rating for a particular message, might lead users to consider that they have already achieved $\mathcal{G}$, and hence, that performing the authentication step only provides some sort of additional security.

Another known problem, highlighted by P4, regards users not performing correctly the ceremony for authentication. This situation introduces a security weakness derived neither from an implementation problem nor from a misalignment, but from the authentication approach. From p≡p's perspective, authentication is reduced to users pressing a "confirm"/"reject" button; nonetheless, this decision is determined purely by human interactions in the security ceremony. The lack of precise steps defining the process for users gives them freedom to follow their own approach, according to which, their perception of security would be determined and then communicated to the system. Thus, the user's assessment would be aligned with the system's assessment, yet, they could both differ from reality.

7.4.1 Limitations

This formalization is designed to provide formal grounds to usability research, and therefore, the relevance of an analysis via this technique is tightly coupled with the input regarding the user layer, more precisely, the emotions and the extension at which the human behavior in the ceremony is represented.

For an analysis to yield realistic conclusions, particular importance shall be given to the use of a sound methodology to capture user's perceptions, and to an appropriate mapping of those perceptions into values comparable with the system's evaluations. For this reason, the analysis of p≡p that we provide requires to be validated by appropriate user studies, as the MUJ of p≡p for goal $\mathcal{G}$ is not based on an extensive user study but rather on the input received during a focus group session.

Selecting or developing appropriate techniques to gather the user's perception pertains to the field of usability and user experience (UX), and therefore is out of the scope of this chapter which is rather focused in the formal modeling. The topic remains open for interested readers.

Another remark is that $\mathcal{M}_{\mathsf{p}\equiv\mathsf{p}}$ models mainly system states. This can be improved by enhancing the MUJ, or alternative sources, with steps taken by users for performing tasks outside the system. For instance, s_{13} could transition to a state s'_{13} when a proposition $contactPeer$, which is true when the user contacts the peer to compare the trustwords, holds, and to s''_{13} otherwise. Then s''_{13} would transition to s_{10} and s_{17} and the path from s'_{13} could be expanded further according to the user's behavior; i.e.,

$$\left\{(s_{13}, contactPeer, s'_{13}), (s_{13}, \neg contactPeer, s''_{13}), (s''_{13}, CONFIRM, s_{10}), (s''_{13}, REJECT, s_{17})\right\} \subset \xrightarrow{x} .$$

In this case, the evaluations of $privacy^{sys}$ and $goal^{sys}$ in s'_{13} and s''_{13} would remain as in the previous state, while user assessments could enrich the model.

Finally, note that a MUJ captures the journey of a specific user and so, for an insightful analysis the verification would need to be executed for each observed participant in a user study, to then study in detail the different causes of misaligned assessments. Alternatively, for a more comprehensive analysis, the results from all the participants in the study could be considered to create a MUJ with the average user behavior per state; the formalization and posterior analysis would then identify common misaligned situations.

7.4.2 Related work

A closely related work which formalizes user interaction on a socio-technical level is by Deagni and Heymann [61]. They create a formal model based on two transition systems: one capturing user's assumptions of the software's state, based on a so called GOMS model of the user behavior, and the other representing the software itself. The principal difference with the approach that we covered in this chapter is that their verification concerns the correctness of the interaction in complex automated control systems, while here we address security expressed in terms of the alignment of the user's perception with the implementation.

Another similar work is by Beckert and Beuster [24], who present a methodology for the formal specification and verification of GOMS models of user behavior, under security aspects. Their formalization focuses only on the user interface layer of a system as opposed to the one presented here that runs across system and user layers.

7.5 Concluding Remarks and Further Directions

Recapitulating, in this chapter we introduced s-t transition systems to provide a mathematical based technique for analyzing a new class of so called misaligned-security properties in socio-technical systems. The formalization is suitable for automatic analysis, which is particularly advantageous for large systems.

We also introduced the class of misaligned-security properties to identify discrepancies between subjective user perceptions of security aspects in a system and the objective system assessment of those aspects; a formal verification yields an analysis of the repercussion of such dissimilarities in the socio-technical security of a system.

As mentioned Section 7.4, additional properties could be defined if the models are expanded with states representing the actions of the users where the system is not involved. Remark though that to obtain meaningful insights, this expansion would make sense if done in an interdisciplinary manner, with basis on user studies that report how users actually perform the modeled process.

The presented approach allowed us to study $\mathsf{p} \equiv \mathsf{p}$ from a different perspective which until the previous chapter was purely technical. From this analysis, we identified potential socio-technical issues related to the OOB authentication process.

There are several further directions from both, the formal methods and the usability perspectives, some of which we discuss next.

With a regular interaction with a system, the perception of users is expected to change, as tasks become familiar. The presented approach models a single observed run of a user, so, a refinement of the approach would involve the dynamic assignment of user's perceptions. An idea considers to look into formalisms for modeling and analyzing uncertainty, such as subjective logic, which allows to express beliefs about the truth of propositions at a specific moment, with some degree of uncertainty modeled by probabilities.

A more practical direction looks into ways to automate the generation of the formal model. This would require developing tools that allow the representation of a multi-layer user journey, which can then be parsed into a formal specification—for instance into the input language of NuSMV. The process needs to consider also the automatic extraction of the system assignments regarding the aspects to be evaluated.

From the $p \equiv p$ analysis perspective, we could extend the verification by considering other user goals within $p \equiv p$.

Finally an interesting direction from the UX perspective conjectures whether the user's assessment of certain aspects of a system is influenced by observing the evaluation of the system, such that they rely more on the values shown by the system rather than on the real facts. For instance, considering that a user confirms the trustwords of another user $\mathcal{B}$ without even contacting $\mathcal{B}$, $p \equiv p$ will show messages from such contact as TRUSTED; so, we wonder if by constantly seeing this trust rating the user starts to believe that the emails to and from $\mathcal{B}$ are authenticated, even being aware that the handshake did not take place.

Part II

Improvements to Secure Email and Secure Messaging

8 Protecting Weak Passwords in Password-based Authentication

In Part I, we explored and applied formal techniques to verify properties related to privacy (e.g. secrecy and authentication) and a class of misaligned-security properties regarding socio-technical aspects of secure communication protocols. In this part we reflect further on some of the findings and propose two approaches to deal with authentication-related problems: the first one aims at dealing with the user's selection of weak passwords; the second makes an attempt at automating as much as possible the authentication of public keys and key management in the context of secure email, to enable authenticated communication among peers.

8.1 Weak passwords and cleartext login credentials

In Chapter 5, we presented a protocol for detecting a password database (password file) leakage, under a compromised login server scenario, considering the architecture of the Honeywords System. Our assumptions included an adversary who managed to obtain a password database whose content was secured by some means, and who in addition could crack the content.

Yet, from a pragmatic angle, the Honeywords System is rarely implemented, and the ability to retrieve all the passwords from a secure database defines a strong adversarial model. However, there are situations that make password discovery easier, for instance, the use of weak passwords.

There are many interpretations for what a weak password implies. For instance, in [142] Pound explains aspects to consider when selecting passwords—e.g minimal length of 8 characters, the ineffectiveness of replacing letters by numbers; then, in [143] he shows a dictionary attack by which, using a computer with 48 Gb of RAM split in 4 graphic cards, he easily retrieves a significant number of passwords that fail to adhere to those mentioned guidelines. Here, we informally characterize weak passwords as low-entropy strings, simply derivable or easily guessable, either by humans, or by publicly available algorithms and tools that run in reasonable time.

Users selecting weak passwords for login is a well-known security problem (e.g., [40, 121, 188]), largely due to the misunderstanding about what makes a password strong and to users' unawareness of attack scenarios [180]. Rather commonly, passwords are short strings, built either from predictable combinations (e.g. "12345678","1q2w3e4r") or from common words that additionally tend to be semantically related (e.g. "lovelydog"), and often reused across different accounts [190].

Given the tendency of users to reuse passwords and the frequency of password databases breaches (as discussed in Chapter 5), the attempt to protect passwords in storage is indeed important for preventing the potential disclosure of a large number of user credentials and their posterior use for accessing multiple systems.

Password storage techniques have notably improved over the years; nowadays, memory-hard hashing schemes [95] are combined with a salt to make password-cracking computationally very

hard. Nonetheless, despite state-of-the-art techniques, poorly chosen passwords are still subject to be retrieved by offline dictionary attacks.

Another usually overlooked aspect that enables password discovery is that of users sending their password every time that they want to login. Regardless of the technique adopted for storing the passwords safe in the servers, the clients usually send the passwords in cleartext [34], perhaps justified by the use of encrypted channels, such as TLS. Still, anyone having access to the server at any time, can learn the passwords too; examples of this occurred in Twitter and Facebook, which until recently were storing logs with passwords in plaintext [7, 110].

In this chapter, we develop further the idea of the protocol presented in Chapter 5, to address both of these issues—i.e, weak passwords and passwords sent in cleartext—in the typical client-server architecture setting. We propose a password-based authentication protocol which masks the passwords with a seed obtained from an OTP device, to generate unpredictable dynamic authentication values.

We believe the usability of this protocol to be the same as in password-based methods with OTP; yet, this approach has the advantage of strengthening the protection of passwords in storage, and of preventing passwords from ever being sent in plaintext, thus eliminating weaknesses derived from their exposure.

We start by providing some context and background in Section 8.2 and Section 8.3. Then, we introduce the authentication scheme in Section 8.4 and provide an informal security analysis in Section 8.5. We presented this proposal in [186], primarily aiming to share and open a discussion on the idea. By now, we have investigated further and concluded that the security of the approach resides mainly in that of embedded solutions; yet, this approach slightly increases the cost in practice. We elaborate on this point in Section 8.6.

8.2 Related Approaches

Several academic and commercial approaches deal with the issue of users selecting weak passwords.

Password vaults—e.g. “Password Safe”[1] or “Password Gorilla”[2]—offer an option to deal with the user’s selection of weak passwords. The software generates secure passwords on behalf of the user and keeps them safe in an encrypted file on the device of the user, who can retrieve them at any moment by entering a master password. A drawback of this approach is that, as now the vault is a central repository, the risk associated to the user selecting an insecure password to access the vault, or moreover, the master key getting compromised, is drastically higher than in the initial problem.

A physical solution is YubiKey[3], a small USB-like stick that implements several protocols for strong two-factor, multi-factor and passwordless authentication. The user authentication is via fingerprint or by tapping. As in the case of password vaults, the main issue here is the centralized storage of keys, so that losing the YubiKey, prevents the user from accessing all the accounts managed by the device.

A more common approach refers to the use of multi-factor authentication—e.g., Google 2-step verification[4]—where in addition to the password, users are required to provide other values coming from different sources associated with something that they have, or to who they are (e.g. biometric

[1] https://pwsafe.org/
[2] https://github.com/zdia/gorilla/wiki
[3] https://www.yubico.com/
[4] https://www.google.com/landing/2step/

fingerprint). The access is granted only if each of the input values is equal to the value expected by the login server. This technique certainly strengthens the authentication process, however, users can still choose weak passwords and the leakage of a password file remains a serious issue.

Our solution remains within the context of password-based authentication. Similar to the use of a second factor, the approach that we propose relies on users possessing a One-Time-Password (OTP) device. Nevertheless, instead of merely considering such a device as a proof of possession, we combine the numbers generated by the OTP with the password, to further create fresh and unpredictable authentication values.

8.3 Cryptographic preliminaries

8.3.1 Securing passwords in storage

An approach used for a long time to protect passwords in storage consists in applying a hash function h (e.g., SHA-512) to the password, pwd, concatenated with a random value known as *salt*, i.e., $h(pwd||salt)$. While this approach protects hashed passwords from attacks that use pre-computed tables (e.g., lookup or rainbow tables), it is not effective neither against dictionary nor brute-force attacks, since the algorithms to compute hash functions are relatively fast.

Key derivation functions (KDFs) are deterministic algorithms used to derive cryptographic keying material from a possibly (but not necessarily) low-entropy secret value, such as a password. Password-based KDFs (PBKDF) help mitigate dictionary or brute-force attacks on the passwords by forcing the attacker to perform more computational work.

A PBKDF is defined by the choice of a pseudorandom function (PRF)—e.g., HMAC—and a fixed number of iterations c. The input parameters include a password p, a salt s recommended to be at least 128 bits, and the desired length in bit of the high-entropy output [176]. PKBDF algorithms typically apply c times the PRF to the input parameters, requiring considerable use of memory, and not allowing parallelization. Some examples are scrypt [139], argon2 [33] or bcrypt [149].

Note that, although PBKDFs make the execution of dictionary attacks expensive, the latter are still possible and weak passwords are still vulnerable.

8.3.2 One-time Password

One-time password (OTP) algorithms generate strings meant to be valid only for one login session or transaction on a system, which allows dealing with replay attacks.

An OTP algorithm is determined by a hash function, which produces a uniformly distributed number based on the following input parameters: a random seed (usually 256 bits or 512 bits long), a time-synched or an event-based parameter—e.g., a timestamp or a counter for time-based or event-based OTPs respectively—and additional variables to add entropy.

The HMAC-based One-Time Password (HOTP) algorithm [129] is an event-based algorithm which depends on an increasing counter value and a static symmetric key known only to the OTP device and to the validation server. The IETF specifications suggest HMAC-SHA-1[5] to generate a hash of the key and the counter. The 160-bits output is then truncated to a 32-bits value, and finally converted to digits, for usability reasons.

[5]Note that SHA-1 is not considered secure to generate digital signatures given that it is not collision-resistant and digital signatures are long-lasting values, thus, the adversary has the time-frame to forge them. However, collisions are not relevant in the case of OTP values since the main concern is to generate a random one-time use output, hence, using SHA-1 in OTP algorithms is still acceptable.

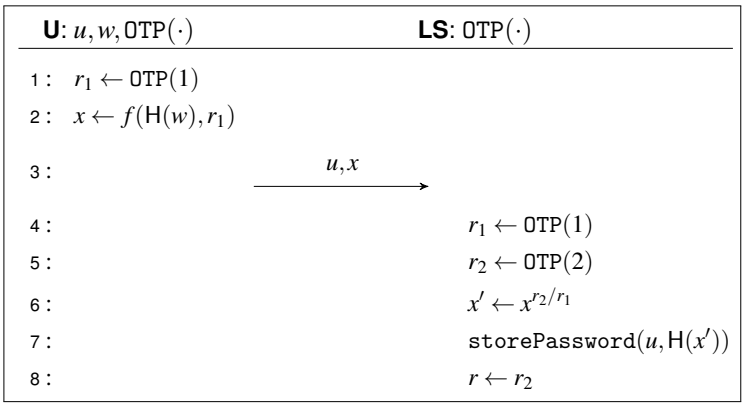

Figure 8.1: Registration protocol

8.4 Proposed Password-based Authentication Protocols

Considering a client-server architecture, we refer to the client as the User Interface (U) and to the server as Login Server (LS). Then, the authentication process requires two protocols: registration and authentication.

Assumptions We assume that U and LS communicate through a secure channel, e.g., implemented by TLS 1.3. As well, we assume that U holds an OTP device which has been delivered securely; the device implements HOTP and its output is aligned with the output produced by a corresponding OTP's generator algorithm in the LS.

The protocols are described next, where $\texttt{OTP}(n)$ is the number generated the n^{th} time that the device is used and H is a PBKDF. We omit implementation details, such as verifying that a given username exists.

8.4.1 Registration

This protocol (Figure 8.1) allows a user to register to a service, with a username u and a password w. The user operates the OTP device for the first time and gets a number $\texttt{OTP}(1)$ which inputs into the system (step 1). Then, U hashes w with H, and afterwards masks this value using $\texttt{OTP}(1)$ as a seed (step 2). The token obtained is sent to LS together with u (step 3). On reception, LS generates the first two OTP numbers, r_1 and r_2, (steps 4, 5), and uses them to mask the identity token received from U (step 6); for this step, LS does not need to know the password, as we explain below. The new token is hashed once more with H, and then stored as u's password (step 7). LS also stores r_2 before concluding the protocol.

The masking function f is implemented by modular exponentiation, akin to the blinding in Chapter 5: for each username, LS possesses g, a generator of a multiplicative subgroup $\mathbb{G}$ of order q, such that, $f(\mathsf{H}(w), r) = g^{r \cdot \mathsf{H}(w)}$, where $r \in \{1, \cdots, q-1\}$ is a random number—here, synchronized with the user's OTP device. This definition allows f to be computed in terms of the current value $f(\mathsf{H}(w), r)$, the current random exponent r, and a new random value r':

$$f(\mathsf{H}(w), r') = g^{r' \cdot \mathsf{H}(w)} = g^{r \cdot \frac{r'}{r} \cdot \mathsf{H}(w)} = (g^{r \cdot \mathsf{H}(w)})^{\frac{r'}{r}} = f(\mathsf{H}(w), r)^{\frac{r'}{r}} \quad (1)$$

8.4.2 Authentication

In the authentication phase (Figure 8.2), U submits a username u, and the hashed password w, masked with the current $\texttt{OTP}(n)$ (step 3). LS retrieves u's authentication value from the password file

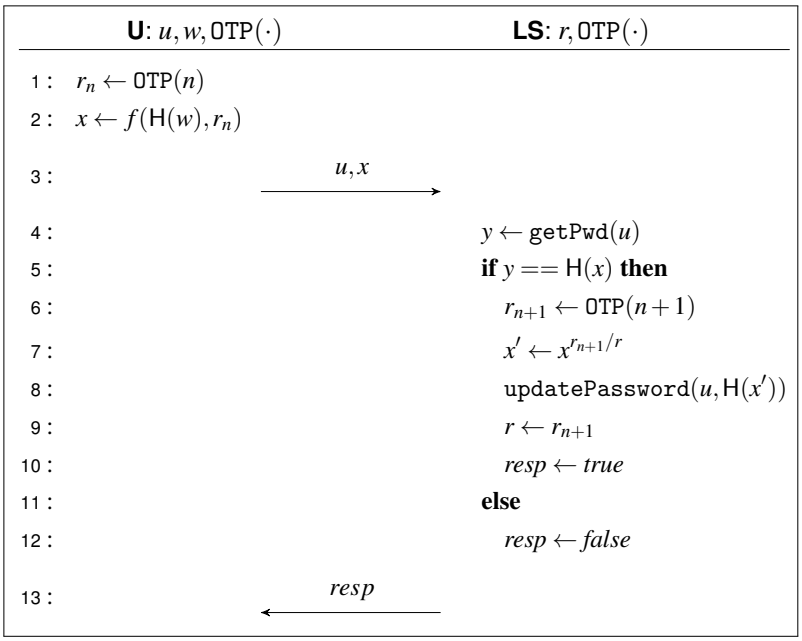

Figure 8.2: Authentication Protocol for $n > 1$

(step 4) and for authenticating, it applies H to the received token x, which must match the value stored in the file (step 5). If the check succeeds, LS generates a new OTP which it uses to compute a new authentication value for u (steps 6,7) as described by equation (1); then LS updates the password file and r (steps 8,9). If the compared values are not equal, the access is denied (step 12). LS concludes the protocol by sending a response to U (step 13).

8.4.3 Implementation

A way to perform the modular exponentiation for the masking function f is by elliptic curve (EC) multiplication, for which there exist efficient implementations (e.g. [116] and references therein). As we mentioned before, for H one can use hard-memory key stretching functions, such as the PBKDFs mentioned in Section 9.1.

8.5 Informal Security Analysis

We discuss informally the security of the protocol with respect to the following scenarios, where an adversary $\mathcal{E}$:

(a) tries to guess the user's password,
(b) has observed previous authentication values from the user,
(c) has stolen the password database,
(d) has stolen the OTP device and the password file, but not the password.

For case (a), $\mathcal{E}$ does not have any information, perhaps only social data if we consider a specific user attack. In this case, the best adversarial strategy is an online guessing attack based on dictionaries or personal information. However, even if $\mathcal{E}$ guesses a weak password within the number of attempts allowed by the system, $\mathcal{E}$ still needs the OTP value for a successful authentication, and therefore, to learn that the guess was correct.

In case (b), by learning the authentication values sent in previous sessions—possibly by accessing a log file—$\mathcal{E}$ cannot do much better, as the values are one-time valid and new authentication values

are independent from previous values. Even if $\mathcal{E}$ somehow learns the password, the only way to obtain the OTP value is by brute-force, which is infeasible considering that systems restrict the number of online attempts.

If $\mathcal{E}$ gets access to the password database, as in case (c), then an offline dictionary attack is possible. Here however, the adversary would require to brute-force not only the password, but also the OTP value. Considering that k is the size of the dictionary and that the standard for HOTP defines the length of an OTP value to be 32 bits [129], the complexity of the brute-force algorithm is $O(k \times 2^{32})$ instead of $O(k)$. Theoretically, the increment by a linear factor has no significance, however, considering that databases are in the size of billions [88], this factor could make a difference in runtime and resources needed. Moreover, the guessing cost can still be increased by an appropriate selection of H.

Note also that a correct guess of the password and the OTP value serves to access the system only once. So, although only finding the next OTP is easier, it still requires an active adversary executing the attack for each login attempt.

Finally, a dictionary attack in case (d) is basically twice as hard as a dictionary attack to H plus some δ. Given that $\mathcal{E}$ knows the required OTP value r_t, for each guessing attempt with word w, $\mathcal{E}$ needs to compute $\mathsf{H}(f(\mathsf{H}(w), r_t))$.The δ is added by the calculations for the modular exponentiation, which however can be efficiently executed by implementations based on elliptic curves (e.g. [43]).

8.6 Comparison with Techniques for Protecting Passwords

Our protocol has two main advantages: (1) it makes attacks more costly for adversaries trying to retrieve weak passwords from a leaked password database, and (2) it prevents the exposure of cleartext passwords at any moment. There are two close approaches that we consider meaningful to discuss.

Password-based KDF. The use of PBKDF already offers a solution for the protection of weak passwords. Yet, there are some advantages of combining it with the protocols presented here.

A first advantage regards the salt, which is a static value per user usually stored in the password database, thus, a password database compromise leaks the salt values too. This allows the adversary to fix the salt value when performing a dictionary attack. In the approach presented here, the database does not need to store any information about the OTP value; one could for instance store only the value n of the counter for the OTP generation algorithm and calculate r_n at runtime. Even in the case that the last OTP was stored, this value would only be useful to log in one time; further accesses would require new cracking as discussed in Section 8.5.

Another advantage relates to the second problem that our protocol aims to address, i.e., sending the passwords in plaintext. In our approach, there is no record of the password anywhere in the protocol, as it is never seen in clear other than by U. This removes the possibility of learning the passwords from other resources than the password database itself, e.g., login logs where the passwords are shown in clear (as in the examples from Section 8.1). With this approach, if $\mathcal{E}$ gets such logs, obtaining the passwords would amount to cracking strings of the form $f(\mathsf{H}(pwd), otp_n)$, which is as hard as cracking a password database.

Lamport's approach. In an influential theoretical work, Lamport [112] proposed a solution for preventing the impersonation of users by an adversary that has compromised the password file, and that can intercept the communication between the login application and the system.

To solve the storage of cleartext passwords, he proposed the use of one-way keyed functions, where the key is the password itself; this is the underlying idea behind hashed password databases and what we achieve here by the use of a PBKDF.

For solving the interception of messages in the insecure channel between the user and the system[6], Lamport's idea roughly consists in pre-computing an authentication value by iteratively applying k times a one-way function F to the password x, let us say $F^k(x)$, which is stored by the login server. Then, in the authentication process the user's client sends each time $x_i = F^{k-i}(x)$, hence, the server only needs to apply F to the received value in order to obtain the value stored; the latter is possible because, after each successful authentication, the system updates the stored value with the last x_i received.

Nowadays, this idea is not scalable in our context, as it works on the premise of computing in advance a value bounded by a fixed number of iterations, from which the rest of the authentication values are computed "backwards"; then, when the first value of the chain is reached, a new boundary should be computed and securely communicated to the system in order to keep the access. As people log into some systems even more than once a day (e.g., to email), a very high number of iterations should be selected, which would provide a big window of opportunity to crack a password by brute-force or dictionary attacks; alternatively, requiring users to contact service providers very often to generate new values poses usability-related concerns[7]. We approach the issue with the use of an OTP device, which provides a random value for every authentication attempt, that allows to execute a similar computation of new authentication values in terms of the previous ones.

OPAQUE. Password-authenticated key exchange (PAKE) protocols are a cryptographic tool originally designed to allow two parties to agree on a cryptographic secret key using a low entropy password and public key cryptography. However, they are suitable for user authentication and moreover, they address the problem of sending passwords in cleartext. PAKE allows a client U and a server LS to agree on a shared key, which is the same if and only if both parties derive it from the same password; hence, the authentication consists on verifying that both parties have derived the same key.

In particular, a recent PAKE protocol called OPAQUE [99], allows the server and client to jointly compute the authentication value via a two-party protocol, called an *oblivious PRF*, in which the LS knows only the salt and U knows only the password and neither party learns anything about the other party's input [87].

Roughly, to agree on a shared key, U generates $b = H(p)^r$, where p is the password, r is a random scalar value, and $H : \{0,1\}^* \to \mathbb{G}$ is a hash function that hashes passwords into elements of a cyclic prime-order group. Then, U sends b to LS, who generates a salt s, computes $c = b^s = H(p)^{rs}$ and sends c back to U. Finally, U computes $k = c^{\frac{1}{r}} = H(p)^s$.

The idea is that upon registration, LS generates a public key pk_u and private key sk_u for the user, and stores $e_u = encrypt(k, sk_u | pk_s)$, where pk_s is the public key of the server. Upon authentication, LS sends e_u to U, which can be decrypted if the user entered the correct password—since U can derive k. Then, sk_u is used to run a standard key agreement protocol allowing LS to authenticate the user.

While this technique seems to solve the problem quite nicely, it is still recent and mostly unknown,

[6] The first released version of SSL, SSL 2.0, became public in February 1995, and TLS 1.0 in January 1999.

[7] Lamport's algorithm has some practical uses, e.g., for authenticating messages exchanged in the establishment of sessions in distributed systems.

perhaps also because it is an indirect approach for verifying passwords, in the sense that the technique that was not originally designed for that. We explore further the use of PAKEs in the next chapter, for peer-to-peer authentication and discuss further on the adoption issue.

8.7 Limitations and further directions

We discussed some practical advantages of the scheme presented here. Yet, from a theoretical perspective it has some limitations derived from the fact that, even if the OTP value has a uniform distribution, it is a 32-bits long value, hence, subject to brute-force attacks, and not suitable to provide the high entropy required for the exponent in the function f.

We could obtain a high-entropy value if instead of the short-string shown in the device, we use the digest of the hash function in the OTP generation algorithm—whose length depends on the hash function selected, usually 160 bits—just before truncation. However, this solution has a significant usability drawback: instead of an OTP device, the user would require for instance an application to obtain each time a representation of the full value—e.g., shown a list of approximately 10 words as in $\mathsf{p} \equiv \mathsf{p}$—which then would need to be input in order to log in. Alternatively, the user could input the parameters that the OTP generation algorithm requires, i.e., a key shared with the server and the counter, but again, this represents too much burden for the user given the size of the cryptographic shared key.

Another option could rely on the client U implementing the OTP generation algorithm and a user's device d storing the input parameters that the algorithm needs. Roughly, upon registration LS would communicate the symmetric shared key, k, to U; this can be done by showing a QR code that encodes k, which will be stored in d (the device used for scanning the code), similar to Google authenticator's approach[8]. To generate the high-entropy tokens, the user is asked to confirm each login attempt; this confirmation triggers d sending k and the counter to U, who then can run the OTP algorithm, and subsequently the protocols as presented in Section 8.4.

As with the OTP device, the difference of the scheme just described with Google authenticator resides in the use of the random tokens for masking the passwords and not only for authentication; and the advantage this time is that we get high-entropy values. Clearly this idea still requires to be carefully developed, and we believe that this is a direction worth pursuing.

8.8 Concluding remarks

Users selecting weak passwords will be a persistent issue as long as password-based authentication exists. While current techniques make it increasingly costly and inefficient for an adversary to retrieve weak passwords, perhaps the advent of new technologies capable of replacing password-based authentication is the only real solution for solving the continuous challenges associated to the dependency on users being responsible for the pivotal element in the achievement of authentication: the password.

[8] https://github.com/google/google-authenticator

9 Automating Entity Authentication from Low-entropy Secrets

As we have seen along this thesis, entity authentication, consisting in users validating each other's public key (key validation, key verification), is an essential process which secure email solutions should implement.

Over the years, several methods for accomplishing key validation have been established; known examples include: manual validation of fingerprints—in diverse representations, e.g., hexadecimal numbers, or trustwords as in p ≡ p—web of trust, public key infrastructure (PKI) and hierarchical validation, public key directories, keys generated by trusted-servers (as in identity-based encryption), and more recently transparent logs. Some of this approaches were introduced in Section 2.5.

The set of techniques available becomes much smaller when we consider a decentralized scenario, where neither a PKI nor a trusted third party (TTP) are used.

Approaches based on the use of out-of-band (OOB) channels for dealing with decentralized key validation have received much attention from the research community (see a survey by Nguyen and Roscoe [134] and references therein). Considered first by Rivest [153], many of such approaches are inspired by the work of Vaudenay [181] based on short authentication string (SAS) comparisons. SAS-based schemes have been recently formally analyzed in the symbolic model [62] .

Due to the involvement of the user required in most of these approaches—e.g., to manually compare two strings representing fingerprints—usability of the solution is a key factor for the successful achievement of entity authentication, since as we discussed in Chapter 7, users failing to correctly execute the process (in particular due to the lack of understanding about the steps or difficulty to follow them) can cause the introduction of security flaws. Therefore, reducing the gap between security and usability by finding optimal trade-offs has been a central theme of research for decades, with a plethora of long-standing open problems [179, 50].

Attempting to improve usability in the entity authentication process, Alexander and Goldberg [9] proposed an improvement to the Off-the-record Messaging (OTR) protocol [42]. Assuming that users share a low-entropy secret, they modify a solution to the socialist millionaires' problem (SMP) [44] to determine if the shared secret is equal, authenticating the peers in an affirmative case. To the best of our knowledge, this is the only approach for key validation based on pre-sharing a secret; the solution is mainly suitable for online (synchronous) settings.

In this chapter we propose a user-friendly solution, based on password-authenticated key exchange (PAKE) protocols, for the authentication of public keys in a decentralized setting. Similar to the OTR solution, our approach assumes that users pre-share low-entropy secrets. These secrets are not expected to be sampled from a large, uniformly distributed space, but rather from a small set of values, e.g., typical human-memorable passwords, pin numbers or even natural words that answer specific questions. Our solution is suitable for online and offline settings, being thus compatible with the inherently asynchronous nature of email and modern messaging systems.

After motivating the use of PAKE for this problem, we start by reviewing a few vulnerabilities of voice-based out-of-band authentication in Section 9.2; in particular, we study a combinatorial attack

against lazy users, which we analyze in the context of $p \equiv p$. Then, in Section 9.3 we describe how a *secure equality test* can be solved using PAKE, consequently achieving entity authentication and establishing a shared high-entropy secret key. We also discuss security and usability properties, along with enhancements that could be implemented thanks to the derived cryptographic shared secret key. We conclude in Section 9.6 with remarks and future work.

PAKE protocols for entity authentication Secure email and messaging solutions in general do not consider the authentication step as mandatory for the communication, but rather only to upgrade its security; furthermore, this feature tends to be unnoticeable—e.g., to access the authentication screen in Signal and Whatsapp, users need to (1) enter a chat, (2) click on the contact's name, (3) select "View safety number/Encryption" respectively—which might cause users to neglect this step. As in the case of $p \equiv p$, solutions that do consider the entity authentication problem in decentralized non-PKI environments, typically rely on users correctly executing a manual comparison. Hence, our first goal is to replace OOB authentication with a cryptographic protocol, to reduce the impact on security of failures derived from the high dependency on user behavior.

We propose using PAKE protocols to solve the problem treated in this chapter, primarily for two reasons: because they are independent from PKIs and TTPs, which conforms with a decentralized trust model, and because they provide a zero-knowledge (ZK) solution for the secure equality test problem using a low number of rounds, which allows to effectively send messages over the network without the need of several responses from users, making them compatible with asynchronous settings.

By solving SMP via PAKE, we additionally establish a shared cryptographically-strong secret key; this key enables the achievement of further cryptographic properties that neither OOB approaches nor other known solutions to SMP can provide. In addition, this approach does not require any understanding of cryptographic concepts from the user, such as public-keys and fingerprints.

The use of PAKE protocols in this context also addresses two open problems in secure email and messaging [179, 50]: bridging the gap between known theoretical results and real-world solutions, and the need for more robust authentication methods that also improve the trade-off between security and usability in secure solutions.

9.1 Framework and Preliminaries

For the rest of the chapter, we use $\mathcal{A}$ and $\mathcal{B}$ to refer to honest parties Alice and Bob, and $\mathcal{E}$ for the adversary, Eve. We use $\leftarrow\!\!\$$ to denote an element sampled uniformly at random, and $\|$ to denote concatenation. We denote low-entropy secrets provided by users with π.

Security model We consider the standard Dolev-Yao model, as defined in Chapter 4. Regarding PAKE protocols (a.k.a. PAKEs), we will discuss various constructions in Section 9.3, largely proven secure in the so-called BPR model [26] under various hardness assumptions. We do not assume any trusted infrastructure. We also assume candidate implementations of our solution to be correct and trusted (not corrupted).

System requirements Our proposal does not require any format modifications for the transmission of messages and preserves compatibility among existing email clients and servers; therefore, we assume the standard requirements for email transfer. As for secure messaging, we do not introduce any extra trust assumptions and no additional infrastructure would be required. In one of the proposed methods for transport mechanism, we assume the existence of untrusted buffer/relay

servers, somewhat akin to the ones used in the design of Signal or OTR4 (see Section 9.3.4).

9.1.1 Cryptographic background

In future sections, we will refer to the computational hardness assumptions, mostly Diffie-Hellman (DH)-based, and models in which PAKEs have been proved to be secure. Roughly, given a cyclic group $\mathbb{G}$ of prime order q, a generator g of $\mathbb{G}$, and two elements g^{x_1}, g^{x_2} sampled from a uniform distribution, with $x_1, x_2 \in \{0, \ldots, q-1\}$ secret exponents, the Computational Diffie-Hellman (CDH) problem assumed to be computationally intractable is to compute the value $g^{x_1 \cdot x_2}$. In turn, the Decisional Diffie-Hellman (DDH) problem underlying the DDH hardness assumption consists in distinguishing the tuple $(g^{x_1}, g^{x_2}, g^{x_1 \cdot x_2})$ from the tuple $(g^{x_1}, g^{x_2}, g^{x_3})$, where g^{x_3} is a group element chosen from a uniform distribution. Detailed explanation on these assumptions can be consulted in books of cryptography, for instance [106].

We also discuss schemes based on the Ring Learning With Errors (RLWE) problem [119]. This is a special case of the Learning With Errors (LWE) problem [150] whose security may be reducible to the hardness of solving the Shortest Vector Problem (SVP) in lattices, for which no efficient quantum algorithms are known; thus RLWE is conjectured to be quantum-secure.

We use $\mathsf{KDF}(s)$ to denote a key derivation function that takes a source s of keying material, typically with a fair amount of entropy but not uniformly distributed, and produces one or more cryptographically strong secret keys; interested readers can refer to [109] for details. We denote with $\mathsf{MAC}(k, m)$ a keyed message authentication code scheme that computes a tag on the message m under the key k (see Section 2.4 for a review).

We refer to zero-knowledge proofs, which allow Alice to demonstrate the correctness of a certain fact to Bob without revealing any additional information. For example, in the context of SMP, Alice can demonstrate via a zero-knowledge proof that a certain exponent used in her calculations has not changed throughout the protocol, without revealing the value of the exponent itself.

Intuitively, the notion of forward secrecy (FS) captures the requirement that a long-term secret compromise should not result in prior session keys getting compromised and consequently the corresponding exchanges. Weak FS (wFS) refers to those schemes satisfying FS against passive adversaries who did not interfere in the previous sessions and perfect forward secrecy (PFS) to those achieving the same against active adversaries. We will come back to this notions in Section 9.4.2.

Post-quantum (PQ) cryptography encompasses schemes that are considered to be safe against adversaries equipped with scalable, cryptographically relevant quantum computers.

9.1.2 PAKE

Password-authenticated key exchange (PAKE) protocols allow two parties who share only a low-entropy secret, hereafter password, to agree on a cryptographically strong shared secret key, using the password for authenticating each other. Thus, these protocols allow to establish secure channels without the need for a PKI, TTPs or empirical OOB alternatives.

Since their introduction by Bellovin and Merritt [27], numerous PAKE protocols have been proposed; we will review some in Section 9.3.3. They largely fall into the two categories of balanced (symmetric), which allow parties that use the same password to negotiate and authenticate a shared key, and augmented (asymmetric or aPAKE), which store one-way mappings of passwords on the server side in client-server scenarios.

PAKE protocols present interesting security properties; among them, a run of the protocol should not leak any information related to the password, they should be resistant to offline dictionary attacks,

and an online guessing attack with at most one test per run should be the optimal attack strategy for an active adversary $\mathcal{E}$ interacting with a party. As it occurs with the SMP protocol, $\mathcal{E}$ can mask failed password guessing attempts as network failures, thus allowing numerous attempts without raising suspicion. This is in general unavoidable, however, we will see in Section 9.3 how a recent work [156] can mitigate this attack.

Regardless of its features, PAKE is a method that does not seem to have enjoyed enough recognition. Possible reasons include a lack of mature implementations, reluctance towards implementing cryptography in browsers, patent-encumbered designs and even unawareness of its usefulness. There are some exceptions though. Secure Remote Password (SRP) [1] for instance is an augmented PAKE protocol designed for securely authenticating clients to servers. SRP was created to work around the patents issue and has been standardized for secure mutual password authentication in TLS. Open source implementations are available, e.g., as part of OpenSSL[2]. Although it has some drawbacks compared to other PAKE protocols, it is presumably the most widely-deployed protocol.

A more recent protocol that solves some of the issues with SRP is OPAQUE (referred later in this chapter), however, robust and mature implementations are still required.

Note that the problem that we treat here with PAKEs differs from the client-server password authentication treated in previous chapters (Chapter 8), since in the latter, the server and client code are controlled by the same entity and authentication is required only in one direction.

9.1.3 Socialist Millionaires' Problem

Yao's millionaires' problem [195] is a famous secure multi-party computation problem in which two millionaires want to find out whose input is greater without revealing any additional information on the actual value. SMP (a.k.a secure equality test) is a variant of Yao's problem with the difference that the parties only wish to know whether their inputs are equal. Then, SMP reduces to two parties $\mathcal{A}$ and $\mathcal{B}$ verifying equality of their inputs $\pi_{\mathcal{A}}$ and $\pi_{\mathcal{B}}$ in a zero-knowledge manner such that by the end they learn nothing but the boolean result of the equality test.

A series of works have been dedicated to solving SMP, including a well-known protocol by Boudot et al. [44] (known as the SMP protocol) that provides, under the DDH assumption, a fair and efficient protocol, where fairness roughly means that no party can evaluate the function and walk away with the result without the other party learning the output.

Garay et al. [82] showed that the fairness and the security definition of Boudot et al. are not compatible with security definitions in the simulation paradigm, and that their solution would not be secure when composed concurrently. They present a construction that can be composed arbitrarily, with similar complexity results.

9.1.4 The Off-the-record Messaging (OTR) authentication protocol

Off-the-record Messaging (OTR) [42] is a cryptographic protocol originally designed by Goldberg and Borisov to enable encrypted, authenticated and deniable instant messaging conversations, which also provides forward secrecy. OTR was proposed as an alternative to PGP for "casual" conversations. Plugins implementing the protocol and related software are provided in the OTR's website [171].

To improve usability of the original process for authenticating public keys, which was assumed to be OOB, Alexander and Goldberg [9] proposed a variant of Boudot et al.'s SMP protocol. Assuming that $\mathcal{A}$ and $\mathcal{B}$ share a low-entropy secret—e.g. a password, common knowledge of a certain event,

[1] `http://srp.stanford.edu/`

[2] https://www.openssl.org/docs/manmaster/man1/openssl-srp.html

etc—each party derives a value $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$ from three elements: a `SHA-256` hash of the session ID, the two parties' fingerprints, and the shared low-entropy secret. The goal of the protocol is to determine equality between $s_{\mathcal{A}}$ and $s_{\mathcal{B}}$.

The protocol starts with g a generator of a group $\mathbb{G}$ of large prime order q, which $\mathcal{A}$ and $\mathcal{B}$ use to jointly compute two more group generators through DH exchanges. Then the parties use these three generators, their respective secret, and each party a chosen element in $\mathbb{Z}_q$, to calculate blind terms that would serve as input for the equality test, and a new exchange has place. All the values passed between Alice and Bob are accompanied by a zero-knowledge proof. The equation to test equality is of the form $R = x \cdot w^{s_{\mathcal{A}} - s_{\mathcal{B}}}$, where x and R are known to each, $\mathcal{A}$ and $\mathcal{B}$; hence, to determine if $s_{\mathcal{A}} = s_{\mathcal{B}}$, they only need to check whether $R = x$, or alternatively, whether $R/x = 1$. For a detailed description and security analysis of the protocol, we refer the reader to sections 4.2 and 4.3 of the original paper [9].

This protocol prevents man-in-the-middle (MITM) and replay attacks, however, it does not enforce fairness as the original SMP protocol. In addition, the process requires $\mathcal{A}$ and $\mathcal{B}$ to be online at the moment of authentication, to enter the secret and for the messages to be exchanged.

9.2 Pitfalls in OOB Authentication

In out-of-band (OOB) authentication, users compare some representation of a cryptographic hash—fingerprint—of their partners' public keys via a separate authenticated channel. This representation is typically presented as a list of words, numbers, or images, although some applications have managed to simplify fingerprint comparison to users scanning QR codes.

Strong security and usability properties can be achieved if users execute the manual verification correctly. Yet, the difficulty of having users do the corresponding tasks correctly while finding the right balance between usability and security is the root cause of security drawbacks, which have been amply discussed by research on fingerprint and SAS comparison via OOB channels (e.g., [105, 103, 170]). Not surprisingly then, usability studies encourage the replacement of manual comparisons by automated software whenever possible [170].

Some of the problems with OOB authentication are the following.

Selection of an adequate OOB channel. OOB (a.k.a. empirical) channels that provide authenticity and data integrity (but not confidentiality) can be classified according to the assumptions made about the adversary. Here we consider Nguyen and Roscoe's classification [134].

While face-to-face conversations provide a strong authenticated channel—where messages cannot be forged, delayed, removed or blocked by the intruder—they are often impractical or not viable, especially if we consider the ubiquitous nature of email. This situation is disadvantageous for instance for authentication methods that depend on users interacting with their devices, such as scanning a QR code from the peer's screen.

A more common selection involves the use of voice-based channels, e.g. a phone call. This instances provide a weak empirical channel, where messages cannot be forged, but they can be blocked, overheard, delayed or replayed. However, some already consider voice-based SAS comparison to be obsolete from a security perspective [179] as nowadays messages can be forged by voice synthesizers with a small sample of the victim's voice. Indeed, a study regarding a voice impersonation attack executed on users comparing PGP words reported that users were not able to distinguish the fake voice in about 50% of the cases [162].

Social engineering attacks. Although there are multiple options for users to interact via OOB, little effort has been put in designing precise protocols for humans to carry out the authentication process in privacy-preserving and fair ways. This situation offers the opportunity to break down a secure communication by misleading users rather than by finding technical vulnerabilities. For instance, assuming that $\mathcal{A}$ and $\mathcal{B}$ use p$\equiv$p, $\mathcal{E}$ could have $\mathcal{A}$ trusting $\mathcal{E}$'s own public key under the identity of $\mathcal{B}$ with a simple attack, even without knowing the trustwords to be compared: (1) $\mathcal{E}$ calls $\mathcal{A}$ pretending to be $\mathcal{B}$ and asks $\mathcal{A}$ to read her words, (2) $\mathcal{A}$ reads the complete list, (3) $\mathcal{E}$ says that the words match, (4) $\mathcal{A}$ confirms to the system the match and in consequence (5) $\mathcal{B}$ appears from now on as an authenticated contact for $\mathcal{A}$ but the trusted key is the public key of $\mathcal{E}$.

Indifferent users. Here we refer to users that confirm an OOB successful authentication to the system without actually carrying out the process. A noted reason for users to skip contacting their peers for fingerprint verification, is that they cannot think why anyone would impersonate their communication partners [167]. Fully disregarding the authentication could in fact be preferable to users neglecting authentication steps or performing them wrongly, as the partner would not inaccurately appear authenticated in the system. These situations could be studied further within the framework discussed in Chapter 7.

Inattentive and lazy users. Users misreading the authentication value (inattentive) or comparing only parts of it (lazy). In a recent paper, Naor et al. [131] analyzed approaches based on SAS authentication and identified vulnerabilities associated with a lazy user behavior. For example, WhatsApp, Signal and Telegram construct the authentication string by concatenating the two peer's fingerprints, thus, their approach would be subject to MITM attacks if users compared only either the first or the second half, given that the comparison would be verifying only the fingerprint of one of the peers. To fix this situations, the authors propose an influence spreading technique in which every bit of the values to be authenticated—in this case, each peer's fingerprint—influences the generation of each element of the SAS value—the string shown to the user, which combines both fingerprints.

Partial preimage attack. Studied by Dechand et al. [60], this attack aims at finding a partial preimage for a fingerprint assumed to be verified by lazy users. They estimate the number of brute-force steps that an adversary would have to carry out to find, with more than 50% success probability, an input such that the output would match with the user's fingerprint only in a subset of the totality of bits. Based on several studies, they assume that users always check the bits at the boundaries, plus variations of subsets in the middle of the SAS. We explain next the argument behind the estimation.

Let p denote the probability of finding a partial preimage for a given fingerprint f and q its complementary event. To calculate $p = 1 - q$, we work out q (i.e., the absence of partial preimages for a specific bit permutation). Let b be the length of the fingerprint f and assuming that r consecutive boundary bits are fixed (checked by the user), in this case, the leftmost and rightmost bits of f, we let ℓ denote the number of remaining bits in the middle from which a possible variation of u bits could be fixed, i.e., checked by the user. Thus, we have $2 \cdot r + u$ fixed bits that the adversary cannot invert without the user noticing. Valid preimages can thus be obtained by flipping up to $t = \ell - u$ bits within the middle bits; by removing these from the total space of size 2^b, we obtain the number of invalid ones. With k denoting a given number of positions to modify, the valid strings are then given by $\binom{\ell}{k}$

choices of positions to flip. Thus, q is given by

$$q = \frac{2^b - \sum_{k=1}^{t} \binom{\ell}{k}}{2^b}. \tag{1}$$

Expressing p as a function of the computational effort in terms of x brute-force attempts, we have $p = 1 - q^x$. Thus, to estimate the number of steps needed for finding partial preimages with a success probability $\geq p$, we simply compute

$$x = \log_q(1 - p). \tag{2}$$

Expressing x in base 2 gives results comparable to those in [60].

9.2.1 Analyzing OOB authentication in p≡p

The authentication method based in comparing either fingerprints or words, implemented in p≡p, suffers from most of the vulnerabilities discussed above. We will elaborate on the partial preimage attack as it is the only one requiring further explanation.

Recall that p≡p's trustwords are generated by a deterministic algorithm that runs locally taking as input the public key of a peer—obtained by email—and the user's own public key (see Section 6.4). Roughly, the algorithm performs an XOR over the PGP fingerprints of each of the input keys, and then maps each block of 16 bits from the resulting 160-bit long string to a word in a predefined dictionary of size 2^{16}, thus yielding a list of ten words.

To encourage users to perform the OOB authentication, p≡p shows by default only five out of the ten words[3]; this means that the peers compare at most the first 80 out of the 160 bits of a fingerprint, assuming that they check all the words. Since an "influence spreading" property, similar to that proposed by Naor et al., is already present, the best adversarial strategy is a brute-force attack over the public key space. Then, the attack would require $O\left(2^{80}\right)$ steps to find a key k such that the first 80 bits of $\mathrm{fpr}(k)$ are equal to those of $\mathrm{fpr}(\mathrm{pk}_B)$, with pk_B being the public key of $\mathcal{B}$ and $\mathrm{fpr}(\cdot)$ the fingerprint function.

However, the complexity of the attack decreases significantly when we consider lazy users and compute estimates for partial preimage attacks according to Equation (2). We consider the two cases where, out of five words, the user verifies:

(i) the first and last words as well as two from the middle
(ii) the first and last words, along with one of the three in the middle.

Then, with $b = 80$ and $\ell = 48$, for (i) we have $u = 32$ and we get $x \approx 2^{38}$; for (ii) we have $u = 16$ and we get $x \approx 2^{32}$. These results show that $\mathcal{E}$ would succeed with costs equal to and lower than the computational power estimated for an average adversary [60].

Were an OOB comparison approach to be kept for authentication, this analysis urges to reconsider the decision of showing five words instead of ten by default. Although users might feel less annoyed by having to compare fewer words, its adverse effect on security is considerable as it practically renders brute-force attacks viable.

[3] Alternative options for comparing the full list of words and the fingerprint are also available.

9.3 Authentication in Email and Messaging via PAKEs

By using PAKE protocols for user authentication in secure email and messaging, one can relax the assumption regarding the existence of OOB channels that voice-based fingerprint comparisons need, and replace their ad-hoc properties with concrete cryptographic guarantees.

In this section we show how PAKE can be used to perform a secure equality test and thereby authentication, and why the PAKE-based approach yields a more efficient solution than the OTR protocol, with better security guarantees, and enables further cryptographic features.

For now, we assume that $\mathcal{A}$ and $\mathcal{B}$ share a low-entropy secret—e.g., a short phrase, a password—either agreed upon beforehand or decided by posing and answering a question at the beginning of the mutual authentication. Intuitively, $\mathcal{A}$ and $\mathcal{B}$ want to authenticate their public keys via a secure equality test of their secrets, π_A and π_B, without revealing any information about them, such that, upon termination of the protocol, $\mathcal{A}$ and $\mathcal{B}$ would only learn whether or not their respective secrets were the same, authenticating each other on the basis of knowing the same secret. Therefore, a zero-knowledge protocol is needed to guarantee that the resulting transcript of their exchanges does not leak any information on π_A and π_B. Also, it should not be possible for $\mathcal{E}$ to brute-force the password via offline dictionary attacks; thus, $\mathcal{E}$'s only strategy would amount to making online attempts.

9.3.1 Validation of public keys via PAKE

Once $\mathcal{A}$ and $\mathcal{B}$ have entered their secrets, a PAKE protocol is executed. To determine whether the user secrets π_A and π_B are equal at the end of the run, without revealing anything else, we suggest the enforcement of explicit authentication using key confirmation (KC) after the key establishment phase. While this step may be optional in the general case for PAKE protocols, here it would be necessary in order to bind the cryptographic material with an identity. The information that $\mathcal{A}$ and $\mathcal{B}$ wish to authenticate—e.g., public keys for email addresses in p≡p or key fingerprints for phone numbers in Signal—can be incorporated either into the KC phase or into the initial user secrets.

Next, we exemplify with a concrete PAKE protocol how this can be constructed. For the moment we do not focus on engineering aspects related to (a)synchronicity and message transport mechanisms, but we will come back to these in Section 9.3.4. The literature contains several well-studied instances of PAKE, therefore, we first pick a candidate to demonstrate how it can be used to validate public keys, and then compare a few prominent schemes according to specific properties of interest.

9.3.2 Entity authentication based on SPAKE2

We propose an extension of the one-round protocol SPAKE2 [4] with a KC step, to achieve explicit authentication, thus binding a public key to an entity. This yields a 2-round scheme, which is the minimum when KC is enforced (see [107] for optimal-round PAKEs). For KC we can use the generic refresh-then-MAC transformation—despite its long history and popularity, this transform was only recently proved secure [78].

The scheme works as follows. With $\mathbb{G}$ being a finite cyclic group of prime order p, generated by an element g, let $\mathbb{G}, g, p, M \leftarrow_{\$} \mathbb{G}, N \leftarrow_{\$} \mathbb{G}$ and a hash function $H(\cdot)$ denote public parameters and $\pi \in \mathbb{Z}_p$ the private low-entropy secret, with the user password assumed to be appropriately mapped to an element in $\mathbb{Z}_p$[4]. The parties perform the key exchange phase as shown in Figure 9.1, which concludes with the generation of a symmetric key, sk_A and sk_B for $\mathcal{A}$ and $\mathcal{B}$ respectively.

[4] E.g., by hashing the password to get the appropriate number of bits, and then getting the remainder modulo p.

Upon termination of the key establishment, $\mathcal{A}$ and $\mathcal{B}$ each use the symmetric key to carry out a key-refreshing step via a key derivation function KDF in order to generate fresh MAC keys (for both parties), along with a new session key, K, which will be the secret shared key. Next, under the freshly generated keys, they each compute a MAC on the fingerprints of both parties' public keys and a session id. The authentication now amounts to exchanging and verifying the obtained tags, τ^a and τ^b, match with the corresponding self generated tags.

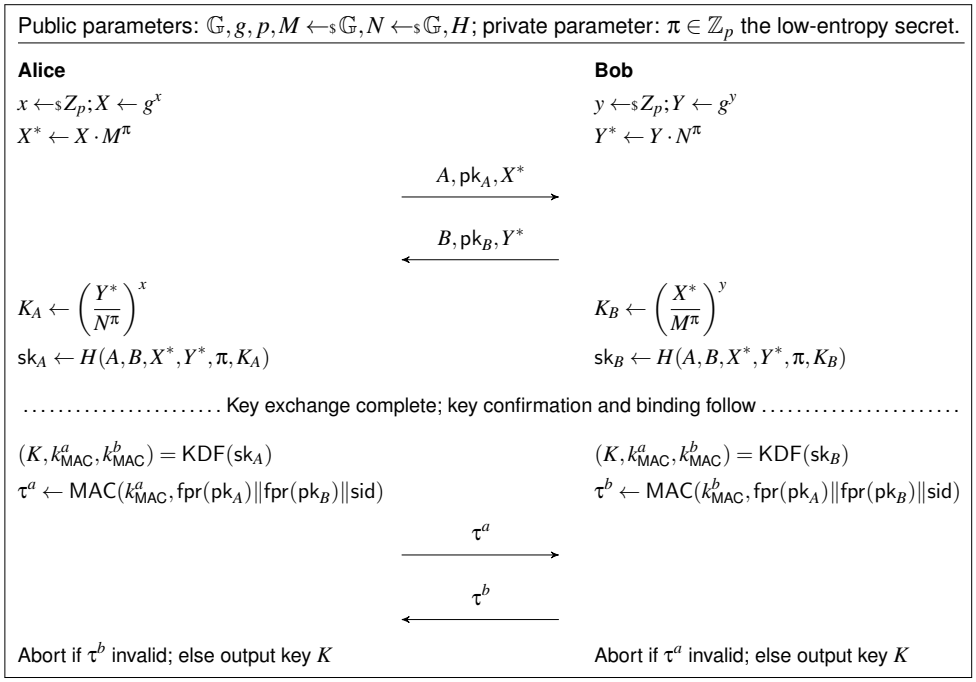

Figure 9.1: Public key authentication using SPAKE2 with refresh-then-MAC key confirmation for entity binding.

Alternatively, the public key fingerprints can be embedded in the secret π, but note that even so, the KC step cannot be skipped as an explicit authentication of the public keys would still be needed. In that case, we would let $\pi = \pi' \| \mathsf{fpr}(\mathsf{pk}_{\mathcal{A}}) \| \mathsf{fpr}(\mathsf{pk}_{\mathcal{B}})$, where π' denotes the original secret provided by the user, and we would compute the tags as $\tau^x \leftarrow \mathsf{MAC}(k^x_{\mathsf{MAC}}, \mathsf{sid})$, where $x \in \{a, b\}$ and the identifier sid is computed over the transcrip. Similar one round KC methods for explicit authentication can be found in IETF internet-drafts for SPAKE2[5] and J-PAKE[6].

The addition of the KC step increases the number of rounds and flows to 2 and 4, respectively. Note that this is merely an illustrative example and as already mentioned, other possibilities for KC do exist, some of which offer additional properties. For instance, Becerra et al. [21] showed that a modified version of SPAKE2, called PFS-SPAKE2, coupled with a KC step can achieve PFS at the cost of increasing the number of rounds from 1 to 3. More recently, Abdalla and Barbosa [3] showed that SPAKE2 does indeed satisfy PFS even without KC under a different hardness assumption. They also prove a version with a KC step (yielding a better bound) almost identical to the one given in Figure 9.1, except that the protocol has one less flow. We elaborate on PAKE protocols and their properties in the next section.

[5] https://tools.ietf.org/id/draft-irtf-cfrg-spake2-08.html
[6] https://tools.ietf.org/html/rfc8236

Table 9.1: Comparison of PAKE protocols.

Protocol	Rounds/ Flows	KC	Forward secrecy	Security model	Hardness assumption
SPAKE2	1/2	✗	✓	ROM	CDH
PFS-SPAKE2	3/3	✓	✓	ROM	CDH
OPAQUE	2/3	✓	✓	ROM	OMDH
J-PAKE	2/4	✗	✓	ROM-AAM	DSDH
KV-SPOKE	1/2	✗	-	CRS	DDH
RLWE-PAK	3/3	✓	✓	ROM	RLWE
RLWE-PPK	2/2	✗	✓	ROM	RLWE

ROM: Random Oracle Model; AAM: Algebraic Adversary Model; CRS: Common Reference String
DH: Diffie-Hellman; CDH: Computational DH; DDH: Decisional DH; DSDH: Decision Square DH; OMDH: One-More DH; RLWE: Ring Learning With Errors

9.3.3 Selecting a PAKE protocol

PAKEs are typically evaluated according to the security model in which they are proven secure, the number of rounds required, whether they support forward secrecy, and their communication and computational complexity. The complexity related aspects are more relevant in a client-server setting wherein a server has to process a high number of requests and sessions; in a decentralized peer-to-peer setting such properties no longer play a major role.

Conversely, minimizing the number of rounds becomes important in the asynchronous setting of email, as it is a strong factor in the selection of the transport mechanism. This might be equally relevant for secure messaging solutions that do not operate in a purely decentralized and peer-to-peer setting (e.g., Signal or OTR4) in which one may wish to reduce the load on relay or buffer servers; but the number of rounds would in general be arguably less of a concern than in email. As for KC, remark that this step can be added to schemes that do not have it by default at the cost of an extra round.

In Table 9.1, we compare relevant properties of a number of representative PAKE protocols: SPAKE2 [4], PFS-SPAKE2 [21], OPAQUE [99], J-PAKE [93], KV-SPOKE [107], RLWE-PAK and RLWE-PPK [67]. Note that except for RLWE-PAK and RLWE-PPK that make use of lattice-based cryptography, all the other referred schemes are Diffie-Hellman-based. In terms of post-quantum (PQ) security, an immediate implication of this observation is that the latter cases would not be safe against quantum adversaries, whereas the first two would provide conjectured quantum-security given that, until now, there is no knowledge of an efficient quantum algorithm for solving the RLWE problem underlying their hardness assumption.

Regarding the security models, in the random oracle model (ROM), an ideal truly random function being accessible to the parties through oracle calls is typically instantiated using cryptographic hash functions. The common reference string (CRS) model implies the accessibility of a random string, generated in a trusted way, to all parties; due to the constraints of decentralization in our case, relying on a TTP for this task is not an option; however, there are approaches, e.g. based on secure multi-party computation protocols [160], for CRS generation in a decentralized setting. As for RLWE-based schemes, their proofs are unfortunately in the ROM, as opposed to the quantum ROM (QROM), and therefore the capabilities of a truly quantum adversary are not captured.

9.3.4 Transport Mechanism

We consider two feasible ways for implementing the exchange of protocol messages.

Email-based approach. Given the small number of rounds required by PAKE protocols, in the case of email we can afford to use standard attachments as messages' carriers, or alternatively, designated emails with a special format. In both cases, the delivered message would be automatically processed by the email client in the background.
For the case of attachments, a PAKE-based implementation could give $\mathcal{A}$ the option to enter her secret $\pi_{\mathcal{A}}$ upon sending her first email to $\mathcal{B}$, allowing thus the first flow of the protocol to occur via an attachment; similarly, when $\mathcal{B}$ replies, if he opts for entering his secret $\pi_{\mathcal{B}}$, the initial PAKE round would be completed. The subsequent exchange for the KC step can be done automatically by the implementation.
Alternatively, when resorting to a specific-email transport model—such as the one used by p≡p for multi-device key synchronization—the implementations would encapsulate cryptographic messages in specially crafted emails, kept hidden from the user (e.g., archived separately) and processed automatically. Since we primarily deal with authentication, our proposal would have minimal impact in terms of communication and computational complexity as it would have to take place only once.

Untrusted server approach. Although early instant messaging tools were entirely online services that maintained an active session for each conversation, modern messaging tools are in fact quite similar to email in that the underlying system follows an asynchronous model. Both Signal and the latest version of OTR [136] achieve offline messaging by using "buffer servers" for hosting pre-key bundles that can be fetched without the other party being online.
We can use a similar mechanism to overcome transport engineering obstacles in email more elegantly, since all aspects related to the exchange of emails remain unchanged and thus interoperable. In fact, the use of an intermediate server would not introduce additional trust assumptions as the transcript of a PAKE protocol does not leak useful information to the adversary; such a server would be untrusted and any entity would be able to set up their own instance.

9.4 Enhancements to Secure Email and Messaging by PAKE

Authentication via PAKE satisfies and improves a number of key properties related to security and usability that have been identified in the literature [179]. We first show how these properties are satisfied and then introduce novel uses of PAKEs in secure email and messaging.

While we primarily focus on enhancements for existing paradigms that depend on public-keys to secure email and messaging—e.g. PGP-based or OTR-inspired systems such as Signal—we also consider possibilities for shifting to entirely symmetric-key solutions. Note that once a PAKE-generated key is established, subsequent PAKE instances can be run automatically via a chaining self-sustaining mechanism. Indeed, once a PAKE-generated shared symmetric key has been established, not only a wide range of well-understood techniques become possible, but one could also consider the benefits of transitioning to symmetric-key constructions, e.g., MAC-based authentication and symmetric-key encryption schemes.

9.4.1 Usability enhancements

These improvements are mostly related to key management automation and error resilience.

Automation of future key pair authentications. This feature facilitates the achievement of some of the subsequent properties. Once authentication between $\mathcal{A}$ and $\mathcal{B}$ is bootstrapped from an initial PAKE, the authentication of new key pairs from either $\mathcal{A}$ or $\mathcal{B}$ can be automated using as input the PAKE-generated shared key without prompting the users to yet again enter new secrets.
Authentication due to key pair generations can be triggered for instance when keys expire, as a new key pair needs to be associated with an existing identity. Furthermore, new email accounts that $\mathcal{A}$ adds to the PAKE-based solution, and the corresponding key pairs, could be automatically authenticated to $\mathcal{B}$ by running a PAKE with the stored shared symmetric key as input. Note that each future execution of a PAKE refreshes the stored PAKE-generated symmetric keys.

Immediate enrolment. This property refers to a user's keys being reinitialized in such a way that other parties can verify and use them immediately. The PAKE-generated key allows to automate the new key exchange and the corresponding authentication as explained above.

Alert-less key renewal. Complementing the previous property, this one refers to users not receiving alerts or warnings prompting them to take action when other parties renew their public keys. This would be automated similarly to immediate enrollment.

Low key maintenance. This property pertains to the amount of user effort required for maintaining keys, e.g., tasks such as signing keys, renewing expired keys. For instance, while the p≡p client does automate key generation and renewal, the established trust level disappears with every key refreshment; key maintenance can be improved with PAKEs as explained above.

Inattentive user resistance. As discussed earlier, manual OOB key/fingerprint verification methods are susceptible to human error and inattentiveness. In the PAKE-based approach, even if the users enter the wrong value, the result would not be as catastrophic as trusting a key prepared by the adversary. At worst, it would be inconvenient as the authentication would fail prompting the user to eventually repeat the process.

9.4.2 Security enhancements

These refer to cryptographic properties that are mainly enabled by the derived symmetric key.

Perfect forward secrecy (PFS). In the context of authentication, PFS means that in the event of a password disclosure, previously derived cryptographic keys remain secure. In other words, an adversary learning the low-entropy password cannot use it to compute high-entropy shared keys derived before the password compromise.
In the case of a symmetric key disclosure, an idea for minimizing the impact of such a compromise consists in implementing a PAKE-chaining mechanism to automatically perform key rotations and periodically refresh the symmetric key; this would provide limited time-windows for $\mathcal{E}$ to compromise the channel, at the end of which, the key would be refreshed and secure again, leaving $\mathcal{E}$ with an expired key. If there is evidence that $\mathcal{E}$ has corrupted the channel, the cryptographic key would have to be discarded and replaced via a PAKE executed with a low-entropy secret. This refreshing paradigm might be expensive, however it would be relevant when PAKE-based approaches are used for synchronization purposes, either device-to-device or device-to-server, where PAKE can be used to both authenticate and establish a secure channel, thus providing PFS for the session keys used for syncing.

Several PAKE constructions provide PFS by default, some of which are listed in Table 9.1; moreover, PFS can be obtained by adding explicit authentication via a KC step to constructions that do not have this property [26]. Alternatively, to improve efficiency we could resort to symmetric-key schemes that provide PFS, e.g. SAKE [12]. In this case, a PAKE can be used once to bootstrap authentication via a low-entropy secret and to generate the initial symmetric high-entropy key—known as master key—required by SAKE.
The use of PAKEs could for instance improve the approach based on regular sub-key rotations, adopted by the Sequoia-PGP project for adding FS to OpenPGP-based solutions; a PAKE-based solution could automate authentication in case that the master key, certifying the short-term sub-keys, needs to be refreshed. For additional security, with slightly hampered usability, a separation of storage can be enforced by for example storing such PAKE long-term keys in dedicated hardware, e.g., hardware security modules or smart key storage devices such as YubiKey or Nitrokey, to protect against a device compromise; see Section 9.4.4 for more details on this.

Deniability. This is another subtle and fundamental property that has been of particular interest in recent secure messaging systems such as Signal and OTR. Deniable exchange, applied to tasks ranging from authentication to encryption, has a long and somewhat controversial history due to the subtleties in various existing security definitions. We limit ourselves to the case of key exchange and the seminal framework of Di Raimondo et al. [63], which provides security definitions in the simulation paradigm for deniable key exchange and authentication, where message and participation repudiation are considered as requirements.
We conjecture that sender/receiver deniability for non-augmented (symmetric) PAKE—e.g., where $\mathcal{E}$ wants to deny having sent a message to $\mathcal{B}$—would satisfy the said definition of deniability in the symmetric-key setting: in a two-party setup, a malicious party $\mathcal{E}$ would not be able to produce binding cryptographic proofs from communication transcripts, associating another party (here $\mathcal{B}$) with a particular exchange, as all exchanges could have been simulated by the accusing party $\mathcal{E}$. More specifically, a simulator in Di Raimondo's framework can be constructed given that π is the only private input shared by both parties and all other parameters are public and drawn at random.
Finally, assuming composability, using the PAKE-generated key with symmetric ciphers and MAC-based authentication would preserve deniability. Clearly, this and other forms of deniability for PAKE need to be studied rigorously in future work.

Post-quantum security. As pointed out in Section 9.3.3, in the event that secure messaging and email tools transition to PQ cryptography, there are already candidate PAKE constructions that provide conjectured PQ security (e.g. in Table 9.1). Moreover, the recent symmetric-key authenticated key exchange (SAKE) by Avoine et al.[12] is conjectured to be PQ-secure due to its use of symmetric-key primitives. Thus, a quantum-resistant PAKE can be combined with SAKE, to obtain a low cost and efficient PQ-AKE with PFS suitable for settings with limited computational power.

As a remark, the OTR application provides deniability of messages and FS, however, such features are independent from the SMP solution for authentication; they are implemented separately by constructions such as MAC tags in the messages. In the case of PAKEs, these properties are inherent to the schemes.

Table 9.2: Comparison of trust establishment approaches.

Paradigm	Example	Security									Usability									Adoption		
		Network MitM Prevented	Operator MitM Prevented	Operator MitM Detected	Operator Accountability	Key Revocation Possible	Privacy Preserving	Deniability Facilitated	Forward Secrecy Facilitated	Post-quantum Security	Automatic Key Initialization	Low Key Maintenance	Easy Key Discovery	Easy Key Recovery	In-Band	No Shared Secrets	Alert-less Key Renewal	Immediate Enrollment	Inattentive User Resistant	No Service Provider	Asynchronous	Multiple Key Support
Web of Trust	PGP	●	●	●	◐	◐	✗	✗	-	✗	✗	✗	◐	◐	✗	✗	✗	✗	✗	●	●	●
KD + SaL	CONIKS	●	✗	◐	●	●	●	✗	-	✗	●	●	●	●	●	●	●	●	●	●	●	●
OE + TOFU	TextSecure	◐	◐	◐	◐	✗	●	-	-	-	●	●	●	●	●	●	✗	●	✗	●	●	✗
OE + TOFU + OOB	p≡p	◐	◐	◐	◐	◐	●	-	-	-	●	◐	●	◐	✗	●	✗	✗	✗	●	✗	✗
OE + TOFU + SMP	OTR	◐	◐	◐	◐	●	●	-	-	✗	✗	●	✗	✗	◐	◐	✗	●	✗	●	✗	●
OE + TOFU + PAKE	-	◐	◐	⊛	⊛	◐	●	⊛	⊛	⊛	●	●	●	●	●	✗	●	●	●	●	●	●
KFV: OOB	SilentText	●	●	●	●	◐	●	-	-	-	✗	✗	✗	✗	✗	●	✗	✗	✗	●	✗	✗
KFV: PAKE	-	●	●	●	●	◐	●	●	●	⊛	●	●	●	●	●	✗	●	●	●	●	●	●

Paradigm property is: ● = satisfied; ◐ = partially satisfied; ✗ = not satisfied; ⊛ = implementation dependent; - = N/A
KD = Key directory; KFV = Key fingerprint verification; OE = Opportunistic encryption; SaL = Self-auditable logs; TOFU = Trust-on-first-use

9.4.3 PAKE-based security and usability compared to the state-of-the-art

Table 9.2 shows a comparison of our proposal with a select set of approaches for trust establishment, extracted from Unger et al.'s survey on secure messaging [179]. We limit our analysis of Table 9.2 to relevant aspects for the comparison and refer the reader to the cited source for a more detailed explanation of the approaches and their properties. If the reason behind a given evaluation is not specified in [179], we provide our own interpretation and evaluate our approach accordingly.

Most of the properties have self-explanatory names, except perhaps *operator accountability*, which is considered to be satisfied if the paradigm provides support for verifying the correct behavior of service providers during the trust establishment process, when a centralized infrastructure is required. The network and operator attackers considered for MITM refer respectively to adversaries controlling large segments of the internet and infrastructure operators (service providers).

PAKE-based approaches are *privacy preserving* as the transcript of a PAKE execution does not leak information. *Deniability facilitated, FS facilitated* and *post-quantum security* are subject to the selection and exact usage of the PAKE scheme.

Approaches built upon opportunistic encryption (OE) partially provide MITM prevention because an attack can be successful during the initial communication round, before a key is authenticated. When OE is combined with SMP, *operator accountability* and *MITM detection* are also partially satisfied given that if the execution of the SMP protocol fails, the users do not learn whether the failure was due to mismatching passwords or an adversarial attempt at compromising the channel. However, when it comes to our PAKE-based approach, these last two properties could be potentially satisfied with the use of auditable PAKEs (see section 9.4.4), mainly in the context of messaging.

It is somewhat ambiguous as to why the authors of [179] consider *key revocation*—users being able to revoke and renew keys—to be fully satisfied for SMP applied to OE. While revocation is possible, the process would still suffer from the known limitations of a truly decentralized setting, e.g., informing all users of an expired key. The latter is indeed stated to be the reason for considering KFV

approaches to only partially satisfy this property. Under this interpretation, PAKE applied to OE would also partially satisfy key revocation. Thanks to the derived cryptographic key, the main advantages of OE with PAKE can be observed at the level of usability related properties, e.g., automation of tasks.

In key fingerprint verification (KFV) approaches, the verification is considered to occur before using the public keys, which leads to achieving most of the security properties. The evaluations for the OOB approach assume that the manual comparison is executed correctly; this assumption is not needed for SMP or PAKE. As we can observe, PAKE-based KFV significantly improves usability compared to OOB and SMP fingerprint verification.

Key directory combined with self-auditable logs (KD+SaL) is arguably the most promising approach identified by Unger et al. due to the wide range of properties that it provides. It allows users to efficiently verify the consistency of their own entry in a central key directory (which can be self-hosted) and therefore to detect and expose misbehavior of a trusted third party. The set of properties that KD+SaL and KFV:PAKE can achieve is similar, yet, the latter has the advantage of enhancing security with the properties discussed in Section 9.4.2.

Overall, PAKE-based key fingerprint verification offers the most complete set of properties with reasonable trade-offs between security and usability in a purely decentralized setting.

Clark et al. [50] present a similar table evaluating primitives used to enhance email security. Considering end-to-end encryption as a baseline, PAKE-based key verification/management would perform as shared secret key verification (R14 in [50]), except that, additionally, our PAKE-based approach partially satisfies the property that refers to providing support for server-side content processing (P12) as this can be enabled without exposing the encrypted content, e.g., via secure secret retrieval (see section 9.4.4).

9.4.4 Additional novel enhancements

The uses of PAKEs for securing email and messaging go beyond entity authentication. Here, we discuss some areas that could benefit from the use of these schemes.

Multi-device synchronization

A quite natural application of PAKE is in the realm of device pairing and multi-device synchronization, where the goal is to create an authenticated and private channel between devices, usually from the same user, for transferring information securely.

Although typical solutions rely on a human interactive security protocol (HISP) and OOB channels, thus requiring manual intervention (which can give rise to new and subtle attacks), the application of PAKEs for device pairing in other contexts has been considered before [111]; it is thus natural to consider its use in multi-device syncing of email and secure messaging systems, for instance, to synchronize the user's keys for encryption and keys of trusted contacts.

A secure email solution can display a screen in each of $\mathcal{A}$'s devices to be paired, D_1 and D_2, so that after $\mathcal{A}$ enters a password in both, a PAKE protocol that establishes a secure channel between them for synchronization is triggered. Alternatively, this process can even be done asynchronously, i.e., without the two devices being online: D_1 pushes its state (e.g., key store, chat or email archive) to a server in encrypted form and later D_2 retrieves the secrets stored on the server in an oblivious manner w.r.t. the server. We discuss this further in the following part regarding secure secret retrieval.

The application of such a PAKE-based solution could for instance bring benefits to the current implementation of p≡p, which resorts to an ad-hoc pairing technique for key synchronization based

on OOB comparison of SAS. The established trusted channel could be used not only for sharing key material but also contact lists, calendars, etc.

Secure Secret Sharing and Retrieval

OPAQUE is a recent construction that, among other things, serves as an aPAKE to offer protection against breaches and server password file compromises. It also offers a secret retrieval mechanism based on oblivious pseudo-random functions, to fetch from a server a secret stored in encrypted form, using only a low-entropy password.

This feature is inspired by the notion of password-protected secret sharing (PPSS) schemes formalized by Bagherzandi et al. [13], which are (t,n)-threshold constructions wherein security is preserved against an adversary controlling up to t servers out of n. A problem that PPSS addresses is protecting $\mathcal{A}$'s secret data d (e.g., a secret key used for decryption, authentication credentials, crypto-currency wallet key, etc) in the event of a device compromise or failure.

An implementation of PPSS would secret-share d among a set of n entities so that only a collusion of more than t corrupt ones would compromise the data. A password-based mechanism would allow the authentication of the owner of d to the secret-share holders in order to trigger a reconstruction protocol and then retrieve the secret. The private storage of d can be shared among n external network entities; alternatively, if $\mathcal{A}$ does not trust external entities, her device can instead partake in the secret-sharing by storing multiple shares, thus preventing online dictionary attacks by a network attacker and not allowing $\mathcal{E}$ to learn anything about the secret without corrupting $\mathcal{A}$'s device.

Secret retrieval would have several use cases in secure messaging. For instance, a general anonymity/privacy related criticism directed at messaging services has to do with the identification of users via their phone numbers. This can be dealt with by securely storing long-term identities in encrypted form on the server, accessible only to the users. Servers could also store per user lists of contacts in encrypted form; this would enable asynchronous syncing of contacts across multiple devices without the service provider learning the content.

Another use case would be to secret-share user data among several of their own devices, e.g., smartphone, laptop and tablet, so that a device compromise would not provide any useful information to an attacker; this can also be used for performing key synchronization among multiple devices. All these mechanisms would work in a similar manner from the user's point of view, i.e., simply by providing a password.

Recently, Signal was enriched with a functionality referred to as "Secure Value Recovery" [163]. Among other things, the design involves a key stretching of a user's PIN along with a master key derivation using the stretched key and a piece of server-side stored randomness. The same core functionality can be achieved with the use of either PPSS or PAKE constructions such as OPAQUE for secure secret storage and retrieval. Signal's developers also mention secret sharing and oblivious pseudo-random functions as future possibilities [164], both of which can be achieved using existing cryptographic primitives, as explained in this section.

Auditable PAKEs for Thwarting Online Guessing Attacks

Online guessing attacks are unavoidable when using password-based authentication. This is usually dealt with by fixing a limit on the number of failed attempts that can be tolerated before invalidating a password.

However, resorting to guess and abort style attack, an adversary could sidestep the (at most) one online test per run: $\mathcal{E}$ intercepts a message at a crucial step of a protocol run—e.g., in Figure 9.1, any of the messages with the MAC-tags τ in the KC step; then $\mathcal{E}$ uses the information to verify a chosen guess at the password; in case of an incorrect guess, $\mathcal{E}$ drops the intercepted message to disguise the guessing attempt as a network communication failure. This can be done in both directions to double the chance of discovering the password, or in parallel against many network nodes depending on the setting. Such an attack can be carried out repeatedly without raising an alarm as the honest parties may simply view this as a network failure.

The modified version of SMP in OTR (Section 9.1.4) is vulnerable to this attack: just before the last phase where the parties perform their secure equality test, when $\mathcal{A}$ and $\mathcal{E}$ exchange their blinded DH terms incorporating the low-entropy password in the exponent, i.e., $(g_3^a, g_1^a g_2^{\pi_{\mathcal{A}}})$, $\mathcal{E}$ could make a guessing attempt at $\pi_{\mathcal{A}}$ and in case of obtaining a value other than 1 (which reflects equality) in the verification, drop the message and force an abort. Note that the non-interactive zero-knowledge (NIZK) proofs that are attached to the messages at every exchange are not meant to protect against this type of attack.

In a relatively recent work [155], Roscoe developed a mechanism based on commitment schemes and delay functions (e.g., timed-release encryption) for protecting against online attacks in HISPs. Later, Roscoe and Ryan [156] combined this idea with a new technique that achieves a so called *stochastic fair exchange*, for making PAKEs auditable, i.e., such that legitimate participants of the protocol can distinguish online guessing attacks from network failures.

Roughly speaking, they propose a transformation in the KC phase of a PAKE, which combines blinding, randomization, commitments and delay functions; the transformation is such that a series of messages consisting of fake ones and the real intended message are exchanged and the parties will only get to know which is the *right* one until their exchange is complete. In a follow-up work [55], Couteau and the previous authors generalize this result to achieve ε-fair exchange using oblivious transfer and timed-release encryption.

The discussed transformation can be used to enhance any PAKE with auditability, hence, allowing the authentication process to achieve such a property. An important limitation here is that, due to the highly interactive design of the solution, it would be more suitable to the setting of secure messaging than to email, unless a given email solution were to opt for untrusted buffer servers for transport, as described in Section 9.3.4.

Finally, note that some of the ideas in this transformation, specifically those related to enforcing fairness, have common elements with the original SMP solution [44] aimed at providing fairness, a property that was removed from the modified version of SMP used in OTR on account of achieving efficiency.

9.5 Security and Low-Entropy Secrets

The schemes considered thus far have been proven secure under different models and assumptions (see Table 9.1). The security properties that they grant can be traced back to the core properties of PAKEs:

- they fulfill the role of ZK proof of knowledge schemes, and so, a run of the protocol does not leak any information on the password and upon termination only reveals whether the secrets were equal
- they resist offline dictionary attacks, and online ones typically by limiting active adversarial

attempts to one password per run

- compromised session keys will not compromise the security of other session keys
- depending on the choice of PAKE, FS would ensure that past session keys remain secure if the password is leaked.

A way for $\mathcal{E}$ to gain knowledge about the secret would be by managing to increase the number of active online guessing attempts. We discussed how making PAKEs auditable can be used to mitigate this class of attacks by distinguishing between failed adversarial attempts and network failures, minimizing the adversary's tries to one, under the assumption of correct input entry by honest users.

A disadvantage of the use of PAKEs in client-server scenarios concerns the need to store plaintext passwords in the server. This issue is irrelevant here since the passwords are used just once to bootstrap authentication and security of the system, after which they can be discarded.

9.5.1 Low-entropy secret agreement.

From a practical perspective, we need to think further about the assumption of $\mathcal{A}$ and $\mathcal{B}$ sharing a low-entropy secret. As already discussed by Alexander and Goldberg [9], the users can either share a secret over a secure channel, e.g. OOB, or agree on one via an in-band functionality without revealing sensitive information about the secret itself, for instance, $\mathcal{A}$ could ask $\mathcal{B}$ to use the nickname of a friend in common.

Note that in the case of in-band agreement, although the question itself could serve to authenticate both parties, this is not necessarily the case and therefore, instead of mutual authentication, only $\mathcal{B}$ should be authenticated to $\mathcal{A}$ when $\mathcal{A}$ poses a question and vice versa. Concretely, let us assume that Eve is a common friend of $\mathcal{A}$ and $\mathcal{B}$, and that $\mathcal{A}$ asks $\mathcal{B}$: "What is the nickname of Eve?". This question already implies that the one asking knows Eve and her nickname, so it is likely to come from $\mathcal{A}$; but it is also likely for Eve to know this information, so by $\mathcal{B}$ posing an adequate question to $\mathcal{A}$, $\mathcal{B}$ could detect a MITM.

Another possibility to agree on a secret would be to use an already authenticated and secure channel. For instance, given the widespread use of messaging tools such as Signal, the parties could use it to agree on a secret for a one-time entity authentication of their secure email solution. While this might be futile from a theoretical point of view, due to the assumption of there being an already authenticated and secure channel, practically speaking, this approach would in fact provide a realistic and usable solution since email and messaging have different use cases.

Usability considerations. Providing users with an adequate interface for entering the low-entropy secret is essential for solutions implementing the PAKE-based approach, in addition to documentation with accessible explanations for users.

A lesson learned from a usability study on the OTR/SMP tool [167] stresses the need for further research on how to guide users towards establishing a secure shared human-memorable secret. Here we briefly discuss some ideas.

Primarily, the interface could warn users not to include the secret itself; this could work similar to standard warnings in many email clients, that remind users to attach documents when they have not done so, and attempt to send a message in whose body the intention is mentioned.

For the in-band agreement, adding a list pre-populated with questions might serve as a guide for users to conceive similar questions, or reduce their effort by allowing them to choose one from the list; the questions should not lead to evident answers or to answers belonging to very small known

sets, such as “yes/no” or colors, as such cases increase the probability of the adversary to guess successfully.

A measure aimed at dealing with disparities due to letter cases would be to just convert the secret to upper-case, at the cost of reducing entropy. To minimize mismatch due to typos, we could resort to comparing the homomorphic distance of words.

Finally, recall that the in-band agreement in principle could be a solution for people that do not know each other. But as mentioned before, defining ways for users to achieve an adequate password selection is clearly a required line of work which we leave as an open question.

9.6 Concluding Remarks and Further Directions

In this chapter, we presented a PAKE-based solution to bootstrap peer-to-peer authentication, which would equally work in the context of secure messaging and in the inherently asynchronous setting of email. Apart from nullifying a bunch of vulnerabilities present in OOB-based protocols, PAKE-based authentication improves both, security and usability properties, maximizing the trade-off between them; furthermore, the PAKE-generated symmetric key can be used to introduce a series of enhancements in secure email and messaging.

From this point there are several research directions. We believe that a natural next step involves developing a formally verified implementation, with respect to specific security properties. This can be achieved for instance with the use of dedicated languages such as F* [151], which has been used, among others, for verifying a reference implementation of TLS 1.2 [29].

Since we primarily deal with authentication, implementing a PAKE-based solution would have minimal impact in terms of communication and computational complexity considering that authentication takes place only once per contact, as opposed to email encryption where protocol executions are required for every email exchange.

As previously mentioned, research on usability dedicated to assisting users for deriving low-entropy secrets while reducing the mental effort and the likelihood of mistakes, is also encouraged. Besides, there is large room for follow-up regarding theoretical work on all the suggested cryptographic enhancements and implementations thereof.

Other interesting directions include the application of PAKEs for authentication on encrypted mailing lists, and studying the possibility of sharing/synchronizing the trust in contacts across different services—e.g., from Signal to $\mathsf{p} \equiv \mathsf{p}$ or vice versa. In the latter, once an entity is trusted in one application, all other applications that recognize this entity could *inherit* the trust which is stored in the user’s device; evidently privacy plays a major role here and one would need to enforce the policies of the service with the strongest requirements.

Finally, although the assumption of users sharing a secret might seem restrictive, we believe that similar limitations occur when using OOB channels. For instance, parties that do not know each other might not have any other contact information than the email address itself, hence making difficult the establishment of a secure second channel. Moreover, one can argue that users verifying a secret gets closer to the idea behind entity authentication than them comparing meaningless strings, as these last ones are not linked to the users in any evident way. This, we reckon, affects the way in which the result of the authentication process is interpreted; after comparing words that match $\mathcal{A}$ might believe: “Ok, $\mathcal{B}$ and I have the same words”, while when a secret matches: “Ok, I am communicating with $\mathcal{B}$”.

10 Conclusions

Considerable effort has been put into securing email, the main means of modern communication, in the last decades. In particular, important advances have been made for preserving the privacy of the message exchanges over the internet using an intricate combination of public-key and private-key cryptography. Various solutions exist for centralized settings, wherein trusted authorities are responsible for certifying public-keys. However, PGP is the pervasive solution when it comes to decentralized settings; and due to fundamental complications rooted in poor key management and the process that ultimately allows users to authenticate each other, it has found limited adoption.

Thus, in the current landscape, the vast majority of people, outside institutional environments, do not use secure email solutions at all. Yet, the exchange of sensitive information via email, and the compromise of information exchanged, keep taking place as you read these lines. Therefore, developing usable, decentralized end-to-end secure email solutions, with proofs of security and potential for being widely adopted, is an evident necessity of significant importance.

In this thesis, we have designed and analyzed cryptographic protocols aimed at securing email communications, introduced formal frameworks and properties that allow a human-inclusive security analysis, and proposed cryptographic solutions for replacing the prevailing manual entity authentication approach, and for strengthening the protection of passwords in password-based authentication schemes, as they play a key role by enabling access to a user's mailbox.

Regarding end-to-end secure email protocols, we performed a symbolic formal analysis of the key distribution and the handshake protocols (p≡p protocol) underlying secrecy and authentication in p≡p, a software solution aimed at providing privacy mainly in email communication. In the process of achieving this objective, we were first confronted with the need for obtaining a precise specification of the protocols, implemented in a library programmed in C. Indeed, we realized that reverse-engineering code is rather frequently needed when modeling security protocols, mostly caused by the lack of documented protocol specifications. We worked out a method, documented in Chapter 3 along with a compilation of relevant state-of-the-art tools, for specifying the p≡p protocol; the technique is semi-automated and provides general guidelines specially for formal security analysts who might not necessarily be familiar with software engineering techniques.

At this point, we would like to stress the importance of documenting large scale software projects; in particular, documenting open source implementations could be beneficial for simplifying adoption and auditability, which is an important factor that contributes for finding flaws in security protocols. Indeed, the lack of auditability for proprietary solutions (e.g. Apple's protocols) has presumably raised concerns in the community; as the implementations are rarely disclosed, the alternative for the user is to trust in the security of the deployed products.

Based on the abstracted specifications, we modeled the p≡p protocol in the applied pi calculus and used ProVerif to execute a formal verification in the Dolev-Yao adversarial model. The analysis confirmed that the model satisfies six security properties: full agreement, trust-by-handshake, privacy-from-trusted, integrity-from-trusted, MITM-detection and confidentiality (Chapter 6). This result implies that exchanges between two p≡p users that have completed a successful handshake, are

private, authenticated and integral, as long as the authenticated public keys remain valid and are not compromised. The ProVerif code of our analysis is provided in the appendix.

Furthermore, we proved correctness and security of the trustwords generation algorithm (Section 6.4), which maps a fingerprint into a list of words shown to the user during the handshake process, so that the peers compare them through a secure out-of-band channel. A relevant characteristic of the mapping is its bijectivity, which eliminates collisions.

This part of the research largely answers our initial question, Q1, regarding whether decentralized implementations of end-to-end secure email solutions provide the essential security properties. To the best of our knowledge, this is the first proof of security that addresses a deployed secure email solution. This result enables "verified security", under the specific adversarial model and assumptions, to become an aspect to be considered when selecting a secure email tool, along with the usual ones: popularity and functionality.

To provide answers for Q2, regarding whether the interaction of a human with a system has an effect on the security of the latter and how one could study those situations, we defined a formal framework, which allows to model the path that a user follows to achieve a security goal in a system, together with the user's assessment and the system's assessment of predetermined security aspects at each stage. A model in this formalism is a variant of a transition system, which we call s-t transition system.

We selected Computation Tree Logic (CTL) to formalize a family of novel security properties, that we call misaligned-security properties, in terms of the discrepancy or agreement between the previously mentioned assessments. These properties: Aligned-AoS (aligned assesment of security), Lower-AoS, Higher-AoS and Misaligned goal, are representatives of classes that can be customized to study each specific system. The introduction of misaligned-security properties addresses Q3, which considers the definition of novel security properties, and enables a security analysis from a socio-technical perspective, shedding some light on vulnerabilities that emerge when a user has an inaccurate assessment of the degree of security that a system has.

Misaligned-security properties can be automatically verified in s-t transition systems by a model checker, NuSMV in our case. In particular, we identified a case where Aligned-AoS does not hold in $p \equiv p$; the root cause of the discrepancy is the use of an out-of-band channel for authentication and the absence of a rigorously defined procedure for carrying out the comparison process.

The whole formalism provides an approach for reasoning about security ceremonies depending on user beliefs, and enables the detection of vulnerabilities derived from sources other than purely technical ones. Since there are certainly many other ways for studying the human interaction in a security protocol, we consider our approach to be a possible answer to Q2.

A security analysis in this framework might be of special interest for other communities, in particular, usability and user experience (UX), as it might promote developments in their area of research. In fact, the analysis yields meaningful results when it is carried out in an interdisciplinary setting, as the usability techniques bring accuracy to the inputs regarding the user's perceptions and our framework provides the mathematical formalism and a technique for reasoning with them.

The computational, symbolic, and socio-technical security analyses mentioned above, complement each other, providing a framework for performing a multifaceted security analysis of cryptographic protocols, which in this case we carried out on the $p \equiv p$ software.

We covered Q4 by addressing three problems related to authentication that directly or indirectly

undermine the security of email:

1. An incredibly large number of credentials (in the order of billions) to access all kinds of web services (Yahoo being one of the most affected ones) has been leaked after password database breaches occurred in the last decade. This information has been freely distributed over the internet, and the corpus keeps becoming richer as database breaches keep taking place. So that users can at least change their passwords, to prevent unauthorized parties from accessing their account, a timely detection of a database leakage is imperative. The Honeywords System achieves such a goal. We designed and formally verified a security protocol that makes a Honeywords System resilient to malicious modification of the code that implements the role of the Login Server (LS) in the authentication protocol. The protocol assumes that the user has a One-Time-Password (OTP) device, however, alternatively to existing multi-factor authentication solutions, in our proposal a random factor that could serve as a second factor authentication, is also used to protect the passwords at rest. Then, even if the password is leaked, the adversary would need to guess this other factor as well in order to gain access to the system.
2. Along with the previous problem, users selecting weak passwords and reusing them on multiple sites is a well-known standard behavior that has made it relatively easy for attackers to retrieve them, and thereby, gain access to all the services for which the user is registered with those credentials. Moreover, adversaries can retrieve the passwords in transit, as they are usually sent in plaintext from the user's client to the password-based authentication server. To strengthen the protection of weak passwords during and after the authentication process, we designed an authentication protocol that works together with an OTP value. The protocol has some limitations due to the low entropy of OTP values displayed in a device, however we already envision potential directions for increasing the entropy (discussed in Chapter 8), one of which consists in using the values generated by the OTP algorithm before they are truncated (to appear in a user device).
3. Finally, we proposed a replacement for the use of out-of-band (OOB)-based entity authentication with a password-authenticated key exchange (PAKE)-based approach; this approach gives provable security guarantees to the process and provides the required cryptographic elements (a shared high-entropy key) to implement additional and more sophisticated security properties, such as perfect forward secrecy (PFS), deniability and post-quantum security, along with usability properties such as automation of future key pair authentication, immediate enrolment, alert-less key renewal and inattentive user resistance.

   The implementation of this approach represents an important step towards improving the overall security and usability of current decentralized end-to-end secure solutions, as the dependence on users for correctly carrying out authentication tasks, would be minimized. Furthermore, this result can be equally applied to email and messaging solutions as in both cases, the prevailing method of authentication in decentralized settings is based on OOB approaches.

Note that end-to-end secure email solutions could still be of help in the case of an adversary having access to a user's mailbox, as depending on the email client settings, a passphrase is usually required for decrypting emails. Nonetheless, recalling the issues that we have highlighted throughout this thesis and considering that users have difficulties already selecting adequate passwords, it is very optimistic to expect the average user to use an end-to-end secure email solution. Hence the

importance of pursuing research in both directions.

Based on the results of this thesis, we conclude that the design of p≡p's security protocols provides privacy-by-default in email communications; p≡p also achieves many of the usability features identified in the literature as relevant for secure email solutions: automatic key initialization, easy key discovery, no need for shared secrets, and partially low key maintenance, given that by design, users need to perform the authentication every time a key is updated. The set of security and usability properties could be extended by the implementation of the technique proposed in Chapter 9.

In addition, p≡p preserves the interoperability and openness features characteristic of email communication, and is a solution for decentralized settings.

We would like to recall the fact that authentication techniques are heavily dependent on users, who play a decisive role in the successful achievement of the security goals; therefore, the human component should not be neglected in the study and development of secure email solutions. At the same time, efforts are worth investing in the research of ways to shift the assignment of completing critical security tasks (such as comparing fingerprints or choosing passwords with an adequate level of security) from the user, to automated methods preferably with provable security.

Limitations

The formal verification and proofs provided in this thesis are based on an abstraction of the protocols implemented in p≡p, which like all mathematical models, are approximations to reality; we did not verify the actual implementation. While our proofs support the protocol design behind p≡p, formal verification of the deployed protocols is still missing. As discussed at the beginning of this work, formal verification of implementations is an active growing area of research, with major challenges regarding the development of a fully automated process that would allow automated formal verification, as is the case in the symbolic model.

To verify the p≡p implementation, one could follow an approach similar to that used by Bhargavan et al. [29] to verify TLS, which roughly consists in creating a reference implementation of the software to be verified, in a language that allows verification of the code, such as F*. Following this approach, however, requires dedicated human resources, who as a team have the required knowledge in formal verification, software development and in the implementation of p≡p. The problem remains open.

10.1 Open questions and further directions

Throughout this work, we have indicated several open problems in different areas of research. For instance, the lack of software solutions for reverse engineering static C code, the need for finding adequate cryptographic techniques for dealing with the strong adversarial model of code-corruption in the Honeywords system, how to design and enhance theoretical schemes for a human-inclusive analysis of security protocols, and in general, how certain cryptographic schemes can be applied to the development of a usable secure email solution. We invite the reader to review the "Concluding remarks and further directions" section in each chapter for detailed explanations of these and other open problems, related to the specific topics discussed in each chapter.

Instead, we comment on specific directions in which we have work in progress:

Secure device pairing protocols for email Secure device pairing aims at establishing a secure communication channel between two unlinked devices, usually owned by the same user, without relying on trusted infrastructure; for instance, the Simple Secure Pairing protocol for Bluetooth.

Our interest lies in secure device pairing for enabling private email communication; thus, the goal

is to establish an authenticated channel ch between (initially) two devices d_1, d_2, from which a user accesses the same mailbox. Once established, d_1 and d_2 can use ch to share secret data that would allow the user to consistently encrypt and decrypt emails in both devices. In particular, we have already started to study the Key Synchronization (KeySync) protocol [30] of p≡p, whose purpose is to carry out an authenticated device pairing, using a common-access resource (the mailbox) as a channel to synchronize the devices.

We aim at providing a formal analysis of the p≡p's KeySync protocol, specially considering the identity misbinding attack [65], which has been pointed out to affect most of the secure device pairing protocols [161]. We also intend to study the approaches that WhatsApp and Signal use to make messages accessible from the mobile device and from the web/desktop application, and compare them with KeySync. Although they work in different trust settings (centralized and decentralized), we could get insights into whether some aspects are worth being carried over from one model to another.

Implementation of PAKE-based authentication The research in Chapter 9 left many open research questions, one of which has to do with the implementation of the proposed technique, i.e., the implementation of an authentication module which asks the user to enter a password shared with the authentication peer, and uses this input to run a PAKE protocol with the corresponding peer, who is also asked to enter the shared password; the algorithm should generate a cryptographic key and then use it to determine whether the passwords are equal, in which case, the peer's authenticity is validated.

The implementation should be interoperable and portable, so that it can be integrated with secure email solutions and used regardless of the platform. For the moment we intend to develop a prototype implementation, using adequate cryptographic libraries, to assess and benchmark our hypothesis.

Verified PAKE implementation As a follow up of the previous point, once we have a prototype with a functional PAKE implementation, a direction that we (or being fair, "I") look forward to following is the development of an implementation of the selected PAKE protocol in F*, with the final purpose of generating a verified implementation. This seems indeed to be a natural direction; first, because it is consistent with what we encouraged in the thesis, i.e., that cryptographic libraries are delicate elements underlying omnipresent protocols, which call for strict proofs of their correctness and security; second, because there is already a lack of robust PAKE implementations, and in fact, to the best of our knowledge, there are no verified ones.

Another intriguing aspect that we started considering concerns the recently developed proximity tracing apps, designed during the COVID-19 health crisis for assisting in the detection of potentially infected contacts of a confirmed case. In particular, further studies on the techniques used for preserving anonymity of users, could provide new ideas for implementing such a property in secure email and messaging, which is currently achieved mostly in the transport layer, e.g., by using the Tor's anonymity network [68], or the recently proposed anonymity system Loopix [140].

To conclude, we hope to have planted in the reader's mind at least a small seed of curiosity for any of the topics addressed in this thesis, which along with the right questions, can lead one to challenging paths and unexplored domains.

BIBLIOGRAPHY

[1] M. Abadi and C. Fournet. “Mobile values, new names, and secure communication”. In: *Acm Sigplan Notices*. Vol. 36. ACM. 2001, pp. 104–115.

[2] M. Abadi and A. D. Gordon. “A calculus for cryptographic protocols: The spi calculus”. In: *Information and computation* 148.1 (1999), pp. 1–70.

[3] M. Abdalla and M. Barbosa. *Perfect Forward Security of SPAKE2*. Cryptology ePrint Archive, Report 2019/1194. https://eprint.iacr.org/2019/1194. 2019.

[4] M. Abdalla and D. Pointcheval. “Simple password-based encrypted key exchange protocols”. In: *Cryptographers’ track at the RSA conference*. Springer. 2005, pp. 191–208.

[5] R. Abu-Salma et al. “Obstacles to the adoption of secure communication tools”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 137–153.

[6] T. Acar, C. Fournet, and D. Shumow. *Cryptographically Verified Design and Implementation of a Distributed Key Manager*. Tech. rep. Microsoft Research, Apr. 2014.

[8] M. Aizatulin, A. D. Gordon, and J. Jürjens. “Extracting and Verifying Cryptographic Models from C Protocol Code by Symbolic Execution”. In: *Proceedings of the 18th ACM Conf. on Computer and Communications Security*. CCS ’11. Chicago, Illinois, USA: ACM, 2011, pp. 331–340.

[9] C. Alexander and I. Goldberg. “Improved user authentication in off-the-record messaging”. In: *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM. 2007, pp. 41–47.

[10] E. Atwater, C. Bocovich, U. Hengartner, E. Lank, and I. Goldberg. “Leading Johnny to water: Designing for usability and trust”. In: *Eleventh Symposium On Usable Privacy and Security ({SOUPS} 2015)*. 2015, pp. 69–88.

[11] M. Avalle, A. Pironti, and R. Sisto. “Formal verification of security protocol implementations: a survey”. In: *Formal Aspects of Computing* (2012), pp. 1–25.

[12] G. Avoine, S. Canard, and L. Ferreira. “Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2020, pp. 199–224.

[13] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. “Password-protected secret sharing”. In: *Proceedings of the 18th ACM conference on Computer and Communications Security*. 2011, pp. 433–444.

[14] W. Bai et al. “Balancing Security and Usability in Encrypted Email”. In: *IEEE Internet Computing* 21.3 (2017), pp. 30–38.

[15] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.

[16] D. Basin, S. Radomirovic, and L. Schmid. “Modeling human errors in security protocols”. In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE. 2016, pp. 325–340.

[17] D. Basin, S. Radomirovic, and L. Schmid. “Modeling human errors in security protocols”. In: *Proceedings - IEEE Computer Security Foundations Symposium* 2016-August (2016).

[18] D. Basin et al. “A formal analysis of 5G authentication”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 1383–1396.

[19] D. Basin et al. “ARPKI: Attack resilient public-key infrastructure”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014, pp. 382–393.

[21] J. Becerra, D. Ostrev, and M. Škrobot. “Forward secrecy of SPAKE2”. In: *International Conference on Provable Security*. Springer. 2018, pp. 366–384.

[22] J. Becerra, P. B. Rønne, P. Y. Ryan, and P. Sala. “HoneyPAKEs”. In: *Cambridge International Workshop on Security Protocols*. Springer. 2018, pp. 63–77.

[24] B. Beckert and G. Beuster. “A method for formalizing, analyzing, and verifying secure user interfaces”. In: *International Conference on Formal Engineering Methods*. Springer. 2006, pp. 55–73.

[25] G. Bella and L. Coles-Kemp. “Layered analysis of security ceremonies”. In: *IFIP International Information Security Conference*. Springer. 2012, pp. 273–286.

[26] M. Bellare, D. Pointcheval, and P. Rogaway. “Authenticated key exchange secure against dictionary attacks”. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 2000, pp. 139–155.

[27] S. M. Bellovin and M. Merritt. “Encrypted key exchange: password-based protocols secure against dictionary attacks”. In: *1992 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society, 1992, pp. 72–84.

[28] D. J. Bernstein, T. Lange, and P. Schwabe. “The security impact of a new cryptographic library.” In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 646.

[29] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. “Implementing TLS with verified cryptographic security”. In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 445–459.

[33] A. Biryukov, D. Dinu, and D. Khovratovich. “Argon2: new generation of memory-hard functions for password hashing and other applications”. In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 292–302.

[34] E. Blanchard, X. Coquand, and T. Selker. “Moving to client-side hashing for online authentication”. In: *9th International Workshop on Socio-Technical Aspects in SecuriTy*. 2019.

[35] B. Blanchet. “A Computationally Sound Automatic Prover for Cryptographic Protocols”. In: *Workshop on the link between formal and computational models*. June 2005.

[36] B. Blanchet. “An efficient cryptographic protocol verifier based on Prolog rules”. In: *14th IEEE Computer Security Foundations Workshop*. IEEE. 2001, pp. 82–96.

[37] B. Blanchet. “Automatic verification of correspondences for security protocols”. In: *Journal of Computer Security* 17.4 (2009), pp. 363–434.

[38] B. Blanchet. “Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif”. In: *Found. Trends Priv. Secur.* 1.1-2 (Oct. 2016), pp. 1–135.

[39] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu. “Using Formal Verification to Evaluate Human-Automation Interaction: A Review”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.3 (May 2013), pp. 488–503.

[40] J. Bonneau. “The science of guessing: analyzing an anonymized corpus of 70 million passwords”. In: *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE. 2012, pp. 538–552.

[41] J. Bonneau, C. Herley, P. C. Van 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*. IEEE. 2012, pp. 553–567.

[42] N. Borisov, I. Goldberg, and E. Brewer. “Off-the-record communication, or, why not to use PGP”. In: *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. 2004, pp. 77–84.

[43] J. W. Bos, C. Costello, P. Longa, and M. Naehrig. “Selecting elliptic curves for cryptography: An efficiency and security analysis”. In: *Journal of Cryptographic Engineering* 6.4 (2016), pp. 259–286.

[44] F. Boudot, B. Schoenmakers, and J. Traore. “A fair and efficient solution to the socialist millionaires’ problem”. In: *Discrete Applied Mathematics* 111 (2001), pp. 23–36.

[46] M. Burrows, M. Abadi, and R. M. Needham. “A logic of authentication”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 426.1871 (1989), pp. 233–271.

[47] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. “Automated verification of equivalence properties of cryptographic protocols”. In: *ACM Transactions on Computational Logic (TOCL)* 17.4 (2016), pp. 1–32.

[48] D. L. Chaum. “Untraceable electronic mail, return addresses, and digital pseudonyms”. In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[49] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. “NuSMV: a new Symbolic Model Verifier”. In: *Proceedings Eleventh Conference on Computer-Aided Verification (CAV’99)*. Ed. by N. Halbwachs and D. Peled. Lecture Notes in Computer Science 1633. Springer, July 1999, pp. 495–499.

[50] J. Clark, P. C. van Oorschot, S. Ruoti, K. Seamons, and D. Zappala. “Securing Email”. In: *arXiv preprint arXiv:1804.07706* (2018).

[51] E. M. Clarke and E. A. Emerson. “Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic”. In: *Logic of Programs, Workshop*. Springer-Verlag, 1982, pp. 52–71.

[52] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. “A formal security analysis of the Signal messaging protocol”. In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2017, pp. 451–466.

[53] V. Cortier and S. Kremer. “Formal models and techniques for analyzing security protocols: A tutorial”. In: *Foundations and Trends in Programming Languages* 1.3 (2014).

[54] V. Cortier, S. Kremer, and B. Warinschi. “A survey of symbolic methods in computational analysis of cryptographic systems”. In: *Journal of Automated Reasoning* 46.3-4 (2011), pp. 225–259.

[55] G. Couteau, A. W. Roscoe, and P. Y. A. Ryan. *Partially-Fair Computation from Timed-Release Encryption and Oblivious Transfer*. Cryptology ePrint Archive, Report 2019/1281. `https://eprint.iacr.org/2019/1281`. 2019.

[56] L. F. Cranor. “A framework for reasoning about the human in the loop”. In: *Proceedings of the 1st Conference on Usability, Psychology, and Security (UPSEC’08)* (2008), pp. 1–15.

[57] C. Cremers. “Key Exchange in IPsec Revisited: Formal Analysis of IKEv1 and IKEv2”. In: *Computer Security – ESORICS 2011*. Ed. by V. Atluri and C. Diaz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 315–334.

[59] P. Curzon, R. Rukšėnas, and A. Blandford. “An approach to formal verification of human–computer interaction”. In: *Formal Aspects of Computing* 19.4 (Nov. 2007), pp. 513–550.

[60] S. Dechand et al. “An empirical study of textual key-fingerprint representations”. In: *25th {USENIX} Security Symposium*. 2016, pp. 193–208.

[61] A. Degani and M. Heymann. “Formal Verification of Human-Automation Interaction”. In: *Human Factors* 44.1 (2002), pp. 28–43.

[62] S. Delaune, S. Kremer, and L. Robin. “Formal verification of protocols based on short authenticated strings”. In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE. 2017, pp. 130–143.

[63] M. Di Raimondo, R. Gennaro, and H. Krawczyk. “Deniable authentication and key exchange”. In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 400–409.

[64] W. Diffie and M. Hellman. “New directions in cryptography”. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[65] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. “Authentication and authenticated key exchanges”. In: *Designs, Codes and cryptography* 2.2 (1992), pp. 107–125.

[67] J. Ding, S. Alsayigh, J. Lancrenon, R. Saraswathy, and M. Snook. “Provably secure password authenticated key exchange based on RLWE for the post-quantum world”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2017, pp. 183–204.

[68] R. Dingledine, N. Mathewson, and P. Syverson. “Tor: The Second-generation Onion Router”. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM’04. San Diego, CA: USENIX Association, 2004, pp. 21–21.

[69] D. Dolev and A. C. Yao. “On the Security of Public Key Protocols”. In: *Proceedings of the 22Nd Annual Symposium on Foundations of Computer Science*. SFCS ’81. Washington, DC, USA: IEEE Computer Society, 1981, pp. 350–357.

[70] B. P. Douglass. *UML for the C programming language*. Tech. rep. IBM, June 2009.

[71] E. Duncan, H. Fanderl, T. Freundt, N. Maechler, and N. K. *Customer Experience: new capabilities, new audiences, new opportunities*. 2017.

[72] E. Eilam. *Reversing: Secrets of Reverse Engineering*. New York, NY, USA: John Wiley & Sons, Inc., 2005.

[73] C. Ellison. “Ceremony Design and Analysis”. In: *IACR Cryptology ePrint Archive* 399 (2007), pp. 1–17. URL: http://eprint.iacr.org/2007/399.

[74] I. Erguler. “Achieving flatness: Selecting the honeywords from existing user passwords”. In: *IEEE Transact. on Dependable and Secure Computing* 13.2 (2016), pp. 284–295.

[75] EU. “Regulation (EU) 2016/679 of the European Parliament and of the Council. On the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation”. In: *Official Journal of the European Union* L119 (May 2016), pp. 1–88. URL: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC.

[78] M. Fischlin, F. Günther, B. Schmidt, and B. Warinschi. “Key confirmation in key exchange: A formal treatment and implications for TLS 1.3”. In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 452–469.

[80] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999.

[81] H. Garavel and S. Graf. *Formal Methods for Safe and Secure Computers Systems*. Tech. rep. Federal Office for Inf. Security, 2013.

[82] J. A. Garay, P. D. MacKenzie, and K. Yang. “Efficient and Secure Multi-Party Computation with Faulty Majority and Complete Fairness.” In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 9.

[86] C. Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 169–178.

[91] F. Günther. “Modeling Advanced Security Aspects of Key Exchange and Secure Channel Protocols”. PhD thesis. Darmstadt University of Technology, Germany, 2018.

[92] R. Haenni and O. Spycher. “Secure Internet Voting on Limited Devices with Anonymized DSA Public Keys”. In: *Proc. of the 2011 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE’11), San Francisco, CA*. USENIX Association, 2011, pp. 8–8.

[93] F. Hao and P. Y. A. Ryan. “J-PAKE: authenticated key exchange without PKI”. In: *Transactions on computational science XI*. Springer, 2010, pp. 192–206.

[94] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon. “Verification of User Interface Software: The Example of Use-Related Safety Requirements and Programmable Medical Devices”. In: *IEEE Trans. Human-Machine Systems* 47.6 (2017), pp. 834–846.

[95] G. Hatzivasilis, I. Papaefstathiou, and C. Manifavas. “Password Hashing Competition-Survey and Benchmark.” In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 265.

[97] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[99] S. Jarecki, H. Krawczyk, and J. Xu. “OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 456–486.

[100] C. Johansen and A. Jøsang. “Probabilistic modelling of humans in security ceremonies”. In: *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*. Springer, 2014, pp. 277–292.

[101] A. Juels and R. L. Rivest. “Honeywords: Making password-cracking detectable”. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, pp. 145–160.

[103] R. Kainda, I. Flechais, and A. Roscoe. “Secure mobile ad-hoc interactions: reasoning about out-of-band (oob) channels”. In: *IWSSI/SPMU* 2010 (2010), pp. 10–15.

[104] R. Kainda, I. Flechais, and A. Roscoe. “Security and usability: Analysis and evaluation”. In: *2010 International Conference on Availability, Reliability and Security*. IEEE. 2010, pp. 275–282.

[105] R. Kainda, I. Flechais, and A. Roscoe. “Usability and security of out-of-band channels in secure device pairing protocols”. In: *Proceedings of the 5th Symposium on Usable Privacy and Security*. ACM. 2009, p. 11.

[106] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014.

[107] J. Katz and V. Vaikuntanathan. “Round-optimal password-based authenticated key exchange”. In: *Theory of Cryptography Conference*. Springer. 2011, pp. 293–310.

[109] H. Krawczyk. “Cryptographic extraction and key derivation: The HKDF scheme”. In: *Annual Cryptology Conference*. Springer. 2010, pp. 631–648.

[111] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun. “A comparative study of secure device pairing methods”. In: *Pervasive and Mobile Computing* 5.6 (2009), pp. 734–749.

[112] L. Lamport. “Password Authentication with Insecure Communication”. In: *Communication of the ACM* 24.11 (1981), pp. 770–772.

[113] B. Laurie. “Certificate Transparency”. In: *Commun. ACM* 57.10 (Sept. 2014), pp. 40–46. URL: https://doi.org/10.1145/2659897.

[114] A. Lerner, E. Zeng, and F. Roesner. “Confidante: Usable Encrypted Email: A Case Study with Lawyers and Journalists”. In: *2017 IEEE European Symposium on Security and Privacy*. 2017, pp. 385–400.

[116] Z. Liu, H. Seo, J. Großschädl, and H. Kim. “Efficient implementation of NIST-compliant elliptic curve cryptography for 8-bit AVR-based sensor nodes”. In: *IEEE Transactions on Information Forensics and Security* 11.7 (2015), pp. 1385–1397.

[117] G. Lowe. “A hierarchy of authentication specifications”. In: *Proceedings 10th Computer Security Foundations Workshop*. IEEE. 1997, pp. 31–43.

[118] G. Lowe. “Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by T. Margaria and B. Steffen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 147–166.

[119] V. Lyubashevsky, C. Peikert, and O. Regev. *On Ideal Lattices and Learning with Errors Over Rings*. Cryptology ePrint Archive, Report 2012/230. https://eprint.iacr.org/2012/230. 2012.

[120] A. Malatras, I. Coisel, and I. Sanchez. *A security analysis of email communications*. Tech. rep. Publications Office of the European Union, 2015. URL: http://publications.jrc.ec.europa.eu/repository/handle/JRC99372.

[121] D. Malone and K. Maher. "Investigating the Distribution of Password Choices". In: *Proc, of the 21st Int. Conf. on World Wide Web*. WWW '12. Lyon, France: ACM, 2012, pp. 301–310.

[125] S. Mauw and C. Cremers. *Operational Semantics and Verification of Security Protocols*. Springer Science & Business Media, 2012.

[126] C. Meadows. "Language generation and verification in the NRL protocol analyzer". In: *Proceedings 9th IEEE Computer Security Foundations Workshop*. IEEE. 1996, pp. 48–61.

[127] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. 1st. USA: CRC Press, Inc., 1996.

[131] M. Naor, L. Rotem, and G. Segev. "The security of lazy users in out-of-band authentication". In: *Theory of Cryptography Conference*. Springer. 2018, pp. 575–599.

[132] J. Narayan, S. Shukla, and T. Charles Clancy. "A Survey of Automatic Protocol Reverse Engineering Tools". In: *ACM Computing Surveys* 48 (Dec. 2015), pp. 1–26.

[134] L. H. Nguyen and A. W. Roscoe. "Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey". In: *Journal of Computer Security* 19.1 (2011), pp. 139–201.

[137] L. C. Paulson. "The inductive approach to verifying cryptographic protocols". In: *Journal of computer security* 6.1-2 (1998), pp. 85–128.

[139] C. Percival. "Stronger key derivation via sequential memory-hard functions". In: (2009).

[140] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. "The loopix anonymity system". In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, pp. 1199–1216.

[149] N. Provos and D. Mazieres. "A Future-Adaptable Password Scheme." In: *USENIX Annual Technical Conference, FREENIX Track*. 1999, pp. 81–91.

[150] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *Journal of the ACM (JACM)* 56.6 (2009), pp. 1–40.

[153] R. L. Rivest and A. Shamir. "How to expose an eavesdropper". In: *Communications of the ACM* 27.4 (1984), pp. 393–394.

[154] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[155] A. W. Roscoe. "Detecting failed attacks on human-interactive security protocols". In: *Cambridge International Workshop on Security Protocols*. Springer. 2016, pp. 181–197.

[156] A. W. Roscoe and P. Y. A. Ryan. "Auditable PAKEs: approaching fair exchange without a TTP". In: *Cambridge International Workshop on Security Protocols*. Springer. 2017.

[157] S. Ruoti, J. Andersen, T. Monson, D. Zappala, and K. Seamons. "A comparative usability study of key management in secure email". In: *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)*. 2018, pp. 375–394.

[158] S. Ruoti et al. "" We're on the Same Page" A Usability Study of Secure Email Using Pairs of Novice Users". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2016, pp. 4298–4308.

[159] P. Ryan and S. Schneider. *The Modelling and Analysis of Security Protocols: The Csp Approach*. First. Addison-Wesley Professional, 2000.

[160] E. B. Sasson et al. “Zerocash: Decentralized anonymous payments from bitcoin”. In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 459–474.

[161] M. Sethi, A. Peltonen, and T. Aura. “Misbinding attacks on secure device pairing and bootstrapping”. In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. ACM. 2019, pp. 453–464. URL: https://arxiv.org/pdf/1902.07550.pdf.

[162] M. Shirvanian and N. Saxena. “Wiretapping via Mimicry: Short Voice Imitation Man-in-the-Middle Attacks on Crypto Phones”. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 868–879.

[166] N.-F. Standard. “Announcing the advanced encryption standard (aes)”. In: *Federal Information Processing Standards Publication* 197.1-51 (2001), pp. 3–3.

[167] R. Stedman, K. Yoshida, and I. Goldberg. “A user study of off-the-record messaging”. In: *Proceedings of the 4th symposium on Usable privacy and security*. 2008, pp. 95–104.

[170] J. Tan et al. “Can unicorns help users compare crypto key fingerprints?” In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM. 2017, pp. 3787–3798.

[173] The Radicati Group. *Email Statistics Report, 2020-2024*. Tech. rep. https://www.radicati.com/wp/wp-content/uploads/2020/01/Email_Statistics_Report,_2020-2024_Executive_Summary.pdf. The Radicati Group, Inc., 2020.

[174] N. Tiwari and L. Prasad. “Reverse Engineering Tools for Simplifying Programming Environment through Flowcharting”. In: *Int. Journal of Eng. Trends and Technology (IJETT)* 26 (Aug. 2015), pp. 65–71.

[175] P. Tonella and R. Potrich. “Reverse engineering of the interaction diagrams from C++ code”. In: *In International Conference on Software Maintenance*. 2003, pp. 159–168.

[176] M. S. Turan, E. Barker, W. Burr, and L. Chen. *Recommendation for Password-Based Key Derivation. Part 1: Storage Applications*. Dec. 2010. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf.

[177] S. Turner. “Secure/multipurpose internet mail extensions”. In: *IEEE Internet Computing* 14.5 (2010), pp. 82–86.

[179] N. Unger et al. “SoK: secure messaging”. In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 232–249.

[180] B. Ur et al. “Do Users’ Perceptions of Password Security Match Reality?” In: *Proc. of the 2016 CHI Conf. on Human Factors in Computing Systems (CHI’16)* (2016), pp. 3748–3760.

[181] S. Vaudenay. “Secure communications over insecure channels based on short authenticated strings”. In: *Annual International Cryptology Conference*. Springer. 2005, pp. 309–326.

[182] E. Vaziripour et al. “Is that you, Alice? a usability study of the authentication ceremony of secure messaging applications”. In: *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*. 2017, pp. 29–47.

[187] L. Viganò. “Towards the Secure Provision and Consumption in the Internet of Services”. In: *Trust, Privacy and Security in Digital Business: 9th International Conference, TrustBus 2012, Vienna, Austria, September 3-7, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 214–215.

[188] E. Von Zezschwitz, A. De Luca, and H. Hussmann. “Survival of the Shortest: A Retrospective Analysis of Influencing Factors on Password Composition”. In: *Human-Computer Interaction (INTERACT 2013)*. Springer-Verlag, 2013, pp. 460–467.

[190] R. Wash, E. Rader, R. Berman, and Z. Wellmer. “Understanding Password Choices: How Frequently Entered Passwords Are Re-used across Websites”. In: *Proc, of 12th Symposium on Usable Privacy and Security (SOUPS 2016)*. Denver, CO: USENIX Association, 2016, pp. 175–188.

[191] A. Whitten and J. D. Tygar. “Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0.” In: *USENIX Security Symposium*. Vol. 348. 1999.

[194] T. Y. Woo and S. S. Lam. “A semantic model for authentication protocols”. In: *Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE. 1993, pp. 178–194.

[195] A. C. Yao. “Protocols for secure computations”. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164.

[196] P. R. Zimmermann. *The official PGP user’s guide*. MIT press, 1995.

[197] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche. “HACL * : A Verified Modern Cryptographic Library”. In: *ACM Conference on Computer and Communications Security (CCS)*. Oct. 2017.

Online References

[7] P. Agrawal. *Keeping your account secure*. May 2018. URL: https://blog.twitter.com/official/en_us/topics/company/2018/keeping-your-account-secure.html (visited on 07/06/2020).

[20] D. Basin et al. *Tamarin Prover*. URL: https://tamarin-prover.github.io/.

[23] K. Beck et al. *Manifesto for Agile Software Development*. Ed. by A. Alliance. 2001. URL: https://www.agilealliance.org/agile101/the-agile-manifesto (visited on 05/30/2020).

[30] V. Birk, B. Hoeneisen, and K. Bristol. *pretty Easy privacy (pEp): Key Synchronization Protocol (KeySync) draft-pep-keysync-00*. Nov. 2019. URL: https://tools.ietf.org/html/draft-pep-keysync-00.

[31] V. Birk, H. Marques, and B. Hoeneisen. *IANA Registration of Trustword Lists*. 2019. URL: https://tools.ietf.org/html/draft-birk-pep-trustwords-03 (visited on 02/22/2020).

[32] V. Birk, H. Marques, and B. Hoeneisen. *pretty Easy privacy (pEp): Privacy by Default*. URL: https://tools.ietf.org/id/draft-birk-pep-05.html (visited on 07/20/2020).

[45] V. C. Bruno Blanchet Ben Smyth and M. Sylvestre. *ProVerif 2.02: Automatic Cryptographic Protocol Verifier, user manual and tutorial*. URL: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf (visited on 08/07/2020).

[58] M. Crispin. *Internet Message Access Protocol - Version 4rev1*. Mar. 2003. URL: https://tools.ietf.org/html/rfc3501.

[66] *DigiNotar*. URL: https://en.wikipedia.org/wiki/DigiNotar (visited on 09/30/2020).

[79] pEp Foundation Council. *Privacy by Default. White paper*. URL: https://pep.foundation/docs/pEp-whitepaper.pdf (visited on 09/30/2020).

[83] R. Gellens and J. Klensin. *Message submission for mail*. Nov. 2011. URL: https://tools.ietf.org/html/rfc6409.

[87] M. Green. *Let’s talk about PAKE*. Oct. 2018. URL: https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/ (visited on 08/20/2020).

[88] A. Greenberg. *Hackers are passing around a megaleak of 2.2 billion records*. Jan. 2019. URL: https://www.wired.com/story/collection-leak-usernames-passwords-billions/ (visited on 09/10/2020).

[89] O. W. Group. *OpenPGP*. URL: https://www.openpgp.org/ (visited on 05/02/2019).

[90] T. Guardian. *NSA Prism program taps in to user data of Apple, Google and others*. URL: https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data (visited on 09/01/2020).

[102] P. Juola and P. Zimmermann. *PGP Word List*. URL: https://en.wikipedia.org/wiki/PGP_word_list (visited on 06/29/2019).

[108] J. Klensin. *Simple Mail Transfer Protocol*. Oct. 2008. URL: https://tools.ietf.org/html/rfc5321.

[110] B. Krebs. *Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years*. Mar. 2019. URL: https://web.archive.org/web/20190322091235/https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/ (visited on 09/20/2020).

[115] X. Leroy. *The compcert C compiler*. URL: http://compcert.inria.fr/publi.html.

[122] M. Marlinspike and T. P. (editor). *Signal Technical Specifications*. URL: https://signal.org/docs/ (visited on 06/30/2020).

[123] H. Marques and B. Hoeneisen. *pretty Easy privacy (pEp): Contact and Channel Authentication through Handshake*. URL: https://tools.ietf.org/html/draft-marques-pep-handshake-03 (visited on 07/20/2020).

[124] H. Marques and B. Hoeneisen. *pretty Easy privacy (pEp): Mapping of Privacy Rating*. Mar. 2019. URL: https://tools.ietf.org/id/draft-marques-pep-rating-01.html.

[129] M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. *HOTP: An HMAC-Based One-Time Password Algorithm*. Dec. 2005. URL: https://tools.ietf.org/html/rfc4226.

[130] J. Myers and M. Rose. *Post Office Protocol - Version 3*. May 1996. URL: https://tools.ietf.org/html/rfc1939.

[133] L. H. Newman. *Yahoo's 2013 email hack actually compromised three billion accounts*. Oct. 2017. URL: https://www.wired.com/story/yahoo-breach-three-billion-accounts/ (visited on 09/10/2020).

[136] OTRv4-development. *Specification of OTR version 4*. Oct. 2019. URL: https://github.com/otrv4/otrv4/blob/master/otrv4.md.

[142] M. Pound. *How to Choose a Password*. July 2016. URL: https://www.youtube.com/watch?v=3NjQ9b3pgIg (visited on 08/30/2020).

[143] M. Pound. *Password cracking*. July 2016. URL: https://www.youtube.com/watch?v=7U-RbOKanYs (visited on 08/30/2020).

[144] P. E. Privacy. *pEp Source Code*. URL: https://pep.foundation/pep-software/index.html (visited on 05/21/2019).

[145] P. E. Privacy. *pEp Source Code. Version studied for the Trustwords Algorithm*. URL: https://pep.foundation/dev/repos/pEpEngine/file/f83c6a876204/src/message_api.c (visited on 07/24/2020).

[146] P. E. Privacy. *pEp User Documentation*. URL: https://www.pep.security/docs/index.html (visited on 02/20/2020).

[147] Project Everest. *EverCrypt*. URL: https://github.com/project-everest/hacl-star/tree/master/providers/evercrypt (visited on 08/21/2020).

[148] Proton Technologies AG. *ProtonMail*. URL: https://protonmail.com/ (visited on 09/20/2020).

[151] M. Research and Inria. *F**. URL: https://fstar-lang.org/ (visited on 07/06/2020).

[152] M. Research, C. M. University, INRIA, and MSR-INRIA. *Project Everest*. URL: https://project-everest.github.io/ (visited on 09/20/2020).

[163] Signal. *Improving Registration Lock with Secure Value Recovery*. URL: https://signal.org/blog/improving-registration-lock (visited on 02/20/2020).

[164] Signal. *Technology Preview for secure value recovery*. URL: https://signal.org/blog/secure-value-recovery (visited on 07/02/2020).

[171] T. O. D. Team. *Off-the-Record Messaging*. URL: https://otr.cypherpunks.ca/ (visited on 09/30/2020).

[172] The GnuPG Project. *The GNU Privacy Guard*. URL: https://gnupg.org/ (visited on 08/28/2020).

[178] Tutanota. *Tutanota*. URL: https://tutanota.com/ (visited on 09/20/2020).

[189] N. H. Walfield, J. Winter, and K. Michaelis. *Sequoia-PGP*. URL: https://sequoia-pgp.org/.

[192] Wikipedia. *Comparison of cryptography libraries*. URL: https://en.wikipedia.org/wiki/Comparison_of_cryptography_libraries (visited on 07/10/2020).

[193] Wikipedia. *List of Unified Modeling Language tools*. URL: https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools (visited on 07/10/2020).

Reverse Engineering Tools

[76] EventHelix. URL: https://www.eventhelix.com/VisualEther/ (visited on 05/01/2020).

[77] FateSoft. URL: http://fatesoft.com/s2f/ (visited on 05/01/2020).

[96] D. van Heesch. URL: http://www.stack.nl/~dimitri/doxygen (visited on 05/01/2020).

[98] IBM. URL: https://www.ibm.com/developerworks/downloads/r/architect/index.html (visited on 04/01/2018).

[128] Microsoft. URL: https://www.visualstudio.com/es/vs/ (visited on 05/01/2020).

[135] Oovaide. URL: http://oovaide.sourceforge.net/ (visited on 05/01/2020).

[138] Z. Peng. URL: https://www.zenuml.com/ (visited on 12/15/2017).

[141] PlantUML. URL: http://plantuml.com/ (visited on 04/03/2017).

[165] N. Sofer. *DLL Export Viewer*. URL: http://www.nirsoft.net/utils/dll_export_viewer.html (visited on 05/01/2020).

[169] S. Systems. *Enterprise Architect*. URL: https://sparxsystems.com/ (visited on 05/01/2020).

List of Publications

[84] Z. A. Genç, G. Lenzini, P. Ryan, and I. Vazquez Sandoval. “A Security Analysis, and a Fix, of a Code-Corrupted Honeywords System”. In: *Proc. of the 4th Int. Conf. on Information Systems Security and Privacy (ICISSP 2018)*. 2018, pp. 83–95.

[85] Z. Genç, G. Lenzini, P. Ryan, and I. Vazquez Sandoval. “A Critical Security Analysis of the Password-Based Authentication Honeywords System Under Code-Corruption Attack”. In: *International Conference on Information Systems Security and Privacy*. Springer. 2018, pp. 125–151.

[168] B. Stojkovski, I. Vazquez Sandoval, and G. Lenzini. “Detecting misalignments between system security and user perceptions: a preliminary socio-technical analysis of an e2e email encryption system”. In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2019, pp. 172–181.

[183] I. Vazquez Sandoval, A. Atashpendar, and G. Lenzini. “Authentication and Key Management Automation in Decentralized Secure Email and Messaging via Low-Entropy Secrets”. In: *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications*. 2020.

[184] I. Vazquez Sandoval and G. Lenzini. “A Formal Security Analysis of the pEp Authentication Protocol for Decentralized Key Distribution and End-to-End Encrypted Email”. In: *Emerging Technologies for Authorization and Authentication* (2019).

[185] I. Vazquez Sandoval and G. Lenzini. “Experience Report: How to Extract Security Protocols’ Specifications from C Libraries”. In: *IEEE 42nd Annual COMPSAC 2018, Tokyo, Japan, Volume 2*. 2018, pp. 719–724.

[186] I. Vazquez Sandoval, B. Stojkovski, and G. Lenzini. “A Protocol to Strengthen Password-Based Authentication”. In: *International Workshop on Emerging Technologies for Authorization and Authentication*. Springer. 2018, pp. 38–46.

Formal Models for Verification Tools

A

A.1 Protocol resilient to code-corruption of the Login Server in the Honeywords System

This code is also available at `https://github.com/codeCorruption/HoneywordsM`.

```
type blinded.
type index.
type counter.
type rowHw.

(* channels all accesible to the LS *)
free ulc:channel.
free luc:channel.
free lhc:channel.
free hlc:channel.

(* The file containing the user and the row of sweetwords *)
table file(bitstring,rowHw).

(* The index of  a sweetword (blinded) in a row of sweetwords (rowHw) *)
fun indexOfHw(blinded,rowHw): index[private].
(* The blinding of the word in the 1st arg, using the OTP given as 2nd arg *)
fun blindWord(bitstring,bitstring): blinded[private].
(*  An OTP generated with a given seed (counter) *)
fun getOTP(counter): bitstring[private].
(*  Increases the seed of the OTP *)
fun next(counter): counter[private].
(* A rowHw shuffled and blinded for the user in 1st arg, using the seed in 2nd arg *)
fun shuffleNblind(bitstring,counter):rowHw[private].
(*  Comparison of indexes in the honeychecker *)
fun checkEqual(index, index): bool
reduc
   forall u:bitstring, w:bitstring, n:counter;
      checkEqual( indexOfHw(blindWord(w,getOTP(n)),shuffleNblind(u,n)),
                  indexOfHw(blindWord(w,getOTP(n)),shuffleNblind(u,n)) ) = true
   otherwise forall x:index, y:index; checkEqual(x, y) = false[private].

(* Reshuffles and blinds the row corresponding to a given index
 * and returns the corresponding new index in the updated row *)
fun reshuffledNblinded(index):index
reduc
   forall u:bitstring, w:bitstring, n:counter;
      reshuffledNblinded( indexOfHw(blindWord(w,getOTP(n)),shuffleNblind(u,n)) )
               = indexOfHw(blindWord(w,getOTP(next(n))),shuffleNblind(u,next(n)))[private].
```

```
(* Obtains the user out of an index term *)
reduc
   forall un:bitstring, pwd:blinded, x:counter;
      getUser(indexOfHw(pwd,shuffleNblind(un,x))) = un[private].
(*  Obtains the seed out of an index term *)
reduc
   forall z:rowHw, y:bitstring, n:counter;
      getCounter(indexOfHw(blindWord(y,getOTP(n)),z)) = n[private].


(* The given index matches with the one in the HC db for the corresponding user *)
event correctIndex(index).
(* 'blinded' has index 'index' in 'rowHw' *)
event indexFound(index,blinded,rowHw).
(* The user in the 1st arg logs in using the word in the 2nd arg *)
event usrLogged(bitstring,bitstring).
(*  Marks a point not expected to be reached in the analysis *)
event unreachable.


(*--- Properties ---*)
query u:bitstring, j:index, p:bitstring, x:bitstring, y:counter;
event(correctIndex(j)) ==>  inj-event(usrLogged(u,p)) &&
                            inj-event(indexFound(j, blindWord(p,x), shuffleNblind(u,y))).

query event(unreachable).


(*--- Processes ---*)
let user(u:bitstring, p:bitstring, n:counter) =
   let hshp = blindWord(p,getOTP(n)) in
      event usrLogged(u,p);
      out (ulc, (u, hshp));
      in(luc,b:bool).

let loginService =
   in(ulc, (u:bitstring, w:blinded));
   get file(=u,rowUser) in
   let j = indexOfHw(w,rowUser) in
      event indexFound(j,w,rowUser);
      out(lhc, (j,rowUser));
      in(hlc, x:bool);
      out(luc, x).

(* The honeychecker keeps running with the updated index. A 'true'answer
   from the checkEqual function is not expected.*)
let honeychecker1(c:index) =
   in(lhc, y:index);
   let z = checkEqual(y,c) in
      out(hlc, z);
      if z then
         event unreachable.

let honeychecker(c:index) =
```

```
   in(lhc, (y:index));
   let z = checkEqual(y,c) in
      out(hlc, z);
      if z then
         insert file(getUser(y),shuffleNblind(getUser(y),next(getCounter(c))));
         (* this models that the LS receives the reshuffled row *)
         out(hlc,(getUser(y),shuffleNblind(getUser(y),next(getCounter(c)))));
         event correctIndex(y);
         honeychecker1(reshuffledNblinded(c)).


process
   ( !(new usrname: bitstring;
      new pwd: bitstring;
      new i:counter;
      insert file(usrname,shuffleNblind(usrname,i));
      user(usrname,pwd,i)|
      honeychecker(indexOfHw(blindWord(pwd,getOTP(i)),shuffleNblind(usrname,i)))
   ))  |  (!loginService)
```

A.2 Key distribution and authentication protocol of p≡p

This model addresses the key distribution and the posterior trustwords comparison processes, along with the consequent association of a privacy level to a peer.

Assumptions: (i) the attacker has no access to the user's device; (ii) the second channel for the handshake is inaccessible to the attacker; and (iii) the trustwords function is correct.

```
type userId.
type skey.
type pkey.
type pRating.


(*--- Var, const, events ---*)
channel outlook, internetAB, internetBA.
free phoneA, phoneB: channel[private].
free mssg: bitstring[private].

(* Key exchange *)
event aAddsBsec(userId,userId,pkey).
event bAddsAsec(userId,userId,pkey).
event responseReceivedOk.
event notEncrypted.
event notReplied.
event signVerifFailed.
event notSigned.
(* Handshake *)
event senderMistrustsR(userId,userId).
event senderTrustsR(userId,userId).
event receiverMistrustsS(userId,userId).
event receiverTrustsS(userId,userId).
event endHandshakeOk(userId,userId,pkey,pkey).
event endHandshakeUnsucc(userId,userId,pkey,pkey).
```

```
event startHndshkS(userId,userId).
event startHndshkR(userId,userId).
event receiverDetectsMistrusted(userId,userId).
event receivedFromNotTrusted(userId,userId).
event signVerifFails(userId,userId,bitstring).
event decryptionFails(userId,userId,bitstring).
event unknownPeer.
event end.
event sendGreen(userId,userId,bitstring).
event receiveGreen(userId,userId,bitstring).
event senderKey(userId,pkey).
event receiverKey(userId,pkey).
event senderPkey(userId,skey).


(*--- Functions ---*)
fun pubKey(skey): pkey.
fun id(skey): userId.
reduc forall s:skey; pkUser(id(s)) = pubKey(s).

fun aenc(bitstring, pkey): bitstring.
reduc forall m:bitstring, sk:skey; adec(aenc(m,pubKey(sk)),sk) = m.

fun sign(bitstring, skey): bitstring.
reduc forall m:bitstring, sk:skey; getMssg(sign(m,sk)) = m.
reduc forall m:bitstring, sk:skey; verifSign(sign(m,sk),pubKey(sk)) = m.

(* Trustwords string generated for the 2 given pEp identities *)
fun trustwords(pkey,pkey): bitstring.

(* p11=p21 & p12=p22 | p11=p22 & p12=p21 *)
fun trustwordsMatch(bitstring, bitstring): bool
reduc
forall pk1:pkey, pk2:pkey;
      trustwordsMatch(trustwords(pk1,pk2), trustwords(pk1,pk2)) = true
otherwise forall pk1:pkey, pk2:pkey;
      trustwordsMatch(trustwords(pk1,pk2), trustwords(pk2,pk1)) = true
otherwise forall w1:bitstring, w2:bitstring;
      trustwordsMatch(w1,w2) = false.


(*--- Properties ---*)

(* Full agreement *)
query a:userId, b:userId, ka:pkey, kb:pkey;
event(endHandshakeOk(a,b,ka,kb)) ==> inj-event(startHndshkS(a,b)) && inj-event(startHndshkR(b,a))
                                 && inj-event(senderKey(a,ka)) && inj-event(receiverKey(b,kb))
                                 && inj-event(senderTrustsR(a,b)) && inj-event(receiverTrustsS(b,a)).


(* Trust by Handshake:
 * If r decrypts from s, then r previously marked a successful matching with s's trstwds *)
query r:userId, s:userId, m:bitstring;
event(receiveGreen(r,s,m)) ==> event(receiverTrustsS(r,s)).
```

```
(* Privacy from Trusted Peer *)
(* Whenever a message from a trusted peer s is received unencrypted, then s did not send it.
 * We are assuming that pEp is by default enabled *)
query r:userId, s:userId, m:bitstring;
      event(decryptionFails(r,s,m)) && event(sendGreen(s,r,m)) ==> false.
(* Conversely, whenever r receives a message m from a trusted peer s,
 *then m is encrypted with his pk and s sent him m *)
query b:userId, a:userId, m:bitstring, pkB:pkey, z:bitstring;
      event(receiveGreen(b,a,z)) ==> event(sendGreen(a,b,z)) && z=aenc(m,pkB)
                                     && event(receiverKey(b,pkB)).


(* Integrity from Trusted Peer *)
(* Whenever a message from a trusted peer A is received unsigned, then A did not send it.
 * We are assuming that pEp is by default enabled *)
query r:userId, s:userId, m:bitstring;
      event(signVerifFails(r,s,m)) && event(sendGreen(s,r,m)) ==> false.
(* Conversely, whenever r receives a message m from a trusted peer s,
 * then m is encrypted with his pk and s sent him m *)
query b:userId, a:userId, m:bitstring, pkB:pkey, z:bitstring, skA:skey;
      event(receiveGreen(b,a,z)) ==> event(sendGreen(a,b,z)) && z=aenc(sign(m,skA),pkB)
                                     && event(senderPkey(a,skA)).


(* MITM Detection *)
(* Fake keys sent during the key exchange phase are detected *)
query a:userId, b:userId, ka:pkey, kb:pkey, pkA:pkey, pkB:pkey;
   event(endHandshakeUnsucc(a,b,ka,kb)) ==> (event(senderKey(a,pkA)) && pkA<>ka) ||
                                            (event(receiverKey(b,pkB)) && kb<>pkB) ||
                                             event(receiverMistrustsS(b,a)) ||
                                             event(senderMistrustsR(a,b)).


(* Secrecy *)
query attacker(mssg).



(*--- Processes ---*)

let receiveEmailT(r:userId, skR:skey, trustedPeer:userId, trstdKey:pkey) =
   in (outlook, (s:userId, m: bitstring));
   if s = trustedPeer   then  (
      let d = adec(m, skR) in (
         let v = verifSign(d, trstdKey) in (
            event receiveGreen(r,s,m);
            event end
         )
         else
            event signVerifFails(r,s,m)
      )
      else
```

```
         (* receive message not encrypted *)
         event decryptionFails(r,trustedPeer,m)
   )
   else
      (* receive message from peer not trusted *)
      event receivedFromNotTrusted(r,s).


(* A is a new pEp user *)
let senderA(idB:userId) =
   (* Key exchange *)
   new skA: skey;
   new m: bitstring;
   let idA = id(skA) in (
      event senderKey(idA,pubKey(skA));
      event senderPkey(idA,skA);
      out( internetAB, (idA,pubKey(skA),m) );
      in( internetBA, (usr:userId, res:bitstring) );
      if usr<>idB then
         event notReplied
      else
         let sgndm = adec(res,skA) in (
            let (t:bitstring, key:pkey) = getMssg(sgndm) in (
               let x = verifSign(sgndm,key) in (
                  (* store the public key 'key' of 'usr' as SECURE *)
                  event aAddsBsec(idA,usr,key);
                  event responseReceivedOk;
                  (* Handshake *)
                  let twds = trustwords(pubKey(skA),key) in
                  (* trustwords validation *)
                  event startHndshkS(idA,idB);
                  out(phoneB, (idA,twds));
                  in(phoneA, (prtnr:userId,confirmTrwds:bool,peertwd:bitstring));
                  if confirmTrwds then
                     (* to mark the end of the protocol between the 2 pep identities *)
                     if prtnr=idB && trustwordsMatch(twds,peertwd) then
                        (* set idA trusts prtnr*)
                        event senderTrustsR(idA,idB);
                        event endHandshakeOk(idA,idB,pubKey(skA),key);
                        (* from this point A can send any number of emails to B, secure and trusted *)
                        !(event sendGreen(idA,idB,aenc(sign(mssg,skA), key));
                          out(outlook, (idA, aenc(sign(mssg,skA), key)))  )
                     else
                        event senderMistrustsR(idA,prtnr);
                        event endHandshakeUnsucc(idA,prtnr,pubKey(skA),key)
                  else
                     event senderMistrustsR(idA,prtnr);
                     event endHandshakeUnsucc(idA,prtnr,pubKey(skA),key)
               )else
                  (* by default pEp peers send signed and encrypted emails *)
                  event signVerifFailed
            )else
               (* by default pEp peers send signed and encrypted emails *)
```

```
            event notSigned
        )else
            event notEncrypted
    ).


let receiverB(idB:userId, skB:skey) =
    (* Key exchange *)
    in(internetAB, (sndr:userId, pkSndr:pkey, text:bitstring));
    (* store the public key 'pkSndr' of the sender 'sndr' as SECURE *)
    event bAddsAsec(idB,sndr,pkSndr);
    new resp:bitstring;
    let txt = sign( (resp,pubKey(skB)), skB ) in (
        (* B sends his  public key signed and encrypted *)
        event receiverKey(idB,pubKey(skB));
        out( internetBA, (idB, aenc(txt,pkSndr)) );
        (* Handshake *)
        in(phoneB, (peer:userId, words:bitstring));
        event startHndshkR(idB,peer);
        if peer <> sndr then
            event unknownPeer
        else
            (* Compare with the stored pk *)
            let trustwds = trustwords(pkSndr,pubKey(skB)) in
                if trustwordsMatch(words,trustwds) then
                    event receiverTrustsS(idB,peer);
                    out(phoneA, (idB,true,trustwds));
                    (* B receives emails with 'peer' in the list of trusted peers *)
                    !receiveEmailT(idB,skB,peer,pkSndr)
                else
                    event receiverMistrustsS(idB,peer);
                    out(phoneA,(idB,false,trustwds));
                    (* B receives emails, no trusted peers added *)
                    event receiverDetectsMistrusted(idB,peer)
    ).


process
    (* B is an old pEp user, thus he has already a key pair *)
    new skB : skey;
    ( (!senderA(id(skB))) | (!receiverB(id(skB),skB)) )
```

A.3 Model for an aligned sense-of-security analysis of pEp

```
MODULE main

VAR
state: 0..18;
-- to be able to compare the trust level values, we map the domain into integers as follows:
-- i=0, n=1, s=2, st=3 , where 0 indicates the less secure and 3 the most secure
prvcS: 0..3;
prvcU: 0..3;
-- to compare the goal we will assign: fail=-1, ongoing=0, done=1
```

```
goal: {-1,0,1};
goalU:  {-1,0,1};
userAction: {new_email, input_data, enable_pep, disable_pep, do_handshake,
               send, cancel, confirm, mistrust, stop_trusting, end};

pepReceiver: boolean;
handshake: boolean;
trustwordsMatch: boolean;
confirmedTrustwords: boolean;


DEFINE
sendEmail := userAction=send;
--toFinalState := (state=3 & userAction=confirm) | (state=6 & sendEmail);
toFinalState := (state=3 & userAction=confirm) | (sendEmail & state!=2);

ASSIGN
init(state) := 0;
next(state) := case
state=0 & userAction=new_email : 1;
state=1 & userAction=input_data : 5;
state=1 & userAction=disable_pep : 2;
state=2 & userAction=enable_pep : 1;
state=2 & sendEmail : 3;
state=3 & userAction=confirm : 4;
state=3 & userAction=cancel : 2;
state=5 & pepReceiver & !handshake : 6;
state=5 & pepReceiver & handshake : 7;
state=5 & !pepReceiver : 8;
state=6 & sendEmail : 9;
state=6 & userAction=do_handshake : 12;
state=7 & !confirmedTrustwords : 8;
state=7 & confirmedTrustwords : 10;
state=8 & sendEmail: 4;
state=10 & sendEmail : 11;
state=10 & userAction=stop_trusting : 6;
state=12 & userAction=cancel : 6;
state=12 & trustwordsMatch : 13;
state=12 & !trustwordsMatch : 14;
state=13 & userAction=confirm : 10;
state=13 & userAction=mistrust : 17;
state=14 & userAction=confirm : 15;
state=14 & userAction=mistrust : 8;
state=15 & sendEmail : 16;
state=17 & sendEmail : 18;
-- in the model of the system the final states go back to 0, which simulates that
-- the system can run again; since here we are analysing a single execution,
-- we don't want that repetition, thus such states don't have a next one
TRUE : state;
esac;

init(prvcS) := 1;
next(prvcS) := case
next(state)=2 | next(state)=3 | next(state)=4 | next(state)=8 | next(state)=17 | next(state)=18: 0;
```

```
next(state)=6 |  next(state)=9 | next(state)=12 | next(state)=13 | next(state)=14: 2;
next(state)=10 | next(state)=15 | next(state)=16 | next(state)=11: 3;
TRUE : 1;
esac;

init(prvcU) := 1;
next(prvcU) := case
next(state)=2 | next(state)=3 | next(state)=4 | next(state)=8 | next(state)=17
              | next(state)=18: 0;
next(state)=6 |  next(state)=9 : 2;
next(state)=10 | next(state)=13 | next(state)=14 | next(state)=15 | next(state)=16
              | next(state)=11: 3;
TRUE : 1;
esac;

init(userAction) := new_email;
next(userAction) := case
userAction=new_email : {input_data,disable_pep};
next(state)=2 : {enable_pep,send};
userAction=enable_pep : {disable_pep,input_data};
state=2 & sendEmail : {cancel,confirm};
next(state)=6 : {send,do_handshake};
userAction=do_handshake : {cancel,confirm,mistrust};
next(state)=10 : {send,stop_trusting};
next(state)=15 | next(state)=17 | next(state)=8: send;
toFinalState : end;
TRUE : userAction;
esac;

init(goal) := 0;
next(goal) := case
next(state)=4 : -1;
(state=6 | state=15 | state=17) & sendEmail : -1;
state=10 & sendEmail : 1;
TRUE : goal;
esac;

init(goalU) := 0;
next(goalU) := case
next(state)=4 : -1;
(state=6 | state=15 | state=10) & sendEmail : 1;
state=17 & sendEmail : -1;
TRUE : goalU;
esac;

init(pepReceiver) := FALSE;
next(pepReceiver) := case
--userAction=input_data : {TRUE,FALSE};
next(state)=5 : {TRUE,FALSE};
TRUE : pepReceiver;
esac;

init(handshake) := FALSE;
next(handshake) := case
```

```
--userAction=input_data : {TRUE,FALSE};
next(state)=5 : {TRUE,FALSE};
TRUE : handshake;
esac;

init(confirmedTrustwords) := FALSE;
next(confirmedTrustwords) := case
state=5 & pepReceiver & handshake : {TRUE,FALSE};
TRUE : confirmedTrustwords;
esac;

init(trustwordsMatch) := TRUE;
next(trustwordsMatch) := case
trustwordsMatch=FALSE : FALSE;
next(state)=12 : {TRUE,FALSE};
TRUE : trustwordsMatch;
esac;



------ Check correctness of the model -----
-- No other state is reachable from a final state apart from itself
--SPEC AG (goal!=0 -> (AG goal!=0))
-- There is always a way for the user to determine if the goal is reached or failed
--SPEC AG EF goalU!=0


------- Alignment in Security goals ---------
-- The belief of the user about the accomplishment of the goal is always either accurate
-- or less optimistic than the reality
--SPEC AG goalU <= goal

-- In every state, the security implemented is higher or equal than the user's evaluation
--SPEC AG prvcS = prvcU
--SPEC AG prvcU <= prvcS

-- Even if the evaluation of the aspects is aligned all the time, the understanding of the
-- achievement of the goal can be misaligned.
-- Note that this property is false already if AG goal=goalU is true
--SPEC E [prvcU<=prvcS U goalU!=goal]

-- If the trustwords do not match
--SPEC AG (!trustwordsMatch -> !(EF goalU=reached))
--SPEC AG (!trustwordsMatch -> AG goalU!=1)
SPEC AG (!trustwordsMatch -> AG prvcU!=3)
SPEC AG (!trustwordsMatch -> AG prvcS!=3)
```