CONTINUOUS AND TRANSPARENT AUTHENTICATION

A SECURE AND PASSWORD-LESS EXPERIENCE USING WEARABLES

JACOB BENJAMIN CHOLEWA AND RASMUS LOMHOLT WISMANN

Department of Computer Science, IT University of Copenhagen

June 2017

Jacob Benjamin Cholewa and Rasmus Lomholt Wismann: *Continuous and Transparent Authentication,* A secure and password-less experience using wearables, © June 2017

SUPERVISOR:
Alessandro Bruni

ABSTRACT

Password-based authentication is the most widely used method of authentication, even though research for decades have concluded that it is insecure and user-unfriendly. The purpose of this work is to investigate existing research into alternative authentication schemes, and based on a semi-structured review of related work, propose a new design for an authentication scheme, that has a balanced focus on both usability and security aspects. The proposed solution is a continuous and transparent authentication scheme designed to run on consumer-level wearable and mobile technology. The scheme is facilitated by an authentication protocol, that is formally verified and analyzed using a mixed computational and symbolic model approach. To showcase the feasibility of the design, a prototype is implemented using the protocol. The authentication scheme and protocol proposed in this thesis are designed with the intention of being a complete password replacement, and could realistically be implemented as a mature solution.

This thesis is a joint work, however, in the later refinement and writing stages of the thesis, Jacob focused on protocol design and verification, while Rasmus focused on analysis, scheme design and implementation.

Our academic background is five years of studies in computer science and software development. Our majors are Pervasive Computing and Critical Systems respectively, and so our knowledge of both security and cryptography was very limited before starting this project.

However, our strong background in pervasive and distributed computing systems, and how users interact with such systems, gives rise to this cross disciplinary work that bring a rare user-oriented perspective to an otherwise very security dominated research area.

Last but not least, we wish to thank our supervisor Alessandro Bruni — Thanks for giving us the opportunity to work with you. Your continued presence, enthusiasm, and many insights have been an invaluable source of knowledge, inspiration and motivation. We could not have wished for a better supervisor.

Denmark, June 2017

Jacob Benjamin Cholewa and
Rasmus Lomholt Wismann

CONTENTS

I INTRODUCTION & RELATED WORK 1

1 Introduction 3
- 1.1 Contributions 4
- 1.2 Outline 4

2 Related Work 5
- 2.1 Authentication Schemes 5
  - 2.1.1 The Problem With Passwords 5
  - 2.1.2 Why Passwords Have Not Yet Been Replaced 6
- 2.2 Pervasive Authentication 7
  - 2.2.1 Wearable Authentication 8
  - 2.2.2 Possession-Based Authentication 8
  - 2.2.3 Continuous and Transparent Authentication 9
- 2.3 Cryptographic Assumptions 9
  - 2.3.1 Cyclic Groups 9
  - 2.3.2 The Discrete-Logarithm Assumption 10
  - 2.3.3 The Decisional Diffie-Hellman (DDH) Assumption 11
  - 2.3.4 The Eavesdropping Indistinguishability Experiment 11

3 Pervasive Authentication Schemes 13
- 3.1 Context-Aware Authentication 13
- 3.2 Wearable Authentication 14

II CONTINUOUS & TRANSPARENT AUTHENTICATION 17

4 Scheme Design 19
- 4.1 Design Choices 19
  - 4.1.1 Learning From Passwords 19
  - 4.1.2 Transparency 19
  - 4.1.3 Adjustable User Awareness and Explicit Consent 20
  - 4.1.4 Why Wearables? 21
  - 4.1.5 Proximity 22
  - 4.1.6 Continuity 22
  - 4.1.7 Authenticator and Sibling 23
- 4.2 The Scheme in Practice 24
- 4.3 User Scenarios 25
- 4.4 Scheme Evaluation 25
  - 4.4.1 Extending the Evaluation Framework 25
  - 4.4.2 Rating Process 29
  - 4.4.3 Extending the Review 31

5 Protocol Design 33
- 5.1 Definition of Authentication 33
- 5.2 Design Rationale 33
  - 5.2.1 Distributed Authentication 34
- 5.3 Partial Crypto Systems 35
- 5.4 The Protocol 36

5.4.1 Registration 37
5.4.2 Authentication 37
6 Security Analysis 39
6.1 Security in the Computational Model 39
6.1.1 Weak External Observation 40
6.1.2 Weak Internal Observation 41
6.1.3 Strong External Observation 42
6.1.4 Strong Internal Observation 43
6.2 Security in the Symbolic Model 43
6.2.1 The Tamarin Prover 44
6.2.2 Modeling our Protocol in Tamarin 44
6.2.3 Equational Theory 45
6.2.4 Proving Injective Agreement and Unforgeability 47
7 Implementation 51
7.1 Goal and Scope 51
7.2 Solution Description 51
7.3 Technical Description 53
7.3.1 Technologies Used 53
7.3.2 Algorithms 55
7.3.3 Cryptography Implementation 57
7.3.4 Token Issuing and Verification 58
7.4 Prototype Evaluation 58
7.4.1 Rating process 58

III DISCUSSION & CONCLUSION 61
8 Discussion 63
8.1 Security Assumptions and Attacks 63
8.2 Confidence in Tamarin 64
8.3 Trusted Input, Output and Computations 65
8.4 Number of Siblings 66
8.5 Recovery Problem 66
8.6 Reflecting on Evaluation Results 68
9 Conclusion 69

REFERENCES 71

IV APPENDIX 73
A Android Cryptography Implementation 75
B Server Cryptography Implementation 81
C Server JWT Implementation 83
D Tamarin Model 85
E Strong Internal Observation Attack 89

LIST OF FIGURES

Figure 1	Example of Explicit User Consent	21
Figure 2	Component overview for the propose scheme	25
Figure 3	Registration sequence diagram	37
Figure 4	Authentication sequence diagram	38
Figure 5	Example of notifications on the watch	52
Figure 6	Example of explicit consent on the phone	53
Figure 7	The pairing menu in Chrome	54
Figure 8	Authenticating with a service in Chrome	54
Figure 9	Sequence diagrams of known attacks	63
Figure 10	Tamarin trace of SIO attack	89

LIST OF TABLES

Table 1	Overview of benefits of related work	14
Table 2	Overview of device requirements	24
Table 3	Scenario: Registration	26
Table 4	Scenario: Unlocking	26
Table 5	Scenario: Authentication	27
Table 6	Scenario: Theft or loss	27
Table 7	Overview of benefits of our design	30
Table 8	Definitions of adversarial capabilities	48
Table 9	Overview of benefits of our prototype	59

Part I

INTRODUCTION & RELATED WORK

1 INTRODUCTION

Authentication is a cornerstone of computer systems, and is the process by which a users identity is verified. While most areas of computing continue to evolve rapidly, the dusty old way that we authenticate with computers has not changed for decades.

Usernames and passwords have their origin from back when multiple users were accessing the same mainframe and the operating systems most important job was to handle its different users and their individual files and programs. With the way we use computers today, and especially because of their now ubiquitous presence in almost everything we do, it is surprising that password-based authentication, that was originally designed for such radically different usage, is still the behemoth used practically everywhere.

> *The continued domination of passwords over all other methods of end-user authentication is a major embarrassment to security researchers. As web technology moves ahead by leaps and bounds in other areas, passwords stubbornly survive and reproduce with every new web site.*
>
> — Bonneau et al. [9]

Password-based authentication is known to be sub-optimal in both its usability and security. Already in 1979 Morris and Thompson [18] showed how they were able to crack 81% of 3000 user generated passwords, using only a small dictionary. A clear impediment is that good security implies a strict and rigours behavior from the end-user, that only a small minority of users are even cognitively capable of [24].

As authentication is merely an enabling task before using a system, many users consider it a hindering, and either unknowingly or purposefully chooses to compromise security for better convenience and usability (such as using the same short password for all services). It is time for a long overdue paradigm shift.

Pervasive Authentication is a research topic, in particular, exploring how to reduce or ease the user-interaction when authenticating, and aims for seamless, but secure authentication. The proof-*of*-concept by Ojala, Keinanen, and Skytta [20] shows how a watch (wearable) can provide an inexpensive, effective, and usable way of authenticating with systems. The user *transparently* authenticates with the system by simply approaching a client. The session is *continuously* kept alive, and as soon as the user leaves the client, the session is autonomously terminated. When the watch is put on, it first needs to be unlocked. This is done using the user's fingerprint where after it starts monitoring the wearer's vitals. If the watch is stolen, or in any way removed from the wearer's arm, it locks itself. In this way it provides (seem-

ingly) secure authentication in a perfectly seamless and transparent manner, and shows how usable and effortless authentication can be.

However, much work is still left; first of all the proof-*of*-concept is designed for authenticating with a client machine, while most real-world authentications is in fact with web-services [14]. Furthermore no security protocol is utilized, and existing protocols are not suitable. Last but not least, for the concept to mature, there is a need for designing for commodity hardware and services.

1.1 CONTRIBUTIONS

This thesis has two main contributions. First of all we conduct a semi-structured review of several existing designs for pervasive authentication schemes. Based on the review, we put forward a requirement analysis and a design for a modern Continuous and Transparent Authentication (CTA) scheme. Our design is the first that targets existing commodity hardware, and could realistically be implemented as a mature solution.

Secondly we propose an authentication protocol that can facilitate an implementation of the design. The protocol is formally verified and analyzed using a mixed computational and symbolic model approach. To showcase the feasibility of core features of the design, a prototype is implemented using the protocol.

1.2 OUTLINE

This thesis is divided into three parts: 1) Introduction and Background, 2) Continuous and Transparent Authentication, and 3) Discussion and Conclusion.

The introduction and background introduces and motivates the problem of password-based authentication, and hints to a solution using wearable devices for end-user authentication. Next, relevant related work is presented and reviewed.

The second part presents our proposed design and implementation. The design is divided into two parts: The first part introduces a novel authentication scheme. The second part presents a protocol supporting the design. Lastly a prototype implementation is presented. The third part discusses and concludes on the presented work.

All material from this thesis is available at

https://github.com/cholewa1992/cta

2 RELATED WORK

> *Authentication denotes the process by which the identity of a user (or any subject) is verified. In this process, the user provides his claimed identity together with evidence in the form of credentials. If the authenticating system accepts the evidence, the user is successfully authenticated.*
>
> — Basin, Schaller, and Schläpfer [4]

2.1 AUTHENTICATION SCHEMES

2.1.1 *The Problem With Passwords*

A large emphasis has historically been placed on technical aspects of security, when it comes to designing and evaluating authentication schemes. Obviously, it is desirable to provide secure authentication, since its entire purpose is to deny or grant access to some protected resource. However, when it comes to designing authentication schemes for use in practice, usability is often an overlooked factor. It is not uncommon that schemes, which are secure in theory, can turn out to be very insecure in practice, if user behavior is not taken into consideration.

This is, in particular, true for schemes which rely on passwords. If a user chooses a weak password, it does not matter how secure the rest of the chain in the authentication process is.

Passwords are the most common security mechanisms for end user authentication in computer systems today, and has been so, for a very long time. In 1979 Morris and Thompson [18] showed how they were able to crack 81% of 3000 user generated passwords, using a small dictionary only.

With the increased computing power and much more developed tools available today, such as rainbow tables [19] and huge databases of known passwords, it has become trivial to crack most user generated passwords.

Adams and Sasse [1] advocate the need for user-centered design when it comes to security mechanisms. In a field study, they investigate user behavior and authentication practices within organizations that use passwords. Their findings show, that stricter password policies does not necessarily make users produce stronger and more secure passwords. This can lead to bad practices, such as writing down passwords on post-it notes or composing passwords of easily acquirable personal details, such as family member names.

Usage pattern such as these, will reduce security, due to how easy it would be for an attacker to acquire or guess the password.

Another significant problem with passwords, is that, even if an extraordinary user is cognitively capable of remembering many different, complex and unique passwords, and replacing them regularly, research still shows that users will choose the convenience of short, weak passwords over strong and secure ones, simply to reduce the effort and time required to authenticate.

Weirich and Sasse [24] explains the concept of authentication from a user's perspective as an "enabling task", that is only a means to achieve the "primary task", which is accessing the protected resource. Thus the authentication process creates an overhead in getting to the "primary task". They argue that the willingness for users to behave in accordance with password practices, is small because many users do not recognize the threat of password compromisation, as being high enough to outweigh the benefits of being able to quickly type the password and being easier to remember.

2.1.2 *Why Passwords Have Not Yet Been Replaced*

With all problems that come with passwords, the natural question to ask is; why have passwords not been replaced, by some alternative mechanism that is more secure in practice? Bonneau et al. [9] tries to answer that question, by exploring 35 different authentication schemes from the real world, to see how they perform in comparison to passwords. They define 25 different properties, within the categories of usability, deployability and security, where each scheme receives a rating for each property. Each scheme is rated by specifying, that it can either offer the property, partially offer the property (quasi) or not offer the property.

Bonneau et al. [9] propose that the methodology they use, and the different properties defined, can be used as a framework, for rating future schemes, to get an insight on how well it performs, in comparison to the different schemes they explored in their research.

In addition to providing an evaluation framework, the results of the scheme ratings in the paper, also reveal some insight into the state of current authentication technologies. There seems to be a clear indication, that schemes with many usability properties often have fewer security properties and vice versa. Another interesting finding from their research, is that there actually exists alternatives to passwords, that achieve higher overall ratings in terms of both usability and security. However, these solutions often lack behind in deployability. In other words, it appears to be a non-trivial task to find an authentication scheme, that provides a lot of benefits, without missing out in some area. Bonneau et al. [9] argues that the main reason why passwords are still the most widely used authentication mechanism, is because current alternatives only offer marginal gains, which is simply not enough to transition away from passwords on a global scale, especially not considering the cost and effort it would require.

2.2 PERVASIVE AUTHENTICATION

Pervasive Computing (also sometimes referred to as ubiquitous computing), entails the vision of computers, that are available throughout the physical environment embedded into everyday artifacts [25]. This vision is becoming more of a reality, as the number of personal portable devices we carry around with us increases more than ever. Many of the artifacts used in everyday life are now digital and part of the 'Internet of Things', and the average user is increasingly interacting with more and more computers throughout their day, making our lives ever more connected. Pervasive computing is often focused on building technology that assists the user's everyday life. Importantly, the technology should fade away and seamlessly support the user in his primary task. Clearly, there is room for improvements in this regard when it comes to authentication.

> *Imagine that a user would need to type in username and password on all 'pervasively' available computers before he could start using them. Clearly, if the pattern of login and logout is not considered a usability problem today, it will most certainly become one in the years to come.*
>
> — Bardram, Kjær, and Pedersen [3]

The fact that user interaction frequency with computers increase, and you still need to verify your identity every time before you can actually use the computer for the intended purpose, does not harmonize well with how most authentication methods work today, which is all very explicit and time-consuming in terms of required user interaction.

Furthermore, user authentication is not only limited to unlocking access to a client device. Once a user is logged into a client, it is very likely that he wants to access one or more web services, that again will require him to perform a round of authentication. Hayashi and Hong [14] found that user authentication against web services, is by far the most common type of authentication in everyday usage patterns.

Research projects within pervasive computing, aim to alleviate this problem, by providing mechanisms and technologies that allow for authentication, by requiring only a little or no user interaction at all. This concept is called 'transparent authentication' – it allows faster and more effortless authentication for users. Bardram, Kjær, and Pedersen [3] presents a proximity-based authentication scheme, which is targeted at a nomadic hospital working environment, where multiple users, use the same computers. In this environment, authentication is done frequently, and use of computer systems is often done in short sessions. The authentication scheme is based on proximity as well as a physical key-card placed in a bay at the workstation. If the location of a user in the system corresponds to the place where his key-card is used, he is authenticated.

Their design highlights the need for *proximity-based* authentication where the user can merely approach a work station and start using

it. This allows the pediatricians to quickly access information without the need for explicit authentication. Secondly, they note that *silent* sessions, where users seamlessly alternate being between authenticated or not, are important to the user experience. If you can seamlessly authenticate by approaching the workstation, then it is equally important to logout the user when vacating the machine [2].

2.2.1 *Wearable Authentication*

The 2008 paper by Ojala, Keinanen, and Skytta [20] uses a similar approach, but with a wearable wristband used as authentication mechanism instead. The system is biometric based and uses a fingerprint reader, along with monitoring of heart rate and skin temperature, to continually verify that the legitimate user is in possession of, and wearing the authenticator. Back then wearable computers, often just referred to as wearables, were mainly just a research area within pervasive computing. In the last few years, it has gone from being only a research area, into being consumer products. Bianchi and Oakley [7] describes how the form factor of this technology opens new possibilities and challenges in terms of security, and how wearables can be used for authentication purposes. Wearables are in many ways ideal for CTA, since they provide the benefit of two authentication factors: possession and inherence. Possession, since it is something you carry or wear, and inherence, since it can capture biometric data. This is possible because wearables are closely located or even attached to a user's body, which means that they can be designed to unobtrusively and continuously acquire biometric data from the user. Furthermore, it is possible to detect when a device is equipped and when it is removed, which provides a higher level of confidence about the user's identity, for a period of time. [7]

2.2.2 *Possession-Based Authentication*

Stajano [22] presents a solution to replace passwords using a hardware token called *Pico*. Pico is, the idea of, a small portable device that runs on battery, has a camera and two buttons for registering and authenticating. The device is wirelessly communicating with the client used by the end-user. When the user wants to authenticate with a service, a QR code appears on the screen. When the Pico reads the code the user is authenticated.

Two features are of importance. Hardware tokens are prone to theft. To mitigate the risk Pico only unlocks in the presence of k-*out-of*-n siblings. A sibling is a small RF-enabled wearable token that the user can carry, such as a watch, jewelry, glasses etc. The Pico needs to have a least k siblings in its proximity to unlock, and automatically locks again if the siblings should disappear. Thereby the adversary would have to steal several tokens, of the individual user, to authenticate on his behalf.

2.2.3 *Continuous and Transparent Authentication*

Both Bardram, Kjær, and Pedersen [3] and Ojala, Keinanen, and Skytta [20] solutions are examples of transparent user authentication, where the traditional knowledge factor, is substituted by factors of possession and inherence, and the actions required from the user, to perform authentication is reduced[1]. Furthermore [20] is also an example of a continuous authentication scheme, which means that after initial verification of the user, the identity of the user is continually verified throughout the entire session of the authentication. In case a user's identity can no longer be correctly verified, the session should terminate in a similar manner to classic log out. In [3], the definition of silent authentication is similar to that of continuous, with the difference that only the event of inserting and removing a key-card is monitored, and hence the user could move away from the workstation with his key-card still inserted. The Pico [22] is continuous in terms of it locking if less than k siblings are in its proximity, but not in terms of terminating a client's session if the Pico is locked. Thus it is a *weaker* definition of continuous than that of [20]. However, the latter is also limited to authenticating with a client, where Pico is authenticating with a web-service *through* a client.

2.3 CRYPTOGRAPHIC ASSUMPTIONS

Modern cryptography relies heavily on certain mathematical assumptions. In this section we will present a few assumptions that are needed, to understand the remainder of this thesis. These definitions are taken from "Introduction to Modern Cryptography" by Katz and Lindell [15], and the following section is interchangeably quoting the textbook.

2.3.1 *Cyclic Groups*

Let $\mathbb{G}$ be a finite set of elements. A set of elements and some binary operator $\circ$ is an abelian group if [15, page 291]

- there is closure such that $\forall(g, h) \in \mathbb{G}, g \circ h \in \mathbb{G}$,
- exists an identity element e such that $\forall(g) \in \mathbb{G}, e \circ g = g$,
- exists inverses such that $\forall(g, h) \in \mathbb{G}, g \circ h = e$,

and the operator is both

- associative $(\forall(g_1, g_2, g_3) \in \mathbb{G}, (g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3))$,
- and commutative $(\forall(g, h) \in \mathbb{G}, g \circ h = h \circ g)$.

1 Note that we can have different degrees of transparency. Later we introduce a more fine-grained definition.

MULTIPLICATIVE GROUPS AND PRIME ORDER GROUPS One kind of abelian group is multiplication groups. These groups are denoted by $\mathbb{Z}_n^*$, and is given as all elements relatively prime to n

$$\mathbb{Z}_n^* \stackrel{def}{=} \left\{ x \in \{1, ..., n-1\} \;\middle|\; gcd(x, n) = 1 \right\}$$

We say that the group is a multiplicative group, (denoted by $*$) if the operator is multiplication module n ($ab \stackrel{def}{=} [ab \bmod n]$), and the identity element is 1 [15, page 295].

If n is prime, then $\mathbb{Z}_p^* = \{1, ..., p-1\}$, as all elements less than p are relatively prime to p. The order of such group is then given as $|\mathbb{Z}_p^*| = p - 1$. These groups are called primer-order groups.

CYCLIC GROUPS We say that an element g is a generator of the cyclic group $\mathbb{G}$ of order $|\mathbb{G}| = m$, if all elements of the group can be expressed as $\langle g \rangle = \{g^0, g^1, ..., g^{m-1}\}$, and $g^0 = g^m$ [15, page 316].

A cyclic group can be constructed using prime order groups. This is due, to the fact that, if the order of the finite group $\mathbb{G}$ is prime, then it is cyclic. Furthermore, all elements except for the identity element in such a group is a generator of the group [15, page 321].

Definition 2.1. *Let $\mathcal{G}$ denote an algorithm that on input of a security parameter 1^n finds two large primes p and q, such that q divides $(p-1)$ and $\|q\| = n$, and then outputs the definition of a cyclic group $\langle \mathbb{G}, q, g \rangle$, where g is the generator of the group, q is the order of the group, and the group is given as*

$$\mathbb{G} \stackrel{def}{=} \left\{ \left[h^{(p-1)/q} \bmod p \right] \;\middle|\; h \in \mathbb{Z}_p^* \setminus \{1\} \right\}$$

2.3.2 *The Discrete-Logarithm Assumption*

This can be any cyclic group

Let $\mathcal{G}$ denote a polynomial-time group-generation algorithm that on input 1^n, outputs a description of a cyclic group $\langle \mathbb{G}, q, g \rangle$, where $\mathbb{G}$ is the group, q is the order (with $\|q\| = n$), and g is the generator. For every element $h \in \mathbb{G}$ there exists a unique $x \in \mathbb{Z}_p$ such that $g^x = h$. The inverse operation is called the *discrete logarithm* and is denoted $x = \log_g(h)$. Consider the following experiment: [15, page 320]

$\mathbb{Z}_p$ is an additive group with $e = 0$.

The discrete-logarithm experiment $\mathrm{DLog}_{\mathcal{A},\mathcal{G}}(n)$**:**

1. Run $\mathcal{G}$ to obtain $\langle G, q, g \rangle$, where $\mathbb{G}$ is a cyclic group of order q with (with $\|q\| = n$), and g is a generator of $\mathbb{G}$.
2. Choose a uniform $h \in \mathbb{G}$.
3. $\mathcal{A}$ is given $\mathbb{G}, q, g, h$, and outputs $x \in \mathbb{Z}_q$.
4. The output of the experiment is defined to be 1 if $g^x = h$, and 0 otherwise.

Definition 2.2. *We say that the discrete-logarithm problem is hard relative to $\mathcal{G}$ if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that*

$$\Pr\left[\mathrm{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1\right] \leqslant \mathrm{negl}(n)$$

In other words; The mapping $f : \mathbb{Z}_n \to \mathbb{G}$ given by $f(a) = g^a$ is an isomorphism between $\mathbb{Z}$ and $\mathbb{G}$, and is easy to compute, while the inverse mapping $f^{-1} : \mathbb{G} \to \mathbb{Z}_n$ (although it exists) is infeasible to compute. This one-way behavior is fundamental in public-key cryptography.

2.3.3 *The Decisional Diffie-Hellman (DDH) Assumption*

For a cyclic group $\mathbb{G}$, a generator $g \in \mathbb{G}$, and two elements $h_1, h_2 \in \mathbb{G}$, we define $\mathrm{DH}_g(h_1, h_2) \stackrel{def}{=} g^{\log_g(h_1)\cdot\log_g(h_2)}$. If $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$, then

$$\mathrm{DH}_g(h_1, h_2) = g^{x_1 \cdot x_2} = h_1^{x_2} = h_2^{x_1}$$

The DDH problem is, given h_1, h_2 and h', to decide if h' is equal to $\mathrm{DH}_g(h_1, h_2)$, or a uniform element in $\mathbb{G}$ [15, page 321].

Definition 2.3. *We say that the DDH problem is hard relative to $\mathcal{G}$ if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl such that*

$$\left|\Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^x, g^y, g^z\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^x, g^y, g^{xy}\right) = 1\right]\right| \leqslant \mathrm{negl}(n)$$

It follows, that if the discrete-logarithm problem is easy, then so is the DDH problem. While there is no proof of the discrete logarithm being hard, no efficient algorithm is known.

2.3.4 *The Eavesdropping Indistinguishability Experiment*

Definition 2.4. *A public-key encryption scheme is a triple of probabilistic polynomial-time algorithms (Gen, Enc, Dec) such that:*

1. *The key-generation algorithm Gen takes as input the security parameter 1^n and outputs a pair of keys $(\mathrm{pk}, \mathrm{sk})$. We refer to the first of these as the public-key and the second as the private-key. We assume for convenience that pk and sk each has length at least n, and that n can be determined from pk, sk.*

2. *The encryption algorithm Enc takes as input a public-key pk and a message m from some message space (that may depend on pk). It outputs a ciphertext c, and we write this as $c \leftarrow \mathrm{Enc}(m, \mathrm{pk})$.*

3. *The decryption algorithm Dec takes as input a private-key sk and a ciphertext c, and outputs a message m or a special symbol $\perp$ denoting failure. We write this as $m := \mathrm{Dec}(c, \mathrm{sk})$.*

It is required that, except possibly with negligible probability over (pk, sk) *output by* $Gen(1^n)$*, we have* $Dec(Enc(m, pk), sk) = m$ *for any (legal) message* m*.*

Encryption schemes are defined to be secure against chosen-plaintext attacks (CPA), if adversaries capable of obtaining plaintext/ciphertext pairs of its choice [15, page 20], are not able to distinguish between the encryption of two known messages. More formally, for a public-key encryption protocol $\Pi = (Gen, Enc, Dec)$ and an adversary $\mathcal{A}$, consider the following experiment: [15, page 378]

The eavesdropping indistinguishability experiment $PubK^{eav}_{\mathcal{A},\Pi}(n)$**:**

1. $Gen(1^n)$ is run to obtain keys (pk, sk).
2. Adversary $\mathcal{A}$ is given pk, and outputs a pair of equal-length messages m_0, m_1 in the message space.
3. A uniform bit $b \in \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow Enc(m_b, pk)$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext.
4. $\mathcal{A}$ outputs a bit b'. The output of the experiment is 1 if $b' = b$, and otherwise 0. If $b' = b$ we say that $\mathcal{A}$ succeeds.

Definition 2.5. *A public-key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *has indistinguishable encryptions in the presence of an eavesdropper if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* $negl$ *such that*

$$Pr\left[PubK^{eav}_{\mathcal{A}',\Pi}(n) = 1\right] \leqslant \frac{1}{2} + negl(n)$$

3 PERVASIVE AUTHENTICATION SCHEMES

In this chapter, we introduce a semi-structured review through pervasive authentication schemes. We evaluate the schemes, by rating how well they offer different benefits, that we consider relevant for any human-to-computer authentication scheme. The evaluation is based on the comparative evaluation framework for web authentication schemes presented by Bonneau et al. [9] (and introduced in the previous chapter). The purpose of this evaluation, is to identify the benefits of pervasive authentication, and highlight potential shortcomings. We evaluate the schemes of both Bardram, Kjær, and Pedersen [3] and Ojala, Keinanen, and Skytta [20]. The Pico authentication scheme [22] and passwords have both already been evaluated in the original paper [9]. The ratings are shown in table 1.

We have attempted to rate the schemes fairly, by adhering to the property definitions and rating criteria specified by the framework. Additionally, we have attempted to document and describe the rating process in the same way as done in the supplementary technical report [10].

3.1 CONTEXT-AWARE AUTHENTICATION

Bardram, Kjær, and Pedersen [3] presents a prototype and authentication protocol for secure and usable authentication for physicians in hospitals. The system is comprised of a personal smart-card, that can be inserted into the hospital computers to access the computers, and a context-aware subsystem that as a minimum is location-aware. If the practitioners try to access a computer using their key-card, and their location is the same as the workstations, then they are authenticated without further interaction. If the location differs, then they are asked to type their password.

When a new key-card is initialized it generates a public-private key pair and sends the public key to a central server. The key-card uses a one-way authentication protocol and the user's password is only known to the keycard.

We grant the system *Quasi-Memorywise-Effortless* as the user is still required to remember the keycard password. It is *Scalable-for-Users* as the card could easily submit the same public-key to many verifiers. It is not *Noting-to-Carry*, although, in the hospital setup where it is applied, the staff is required to carry their identity card, and it could qualify for a *Quasi-Noting-to-Carry* in some scenarios. It is both *Easy-to-Learn*, *Efficient-to-Use* and *Infrequent-Errors* (assuming that the context-aware service works most of the time). It is not *Easy-Recovery-from-Loss* as a new card needs to be issued, and a new public-private key pair needs to be created and submitted to verifiers.

		Usability								Deployability						Security											
	Reference	*Memorywise-Effortless*	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*
Passwords				●		●	●	○	●	●	●	●	●	●	●		○						●	●	●	●
Context-Aware Auth	[3]	○	●		○	●	●	●		●					●	○	●			○		○	○		●	
Wearable Auth	[20]	●	●	○	○	●	●	●		●					●	●	●	●	●	?	?	●	●	?	○	?
Pico	[22]	●	●		●		○	○							●	●	●	●	●	●	●	●	○	●	●	●

● = offers the benefit ○ = almost offers the benefit ? = not known

Table 1: Comparing the benefits of related work in Pervasive Authentication.

As it is a prototype, deployability benefits are not well documented. However, we grant it *Accessible* and *Non-Propritary*. The system is not *Negligible-Cost-per-User* as the setup is very infrastructure heavy. It requires all employees to both have a key-card and some RF-'badge' for the context-server to estimate a user's indoor location.

The system is not built to access web services and is, therefore, neither *Browser-Compatible* nor *Server-Compatible*. However, it could easily be used for web services by transmitting the users public key to every verifier, or even generate a new key-pair for every verifier. It would however still not be compatible.

On the security aspects, we deem it to be *Quasi-Resilient-to-Physical-Observation* as the user only rarely types the password. However, if the key-card is stolen and the password is known, the adversary has full access, and we, therefore, grant it *Quasi-Resilient-to-Theft*. It is not *Resilient-to-Phishing* as man-in-the-middle (MITM) attacks are possible. It is not *Resilient-to-Throttled-Guessing* nor *Resilient-to-Unthrottled-Guessing*, however, the adversary would have to steal the key-card to start guessing. It is not *Unlinkable*.

3.2 WEARABLE AUTHENTICATION

Ojala, Keinanen, and Skytta [20] presents a prototype for transparent and continuous authentication with workstations. The system is comprised of three components. A ZigBee enabled wearable wrist device that monitors the wearer's vitals, a ZigBee receiver, and the workstation. When the user puts on the watch, it starts to monitor his vitals. The user can now use a fingerprint reader to authenticate with the system. The user remains authenticated for as long, as he is wearing

the watch. If he takes off the watch, or his vitals stops, then he will be logged-out after 10 seconds. While the user is authenticated he can approach any workstation that has a receiver, and without further interaction start using the machine. As soon as he leaves the machine – he is logged out.

We grant the system *Memorywise-Effortless*, *Scalable-for-Users*, *Easy-to-Learn*, *Efficient-to-Use* and *Infrequent-Errors*. We deem it *Quasi-Nothing-to-Carry* as a watch is something that most users always carry, like for some people a smartphone. It is not *Easy-Recovery-from-Loss* as loosing the watch means having to get a new one, that should then be linked to the users existing identities.

As the system, much like ´Context-Aware Authentication' [3] is a prototype that is not built for web-services, we grant it the same scores for deployability.

On the security side it is *Resilient-to-Physical-Observation*, *Resilient-to-Targeted-Impersonation*, *Resilient-to-Throttled-Guessing*, *Resilient-to-Unthrottled-Guessing* and *Resilient-to-Theft*. It is *Quasi-Requiring-Explicit-Consent* as the user only gives explicit consent once when using the fingerprint reader. Other security aspects are not known due to the simplicity of the prototype and are therefore left out of consideration, although we deem them feasible to include.

Part II

CONTINUOUS & TRANSPARENT AUTHENTICATION

4

SCHEME DESIGN

The purpose of the following section is to describe the solution we propose for enabling usable and secure user authentication, but also to document the analytic work and choices we made in the design process. Based on our general scheme design and user scenarios, we conduct an evaluation of our design and the different benefits it provides.

4.1 DESIGN CHOICES

4.1.1 *Learning From Passwords*

The history of authentication schemes has shown, that usability plays a major role in the adoption and practical security of a scheme. During our entire design process, we have carefully aimed to prioritize both usability and security factors, and to continually consider the impact certain design decision may have on the general usability and security of the scheme. Given the many problems that currently exist with the widespread use of password-based authentication mechanisms, we envision a solution that is completely free of passwords, and therefore it is not reliant on the knowledge factor, but instead relies on other factors of authentication.

As pointed out by Weirich and Sasse [24] password reliant schemes have the inherent problem, that users must make a decision between convenience and security when choosing their passwords. This is problematic because users tend to choose convenience in the form of weak passwords, which results in reduced cryptographic strength of the protocol. One major benefit of designing a protocol, that does not rely on knowledge factors, is that the individual user no longer chooses a secret that directly impacts the cryptographic strength of the protocol. That being said, it does not mean that a user's behaviour can not directly or indirectly influence the security, of a knowledge-free authentication scheme in other ways. History has shown that attackers will always seek to find the weakest link when finding security vulnerabilities – so if a protocol is cryptographically sound and strong, it is most likely that an attacker will search for other ways and means to exploit the system.

4.1.2 *Transparency*

We seek to design a transparent authentication scheme, that reduces the user's direct involvement in the task of authentication, and instead performs most of the work in the background. This idea is grounded in the theory by Weirich and Sasse [24], which states that

users consider the act of authentication to be an enabling task, that should be quick and effective, to get to the primary task. Thus if it is possible to design a scheme such that the authentication process can be done securely and efficiently, while at the same time being unobtrusive to the user, it will be ideal, since it enables users to reach their primary tasks quicker and with less effort.

In previous work [7, 12, 20] the term 'transparent login' or 'transparent authentication' is used to describe authentication systems where little or no user interaction is required to perform authentication, and thus the whole process is being 'transparent' to the user because it happens in the background. The meaning of the term 'transparent', in this context, suggests that the authentication process is hidden or even invisible to the user. We consider the term 'transparent' to be rather ambiguous, since it could be interpreted as something that is easy to understand and without secrets. The Oxford English Dictionary definition of the word 'transparent' also indicate different interpretations; In a general context: *Easy to perceive or detect*, and in a computing context: *Functioning without the user being aware of its presence*.

To clarify – when we say we want to design a transparent authentication scheme, we adopt the meaning from previous work [7, 12, 20], to the extent that user interaction should be implicit, and that the authentication process happens in the background. However, we also include the other aspect of the word 'transparent' in our definition, meaning that the actions of the system must be easily perceivable for users. This concept is more precisely expressed by Bellotti and Edwards [5], who uses the term 'intelligibility' to describe a system's ability to present to its users what it knows, how it knows it, and what it will do with its knowledge. They point out the importance of user involvement and 'intelligibility' in context-aware systems, if a system's behaviour must be acceptable to its users. While our authentication system may not be classifiable as context-aware, it will take partially autonomous actions, without a user's explicit instructions, in the same manner as context-aware systems do. In summary, we consider intelligibility as an important factor, and thus define 'transparent authentication' as an authentication process that is without any user interaction, but with the user's awareness of when and where he is authenticated, even though he might not be an active participant. The user must never be in doubt of, whether or not, he is currently authenticated at some service or device – should he want to have this information.

4.1.3 *Adjustable User Awareness and Explicit Consent*

We want to design our scheme, such that different levels of security can be used, depending on the requirement of the particular use case. Specifically, it should be possible in some cases, to increase the amount of user awareness of what is happening transparently, for instance in the form of event notifications given to the user. In some

cases, the user might even be required to explicitly give his consent to authentication requests and transactions. This, of course, implies a reduction in the unobtrusiveness of the scheme from the users perspective, but can potentially provide increased certainty that no adversary is misusing the transparent mechanisms of the scheme, to gain access without the user noticing. We argue that this cost is acceptable in some situations. For instance, it is easy to imagine that users want an extra degree of security for their online banking accounts and for performing money transfers compared to viewing their news feed on a social media account. Thus our scheme should support different levels of user awareness and explicit consent, that can be adjusted based on users and service providers preferences.

Figure 1: Example of Explicit User Consent

4.1.4 *Why Wearables?*

> *A general trend is clear: wearables provide novel opportunities to improve or re- design approaches to authentication.*
>
> — Bianchi and Oakley [7]

Ojala, Keinanen, and Skytta [20] used an approach with wearable authentication in their proposed scheme. Their solution was a custom built wearable wristband with different sensors and a computer attached, which was highly specialized to be used for authentication only. Within recent years wearable computing devices (commonly referred to as 'wearables') has become available to the end-user market, intended for multi-purpose use, and not for authentication only. These recent advances have made wearables regain the attention from security researchers, to explore the different opportunities that wearables can provide for user authentication [7]. We want our authentication scheme to utilize wearables, for several reasons. First, the many sensors embedded into wearables provide an easy way to harvest input data implicitly about the wearing user. Furthermore many wearable devices come with light sensors that can report when a

user is wearing the device or have taken it off. In short, the embedded sensors can help to establish more confidence in a user's identity, which is very useful for providing secure authentication. Second, the processing power and network capabilities of wearables have grown, now enabling them to run authentication mechanisms based on strong cryptography. Third, the use of wearables allows our protocol to run on commodity hardware, that is already widely known and used, thus allowing us to focus less on hardware details of the scheme, and at the same time increasing the chance that users will adopt the system, if they already own the required hardware.

4.1.5 *Proximity*

In order to provide transparent authentication, some mechanisms must be put in place to autonomously decide when and where authentication should take place, whenever the user is not directly involved in the process. Taking inspiration from both Bardram, Kjær, and Pedersen [3] and Ojala, Keinanen, and Skytta [20] we want the authentication scheme to be proximity-based. Ideally, a user should be able to walk up to a computer and as soon as he is within a specified range, the transparent authentication process should initiate. To do this a user must carry a wearable device that is capable of providing evidence of the user's identity, to client computers where the user wants to authenticate at. We imagine that the Bluetooth protocol would be an ideal solution to support proximity-based authentication between the wearable devices and the client computers. While it may be an implementation detail, Bluetooth is to a large extend the communication standard that what we have had in mind when we designed our scheme. Additionally, the choice of a widely used communication standard as Bluetooth also aligns well with our choice of wearables, and the preference of mature commodity technologies over highly specialized solutions.

4.1.6 *Continuity*

When making the authentication mechanism more transparent (unobtrusive), a problem might be that the user is not aware that he is authenticated with the system that he is using (even if the information is available to him, he might overlook the fact), and thus walk away from an active session.

> *During our experiments we discovered that the process of logging out a user is equally important. [...] Although this is normally not considered to be part of a user authentication mechanism, we argue that logout has to be considered as a part of the design as well.*
>
> — Bardram, Kjær, and Pedersen [3]

Continuous authentication entails verification of user identity on an ongoing basis during an authentication session, thus greatly increasing the certainty that only the legitimate user is accessing pro-

tected resources [20]. A problem with many services today, is that clients are allowed to store user credentials or save access tokens without expiration in cookies, such that no confirmation of the user's identity is done after initial verification. This design is needed to reduce constant user interaction, because of the heavy reliance of password in current schemes. However this comes with the trade-off, that a non legitimate user can access private resources, in case of theft or loss of a device.

We found that continuous authentication synergises well with proximity-based, transparent and wearable authentication. Since we are using a transparent authentication scheme we do not need to store login information on the client, since the information needed to verify a user's identity, can be acquired without interrupting the user, even if it is done very frequently. After a user has established initial authentication, we can continually confirm that the basic requirements for authentication are still met, such as the user having to wear the wearable device actively, and be within proximity of the client he is authenticated at.

It is important to mention that literature uses the term 'continuous' differently. It can either describe the process of continuous verification to keep an authenticator unlocked (as in the case of the Pico [22]), or be a continuous verification process with a verifier (as in the case of Wearable Auth [20]). We aim to include *both* aspects.

4.1.7 *Authenticator and Sibling*

In order to make our scheme more resilient to theft or loss of the wearable device, and also less vulnerable in case the device is compromised, the scheme is designed to be used with an additional device, that works together with the wearable device to decide whether authentication is allowed or not. If an attacker gains access (physically or virtually), to one of the devices, but not both, it should not be possible to achieve any successful attacks. We have designed it such that the device pair, form a master/slave relationship, where outside parties communicate only with the master device to request authentication access. We generally refer to the two devices as the authenticator acting as the master and the sibling acting as the slave device. Loosely defined the authenticator could be any type of computational device. Table 2 shows the requirements for the two devices.

While we claim that our scheme design and concepts of authenticator and sibling are not locked to a specific device type, we have primarily considered and worked with the case of a smartwatch as authenticator and a smartphone as sibling. We will therefore not rule out completely, that certain design choices have been made with these two device types in mind. The main reason for this is to use devices that the user will already carry around with him. Although wearables are not so common yet, they are slowly getting traction. By using devices that the user already has in his vicinity, we increase the transparency of the solution, both in terms of the user not needing to

	Being worn by a user	*Verifying a users identity*	*Monitoring if worn*	*Receiving basic user-input*	*Displaying messages*	*Cryptographic computations*	*Bluetooth LTE I/O*
Authenticator	●	●	●	●	●	●	●
Sibling						●	●

Table 2: The requirements of devices used as authenticators and siblings.

remember bringing a dedicated authentication device, and in terms of hiding the solution in devices, he already carries around.

4.2 THE SCHEME IN PRACTICE

The design choices above, lead us to the following more brief and concrete description of how our authentication scheme should work in practice:

The scheme involves 4 different communicating parties. The authenticator, the sibling, the client and the service provider. A user carries two personal physical devices: the authenticator and the sibling, which are a smartwatch and a smartphone respectively. The authenticator acts as a key which is unique to each user and required for authentication. However the authenticator should only allow authentication if it is actively worn by the user, and in proximity of its sibling device. If at any time the user takes off the authenticator from his wrist, or if the sibling is not nearby, the authenticator should go into **locked** state, effectively meaning that all authentication requests it receives are denied. Furthermore, the authenticator should return to **unlocked** state once the sibling is again within proximity, or if the user re-equips the authenticator on his wrist and reconfirms his identity, either by using biometric sensors, pin code or some other mechanism available from the authenticator.

The user interacts with a client which is the device from where the authentication process is initiated. A client could for instance be a laptop or desktop computer, which is used to login to some online service, that requires authentication.

The client is not an active participant in the authentication process. Rather, when the user tries to authenticate with a service provider through the client, and the authenticator is nearby and **unlocked**, then the authenticator will be required to provide evidence of the identity. If the evidence is accepted by the service provider, then the client is authenticated.

The scheme is backed by a protocol that continually authenticates the user as long as the authenticator is worn in close range of the client and in **unlocked** state. If one of the conditions are not met at any time, the authentication session should terminate.

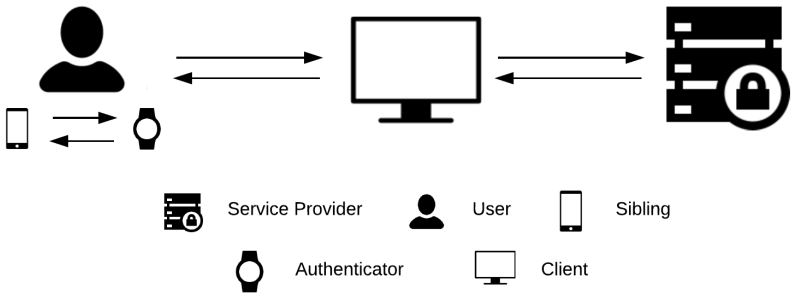


Figure 2: Component overview for the propose scheme

4.3 USER SCENARIOS

Below we describe 4 different user scenarios for our authentication scheme. The scenarios describe the most commonly occurring situations in the use of our proposed scheme. The intention of these scenarios is to gain insight into how the user interacts with the system, and to demonstrate which problems it solves for the user. The different actors appearing across the scenarios are the same as presented above in 4.2: authenticator, sibling, client and service provider. We differentiate between two levels of user involvement, where awareness is the least obtrusive of the two:

> Awareness: *The user is notified about important events, but does not require the user to actively perform any actions.*
>
> Explicit consent: *The user is required to actively perform an action, typically in the form of accepting/denying certain requests between the involved actors.*

The scenarios are listed in table 3, 4, 5, and 6.

4.4 SCHEME EVALUATION

In this section we evaluate our proposed authentication scheme, in comparison to the review in chapter 3. In addition to the benefits defined by the framework, we define four new properties, that we based on our analysis consider equally relevant for authentication. An overview is shown in table 7.

4.4.1 *Extending the Evaluation Framework*

We extend the framework [9], with 2 usability properties: *Awareness* and *No-Config*; and 2 security properties: *No-Intermediary-Knowledge* and *Continuous*.

AWARENESS: The scheme can make its users aware of its current state, and the decisions it takes autonomously. We grant *Awareness*

Scenario name	Registration with a Service Provider
Actors	authenticator A, client C, service provider S
User involvement	Awareness, optional:Explicit consent
Authenticator State	unlocked
Description	A user wants to register for a service S and uses a client C to send a registration request to S. S now communicates with the users authenticator A through C and asks for confirmation of the registration. A now notifies the user and optionally ask the user to confirm the registration request, before sending back needed information to S to complete the registration process.

Table 3: Scenario: Registration

Scenario name	Unlocking the Authenticator
Actors	authenticator A, sibling As
User involvement	Awareness, optional:Explicit consent
Authenticator State	locked $\Rightarrow$ unlocked
Description	A users watch acting as authenticator A is locked and he wishes to unlock it. The user puts on the watch, which registers that it has been put on by its owner, and starts monitoring that it remains attached. A now communicates with its sibling As, and asks for permission to unlock. After an optional user interaction with As, it communicates back with an acknowledgement, that allows A to unlock. A now notifies the user that it is unlocked. A remains unlocked for as long it is attached to the users wrist, and in range of As.

Table 4: Scenario: Unlocking

Scenario name	Establishing a Session
Actors	authenticator A, client C, service provider S
User involvement	Awareness, optional:Explicit consent
Authenticator State	$\mathtt{unlocked}$
Description	A user wants to establish a session with a service provider S through a client C. S replies with a challenge, which requires C to prove that it is a legitimate user who is requesting access. C forwards the challenge to the users nearby authenticator A, which is currently $\mathtt{unlocked}$. A optionally asks the user to confirm the authentication request, and then replies back with a response to the challenge generated by S. C now sends the challenge response back to S, and if S accepts the solution to the challenge, then an authentication sessions is established. A notifies the user that the sessions is established. If the session is not explicitly cancelled by the user, the session is maintained as long as A remains unlocked and A remains in proximity of C.

Table 5: Scenario: Authentication

Scenario name	Theft or Loss
Actors	authenticator A, sibling As
User involvement	Explicit consent
Authenticator State	$\mathtt{unlocked} \vee \mathtt{locked} \Rightarrow \mathtt{lockdown}$
Description	A user notices that either his authenticator A or sibling As is missing. The user uses the remaining device to activate $\mathtt{lockdown}$. In this state A can no longer be unlocked (even when in proximity of As), and therefore not authenticate with any services. The $\mathtt{lockdown}$ state remains until a recovery procedure has taken place, either if the missing device resurfaces, or if a new device is used for recovery.

Table 6: Scenario: Theft or loss

if a user of the scheme is capable of knowing where he is currently authenticating/authenticated at, at all times. This benefit is considered important from a usability perspective, since it helps a user to understand what happens transparently, and avoid undesirable situations as where a user think he is authenticated but actually is not, and conversely actually is authenticated, but think he is not.

NO-CONFIG: We award *No-Config* if the act of authentication or user registration does not require any additional configuration steps the first time a specific client is used by a specific user, compared to if the client was used previously by the specific user. Thus a user must be able to walk up to any arbitrary client and use the scheme for authentication or registration, with the same amount of required effort, as on a client where he has used the scheme, several times before. Note that *No-Config* does not mean that no initial configuration might be needed on a client in order to run the scheme (eg. installing a browser plugin or enabling Bluetooth). Rather it means that no additional configuration is required specifically because it is a new user of the client. Password managers are a type of scheme that typically does not offer this benefit, since a user will have to undergo an extra step the first time he uses a new client in order to transfer his keychain onto it. The benefit of *No-Config* is considered a usability benefit, because it reduces the time a user has to spend on authentication the first time he uses a new client. It is especially important in environments where many public computers are used, and a user does not have a personal computer.

NO-INTERMEDIARY-KNOWLEDGE: During authentication or registration the client is not exposed to any information with long-term value. We grant *No-Intermediary-Knowledge* if no secrets (such as passwords), that could be used to authenticate at a later point in time, is given or inputted on the client. *No-Intermediary-Knowledge* is a security benefit and is considered important because it limits the power of an adversary in full or partial control of the client.

CONTINUOUS: The scheme supports continuous authentication. We grant the benefit if a scheme reconfirms a user's identity with a verifier regularly after initial authentication, and is capable of stopping the authentication session whenever the user's identity can not be verified any longer. This benefit is a security benefit, and limits the power of an adversary with physical or remote access to a client greatly, since the authentication sessions are not kept alive for an extended duration after the user is no longer actively engaged.

We grant *Quasi-Continuous* if the system is using a hardware token and only the token is continuously authenticated, but sessions with services are not.

4.4.2 *Rating Process*

The authentication scheme proposed in this thesis is intended as an improvement over password. It aims to improve security and usability of passwords, but does not aim to surpass passwords in terms of deployability.

The scheme is *Memorywise-Effortless* by design because it is a transparent authentication scheme, and therefore does not require the user to input anything from memory or perform some memorized interaction in order to authenticate. For the same reasons and since no new hardware has to be introduced to the user, we award *Easy-to-learn*. The proximity-based login approach of the scheme ensures that it is *Physically-Effortless* to use, as the only real physical requirements of the user to use the scheme for authentication, is that the user must wear a watch and carry a phone, and move within close range of the client where authentication is performed at.

We grant *Efficient-to-Use* and *Infrequent-Errors* because the scheme is designed to have minor user interaction, and the cryptographic exchanges between service provider and authenticator, is carried out much faster than the time taken for typing even a simple password. Furthermore, the scheme does not rely on biometrics or other methods of authentication that could introduce false-negatives. The scheme is *Scalable-for-Users*, as an increased number of links between users and service providers does not have any impact on the individual users experience. We award the scheme *Quasi-Nothing-to-Carry*, because the benefit can only be granted if the user has to carry nothing at all. However, if the devices are considered something which the user always carry around anyway, in this case, a phone and a watch, the applied burden on the user is arguably reduced significantly. We do not grant the scheme *Easy-Recovery-from-Loss* since no such mechanism exists for the scheme in its current design state. We grant *Awareness* as the scheme is designed to send notifications to a user on his watch, whenever a new registration or authentication session is started. The scheme does not offer the benefit of *No-Config* as a pairing process is required between client and authenticator, the first time a new client is used.

The scheme is *Accessible* since no physically constraining activities are involved in the use of the scheme. We grant it *Quasi-Negligible-Cost-per-User*, as the required devices: smartphone and smartwatch, are relatively costly on one hand, but on the other hand is something that can be used for many other purposes and is already owned by many users. As the scheme aims to be a complete password replacement, it is not designed to prioritize *Server-Compatible* or *Browser-Compatible*. We can therefore not award either benefit, as a whole new infrastructure is most likely needed by the service providers and verifies end. The scheme is *Non-Proprietary*, but it is not *Mature*.

A strong cryptographic protocol is used for the scheme, which in combination with the transparent proximity-based login mechanism ensures full ratings on most of the security benefits. We grant *Resilient-to-Physical-Observation*, *Resilient-to-Targeted-Impersonation*, simply be-

		Usability										Deployability						Security														
	Reference	*Memorywise-Effortless*	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Awareness*	*No-Config*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*	*No-Intermediary-Knowledge*	*Continuous*
Passwords				●		●	●	○	●		●	●	●	●	●	●	●		○						●	●	●	●		
Context-Aware Auth	[3]	○	●		○	●	●	●			●	●					●	○	●			○		○	○		●		○	○
Wearable Auth	[20]	●	●	○	○	●	●	●				●					●	●	●	●	●	?	?	●	●	?	○	?	?	●
Pico	[22]	●	●		●		○	○									●	●	●	●	●	●	●	●	○	●	●	●	●	○
Our design		●	●	○	●	●	●	●		●		●	○				●	●	●	●	●	○	●	●	○	●	○	●	●	●

● = offers the benefit ○ = almost offers the benefit ? = not known

Table 7: Comparing the benefits of passwords with related work in Pervasive Authentication and the (envisioned) benefits of our design.

cause the scheme never requires users to reveal any information, in order to authenticate. The protocol used for the scheme requires a new cryptographic key pair to be used for each association between verifier and user, and thus offers the benefits *Resilient-to-Leaks-from-Other-Verifiers* and *Unlinkable*. The scheme also has *No-Trusted-Third-Party*. We award *Resilient-to-Throttled-Guessing* and *Resilient-to-Unthrottled-Guessing* as the protocol is based on El-Gamal encryption with a sufficiently large key space, to be practically impossible to guess even with large amounts of computing power. The protocol uses a challenge-response architecture, meaning that the only way to authenticate is to respond to a challenge that can only be solved by possessing the right secret key. The protocol never transmits or reveals any secrets keys, and thus no information can be acquired that can be used at a later point of time to authenticate. We therefore grant both *No-Intermediary-Knowledge* and *Resilient-to-Phishing*. The scheme aims for the authenticator to continuously verify the presence of the user wearing the device, and to terminate any active sessions if the devices lock. Thus we grant the benefit *Continuous*.

While the scheme is designed to offer many of the security benefits, it has a few weak points where we can only award 'quasi' rating. The scheme is *Quasi-Resilient-to-Theft* as it is not enough to steal the watch without the phone and vice versa. Furthermore, the watch can be pin-code protected if it gets stolen, although it is a rather weak form of protection. Since the scheme relies on smartphones and smartwatches, and not a dedicated hardware device, we must assume that it is a possibility that malware exists on either device which can potentially be harmful to the execution of the scheme. However it is only harmful

in case both devices are malware infected, and thus we grant *Quasi-Resilient-to-Internal-Observation*.

The scheme can by design not offer the benefit *Requiring-Explicit-Consent* fully, as parts of the authentication process are done transparently. However, the scheme defines an option for adding explicit user consent in certain situations, by prompting users for confirmation of new authentication sessions and registrations.

4.4.3 *Extending the Review*

To compare the previously considered schemes, we extend their review to include the new benefits.

None of the previously considered mechanisms are granted *Awareness*. Passwords, Context-Aware Auth and Pico are granted *No-Config*. Passwords obviously gain this benefit as using the scheme (assuming that user always has to log in when using a service) is the same for all clients. Similarly, for Context-Aware Auth, the login process is the same for all supporting clients. Pico uses a two-channel communication approach. One channel is a visual channel. When the user wishes to authenticate, the Pico is used to scan a QR-code on the screen. The Pico then opens a communication channel to the service and authenticates the user. Therefore no configuration is needed when using a new client.

We grant *No-Intermediary-Knowledge* to Pico as the utilized client is not given any pertinent information except for a temporary authentication token. Context-Aware auth is granted *Quasi-No-Intermediary-Knowledge*, as the user might occasionally input his password on the client.

Wearable Auth is granted *Continues*, as the wearable locks if not actively worn. Furthermore, the wearable must be present near the client or the client will lock. Pico is granted *Quasi-Continuous* as the Pico continuously verifies the presence of its siblings, but does (seemingly) not terminate active sessions with services if locked. Context-Aware Auth is granted *Quasi-Continuous* as a user can walk away from an active session if he forgets his keycard in the machine. The system is not described to terminate the session if the context service no longer registers the user at the client.

5

PROTOCOL DESIGN

In the previous chapter, we present a design for a CTA authentication scheme. In this chapter we will present a protocol, which is simply a sequence of well-chosen messages, that can support this scheme. In this chapter, we will only highlight a few important properties from the previous chapter in regards to certain choices, and will then later evaluate which properties the proposed protocol achieves.

5.1 DEFINITION OF AUTHENTICATION

So far we have not clearly defined what 'authentication' actually entails. In his well-renowned article "A Hierarchy of Authentication Specifications" Lowe [16] puts forward the following definition:

Definition 5.1 (Injective Agreement)**.** *We say that a protocol guarantees to an initiator* A *agreement with a responder* B *on a set of data items* ds *if, whenever* A *(acting as initiator) 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 variables in* ds*, and each such run of* A *corresponds to a unique run of* B*.*

This definition infer that if two parties agree on a set of data (or evidence of identity), and if there is a one-to-one relationship between the protocol run at the responder and verifier, then they achieve (injective) agreement, and thus the responder authenticates with the verifier.

5.2 DESIGN RATIONALE

Our design inspires from this definition, and effectively, the service provider S, will on request from the client C, issue a fresh challenge. If the client can reply with a proper response to the challenge then it should be authenticated for a given period of time. Let us consider a simple example in which the service provider encrypts a random nonce with the public-key of the client, and sends the cipher to the client. The client then decrypts the cipher and responds to the challenge with the recovered nonce.

$$S \to C : enc(n, pk_c) \qquad \text{(message 1)}$$
$$C \to S : n \qquad \text{(message 2)}$$

Only an actor in possession of sk_c would be able to recover the correct answer, and thus presenting n, serves as evidence of knowing sk_c. This can be used to authenticate if the server links the public key

to a user-identity. This will serve as a starting point for this protocol. However, a few design features from the previous chapter needs to be considered:

- The client should ask the authenticator for 'permission' to authenticate.
- The authenticator should only give permission if unlocked; thus in range of the sibling.
- The client should not be exposed to any piece of information that could be used in a later successful round of authentication.
- Hijacking an active session or token should not give an adversary unauthorized access for longer than the session is actively kept alive by the service provider and authenticator.

In the design, we mention that the authenticator can be in a state of either unlocked, locked or lockdown. In practice, we take a different approach to implementing these states. As we envision using general-purpose commodity platforms such as e.g. iOS and Android devices, compromised devices are a concern that should be inherently designed for, and solely trusting a single device to uphold these states is therefore not an option.

5.2.1 *Distributed Authentication*

To ensure that authenticator and sibling are always involved in an authentication run, the unlocked state will in practice be implemented by forcing the authenticator and sibling to collaborate, in computing the response to an authentication challenge. If communication between authenticator, sibling and client are forced onto a local channel (such as Bluetooth), then authentication is only possible when all three devices are in each others proximity.

In comparison, Pico [22] functions by having a main device, the Pico, that is the users key-store. The key-store is encrypted with a k-*out-of*-n threshold encryption system. This means that at least k siblings (small wearable tokens) must be in its vicinity for it to unlock. On request from the Pico, the siblings send their key-share, allowing the Pico to unlock its key-store. An assumption is made on the Pico, that it periodically 'forgets' the key, thus forcing it to continuously interact with its siblings.

We see this as a problem because if the Pico is compromised in the unlocked state, then *all* of the user's services are compromised. In the original paper, it is assumed that an adversary is not capable of compromising the Pico in the unlocked state. This is an unrealistic assumption when using general-purpose devices.

The rationale behind having a central unit holding all keys, and having peripheral devices unlocking, instead of having all devices actively participate, was taken with consideration to the technical limitations of wearables [23]. However, since the Pico was proposed in 2011,

these limitations are diminishing, and modern wearables are now fully capable of making cryptographic calculations without draining their battery.

The collaboration between authenticator and sibling can be achieved with 'secure multiparty computations' (MPC). MPC entails a group of agents jointly computing a function, such as decrypting a cipher, without revealing anything about their individual input to the function. Furthermore, only with full participation from all actors will the output of the function be meaningful. This means that both devices would have to be compromised, for an adversary to be able to obtain the secrets needed to compromise the user's services. In practice, MPC can be implemented by utilizing a partial crypto system.

5.3 PARTIAL CRYPTO SYSTEMS

A partial crypto systems is a system in which multiple actors have to collaborate to encrypt and decrypt messages. Such systems are useful because they ensure that even if one of the actors is compromised, then the system is not compromised. Many such systems exists, but in this section we will present Distributed ElGamal, which is a partial crypto system [11].

ELGAMAL CRYPTO SYSTEM ElGamal is a probabilistic and homomorphic public-key crypto system based on the Diffie-Hellman assumption. ElGamal is secure against Chosen-plaintext attacks (CPA), if the Decisional Diffie–Hellman (DDH) problem is hard [15, page 400].

Using the cyclic prime order groups, as defined in section 2.3.1, we can define ElGamal in the following way:

- **Gen:** on input 1^n run $\mathcal{G}(1^n)$ to obtain a cyclic group $\langle \mathbb{G}, q, g \rangle$. Then choose a uniform $x \in \mathbb{Z}_q$ and compute $y := g^x$. Then output the public-key $\langle \mathbb{G}, q, g, y \rangle$ and the private-key $\langle \mathbb{G}, q, g, x \rangle$
- **Enc:** on input of a public-key and a message $m \in \mathbb{G}$, choose a uniform $r \in \mathbb{Z}_q$ and output the ciphertext $c := (my^r, g^r)$.
- **Dec:** on input of a private-key and a ciphertext $c = (\alpha, \beta)$, output the message $m := \alpha/\beta^x$.

DISTRIBUTED ELGAMAL The distributed variant of ElGamal leverages that the original crypto system is homomorphic. Although the system is homomorphic over both message and keys for both encryption and descryption, the following is focused on homomorphism over the keys for decryption[1].

$$\mathrm{Dec}(c, sk_1) \times \mathrm{Dec}(c, sk_2) = \mathrm{Dec}(c, sk_1 + sk_2)$$

1 For some set of operators '$+$' and '$\times$'

Let each participating actor i in the distributed system generate an ElGamal key-pair (pk_i, sk_i), using the same cyclic group $\langle \mathbb{G}, q, g \rangle$, by choosing an uniform $x_i \in \mathbb{Z}_q$ and calculating $y_i = g^{x_i}$ [11]. The joint public and private-key is now given as:

$$y = \prod_{i=1}^{n} y_i \qquad\qquad x = \sum_{i=1}^{n} x_i$$

An encrypted message $Enc(m, pk) \to (\alpha, \beta)$ can be jointly decrypted by each participant calculating β^{x_i}. The message can then be recovered as:

$$m := \frac{\alpha}{\prod_{i=1}^{n} \beta^{x_i}} = \frac{\alpha}{\beta^x}$$

The advantage of this is that the computation and sharing of β^{x_i}, following the DDH assumption, does not leak any information about the private-keys. Neither does the shares leak any information about the encrypted message before all shares are combined. We define the new operations as:

- **Gen':** on input of a cyclic group $\langle \mathbb{G}, q, g \rangle$, choose a uniform $x \in \mathbb{Z}_q$ and calculate $y = g^x$. Then output the public-key $\langle \mathbb{G}, q, g, y \rangle$ and the private-key $\langle \mathbb{G}, q, g, x \rangle$
- **Dec':** on input of a private-key and a ciphertext $c = (\alpha, \beta)$, output the partial decryption $c' := (\alpha, \beta^{x_i})$
- **Combine:** on input of two partially decrypted ciphertexts $c'_1 = (\alpha, \beta^{x_1})$ and $c'_2 = (\alpha, \beta^{x_2})$, output

  notice that the α's must match

  $$c' := (\alpha, \beta^{x_1} \cdot \beta^{x_2}) = \left(\alpha, \beta^{x_1 + x_2}\right)$$

- **Recover:** on input of a partially decrypted ciphertext $c' = (\alpha, \beta^x)$, output the message $m := \alpha / \beta^x$

We denote the recovery of a message from combining partially decrypted ciphers as $m := Dec'(\cdot) \times Dec'(\cdot)$. Furthermore, we denote the product of public-keys as $pk := pk_1 \times pk_2$.

5.4 THE PROTOCOL

Two steps of the protocol has to be defined. A registration and authentication step. The purpose of the registration step is to establish a set of shared knowledge. The purpose of the authentication step is to provide the service provider with evidence, and for the service provider to be able to verify the authenticity of the evidence based on the knowledge acquired during the registration. The proposed protocol builds on the Distributed ElGamal crypto system as presented in the previous section.

5.4.1 *Registration*

The registration is initiated by the client (and thus the end-user) by sending a message to the authenticator with a universally unique user-id (such as a uuid[2]). The client might have asked the service provider in advance to issue this id based on some data such as a username. The authenticator then forwards the message to the sibling and starts computing a new Distributed ElGamal key-pair. The sibling also computes a new Distributed ElGamal key-pair and sends its public-key to the authenticator. The authenticator now combines the keys to a joint public-key and sends it back to the client. Lastly the client sends the new public-key to the server along with the user-id. This is shown in figure 3.

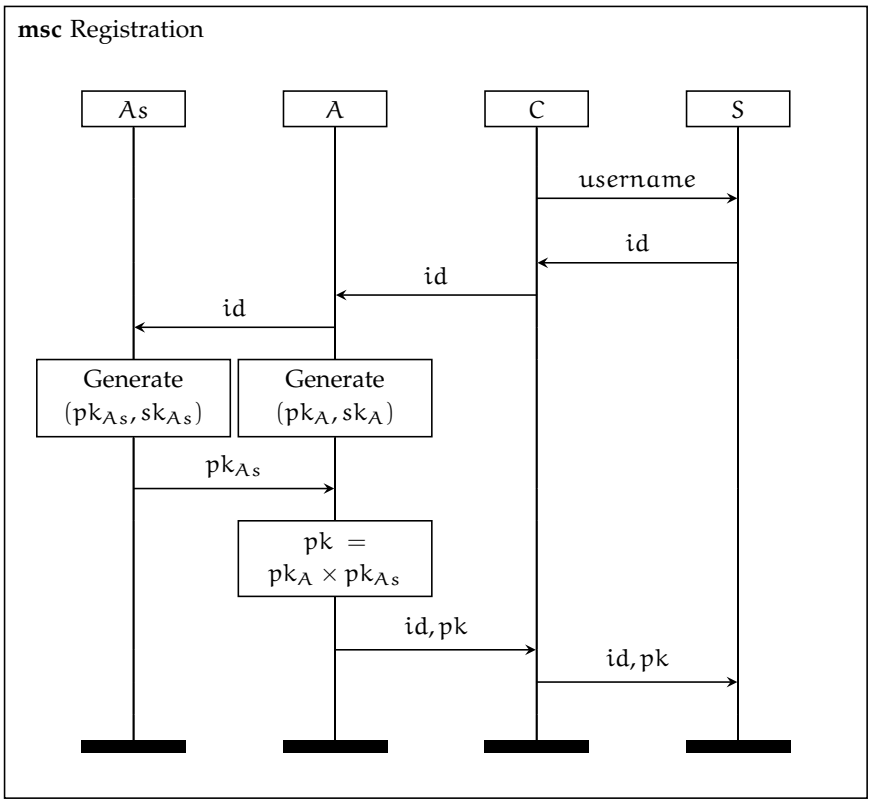


Figure 3: The sequence of messages involved in a successful registration.

5.4.2 *Authentication*

Authentication is initiated by the client by sending an authentication request with a user-id to the service provider. The service provider responds with a challenge $c \leftarrow enc(n, pk)$, where pk is the public-key corresponding to the user-id, and n is an arbitrary nonce in G. After sending the challenge the service provider starts a timer T. The challenge is forwarded to the authenticator and sibling which both partly decrypts the challenge. The sibling sends its partial decryption to the authenticator which combines and recovers the nonce and sends it

2 See https://en.wikipedia.org/wiki/Universally_unique_identifier

back to the service provider. If the received nonce is correct then the service provider issues a token to the client, valid for a given duration (in regards to the timer T). Before the token expires, the process is repeated to continuously keep the session active. This is shown in figure 4.

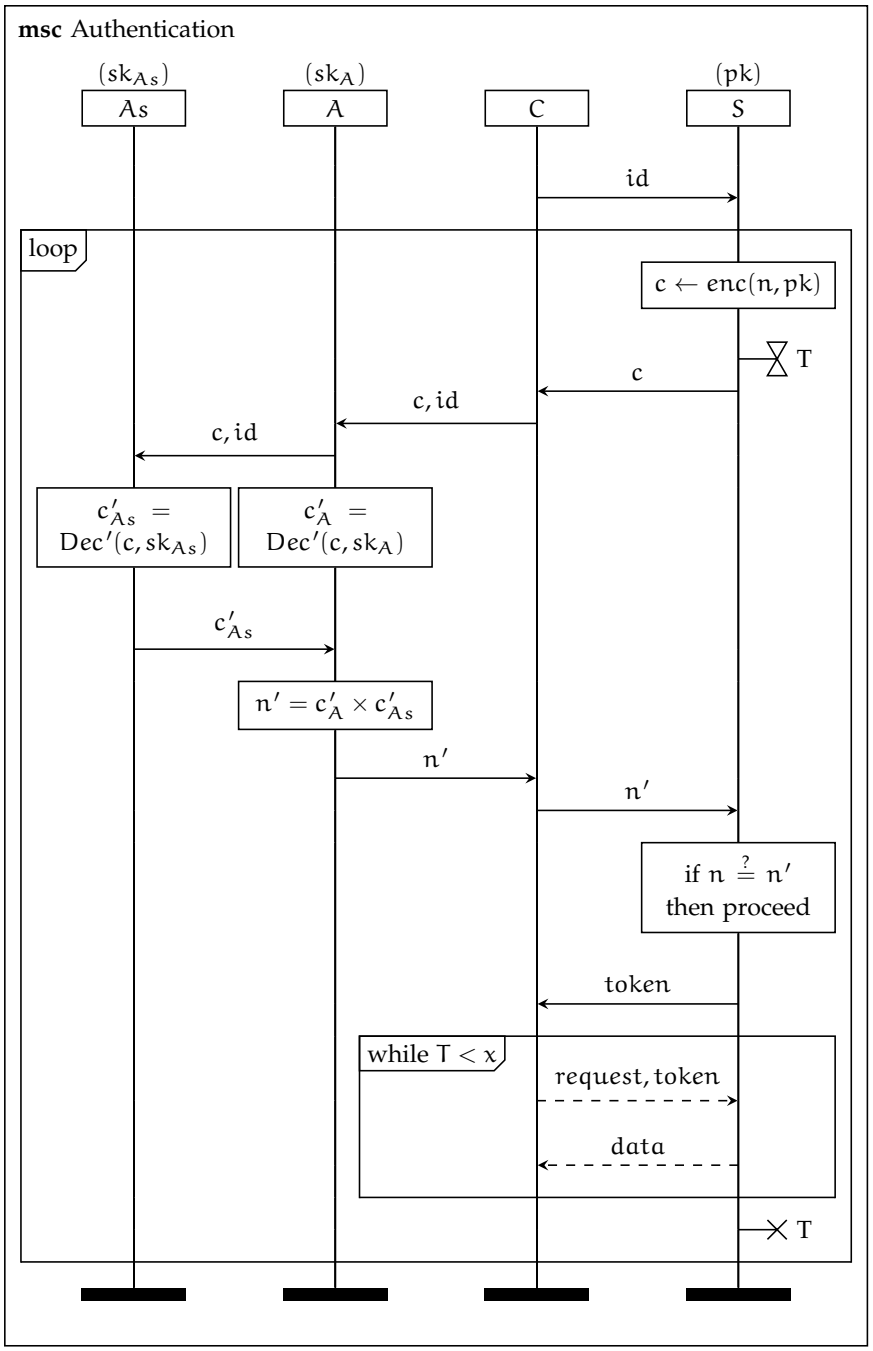


Figure 4: The sequence of messages involved in a successful authentication.

6 SECURITY ANALYSIS

This chapter presents a security analysis for the protocol proposed in the previous chapter. This chapter will prove that our authentication protocol achieves injective agreement (IA). In protocol verification, two models are commonly used for proving properties [8]:

- In the computational model, we consider passive probabilistic polynomial-time adversaries limited to certain observed information. In the computational model, proofs are often manual, and we say that a given security property holds, if the probability that it does not, is negligible in a security parameter.
- In the symbolic model (or Dolev-Yao model [13]), we consider active adaptive adversaries that can observe, intercept and synthesize new messages. Cryptographic primitives are represented as function symbols, and messages as composite terms of these symbols. Proofs are often automated essentially by generating the set of all information an adversary could possibly know, and security properties are proven under the assumption that the primitives are unbreakable and non-malleable.

An attack found in the symbolic model will also yield an attack in the computational model. However, an attack in the computational model might not be found in the symbolic model [8].

We will use the computation model for crypto analysis to prove against attacks that break the protocol by breaking the underlying cryptographic primitives, and the symbolic model for logical analysis to prove against attacks such as man-in-the-middle, replay and reflection attacks.

6.1 SECURITY IN THE COMPUTATIONAL MODEL

A prerequisite of injective agreement is that evidence of identity is unforgeable.

Definition 6.1 (Unforgeability)**.** *For all challenges, without knowing the pertinent secrets, an adversary $\mathcal{A}$ should not be able to forge a response that the service provider will accept as valid.*

This property is strongly related to the underlying cryptographic primitives, and we have therefore chosen to prove unforgeability in a computational model. The above definition is moreover very strong, and is not in a provable form for the computational model. We will therefore in the following section state very precise definitions of when we consider unforgeability to be broken, and prove unforgeability for different levels of observable information.

6.1.1 *Weak External Observation*

Let us start by defining a *simulator* of our protocol.

Definition 6.2 (Weak external observation)**.** *Let* $\mathtt{Sim}_{\mathcal{C}}^{weo}(\cdot)$ *denote a simulator of our protocol that outputs challenge and response pairs* (c, n)*.*

The purpose of this definition is to be able to model the view of an adversary $\mathcal{A}$ observing, or eavesdropping, rounds of authentication. This particular definition corresponds to the view of an adversary capable of eavesdropping the communication between a client and a service provider.

Definition 6.3 (Registration)**.** *Let* $\mathtt{Register}(\cdot)$ *denote an algorithm that on input of a security parameter* 1^n *runs some generate primitive and outputs a public-key* $\mathtt{pk}$*, and a set of corresponding private-keys* $\mathtt{sk}$ *with* $|\mathtt{sk}| = 2$*.*

In this section we consider only the authentication part of our protocol, and therefore model registration as an atomic operation. In practice that is not the case. This we be revisited in the discussion.

With these definitions in place, given an adversary $\mathcal{A}$, consider the following experiment:

Unforgeability experiment $\mathtt{Forge}_{\mathcal{A},\mathcal{C}}^{weo}(n)$**:**

1. $\mathtt{Register}(1^n)$ is run to obtain keys $(\mathtt{pk}, \mathtt{sk})$.
2. The adversary $\mathcal{A}$ is given the public-key $\mathtt{pk}$, and access to a simulator $\mathtt{Sim}_{\mathcal{C}}^{weo}(\cdot)$.
3. The adversary is eventually given $c \leftarrow \mathtt{enc}(n, \mathtt{pk})$, with n being an arbitrary value of $\mathbb{G}$, and outputs n'.
4. $\mathcal{A}$ succeeds if and only if $n = n'$. In this case the output of the experiment is defined to be 1.

Proposition 6.4. *Our authentication protocol* $\mathcal{C}$ *has unforgeability if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *capable of weak external observation, there is a negligible function* $\mathtt{negl}$ *such that*

$$\Pr\left[\mathtt{Forge}_{\mathcal{A},\mathcal{C}}^{weo}(n) = 1\right] \leqslant \mathtt{negl}(n)$$

Proof. An adversary $\mathcal{A}$ for $\mathtt{Forge}_{\mathcal{A},\mathcal{C}}^{weo}(n)$ can be used to construct an adversary $\mathcal{A}'$ for the eavesdropping indistingushability experiment $\mathtt{PubK}_{\mathcal{A}',\Pi}^{eav}(n)$, where Π is standard ElGamal as defined in the previous chapter. The adversary can be constructed in the following way:

Adversary $\mathcal{A}'$**:**

1. On input of an ElGamal public-key $\mathtt{pk} := \langle \mathbb{G}, q, g, y \rangle$, call $\mathcal{A}$ with $\mathtt{pk}$ and a simulator $\mathtt{Sim}_{\mathcal{C}}^{weo}(\cdot)$, and output two arbitrary messages $m_1, m_2 \in \mathbb{G}$.
2. On queries to the simulator, select an arbitrary $n \in \mathbb{G}$, compute $c \leftarrow \mathtt{enc}(n, \mathtt{pk})$, and return (c, n).

3. On input of a ciphertext $c \leftarrow enc(m_b, pk)$, call $\mathcal{A}(c)$.
4. If $\mathcal{A}$ outputs m_0, then output 0, if it outputs m_1, then output 1, otherwise choose arbitrarily.

For a public-key encryption scheme to be indistinguishable in the presence of an eavesdropper, then the probability must be:

$$\Pr\left[\mathrm{PubK}^{eav}_{\mathcal{A}',\Pi}(n) = 1\right] \leqslant \frac{1}{2} + negl(n)$$

However, if an adversary $\mathcal{A}$ exists that succeeds in $\mathrm{Forge}^{weo}_{\mathcal{A},\Pi}(n)$ with probability greater than $negl(n)$, then the probability of the adversary $\mathcal{A}'$ succeeding in $\mathrm{PubK}^{eav}_{\mathcal{A}',\Pi}(n)$ would be:

$$\Pr\left[\mathrm{PubK}^{eav}_{\mathcal{A}',\Pi}(n) = 1\right] = \frac{1}{2} + \Pr\left[\mathrm{Forge}^{weo}_{\mathcal{A},\mathcal{C}}(n) = 1\right] \nleqslant \frac{1}{2} + negl(n)$$

Since ElGamal is proven to be indistinguishable in the presence of an eavesdropper (CPA-secure) [15, page 402], under the assumption that the DDH problem is hard, then by reduction, proposition 6.4 also holds under the DDH assumption. ■

This proof shows that for all PPT adversaries capable of weak external observation; meaning adversaries capable of eavesdropping on the communication between client and service provider; they will not be able to forge a valid response to a challenge.

6.1.2 *Weak Internal Observation*

Lets now consider a slightly different experiment, where adversaries are capable of weak internal observation. This means that they are *also* capable of obtaining private-keys.

Definition 6.5. *Let* $\mathrm{LtkReveal}(\cdot)$ *denote a private-key oracle that can be queried for private-keys in the set* sk.

With this definition the unforgeability experiment for adversaries capable of *Weak Internal Observation* is given as:

Unforgeability experiment $\mathrm{Forge}^{wio}_{\mathcal{A},\mathcal{C}}(n)$**:**

1. $\mathrm{Register}(1^n)$ is run to obtain keys (pk, sk).
2. The adversary $\mathcal{A}$ is given the public-key pk, access to a simulator $\mathrm{Sim}^{weo}_{\mathcal{C}}(\cdot)$, and access to a key oracle $\mathrm{LtkReveal}(\cdot)$. Let $\mathcal{K}$ denote the set of queries to the key oracle.
3. The adversary is eventually given $c \leftarrow enc(n, pk)$, with n being an arbitrary value of $\mathbb{G}$, and outputs n'.
4. $\mathcal{A}$ succeeds if and only if (1) $n = n'$, and (2) $sk \notin \mathcal{K}$. In this case the output of the experiment is defined to be 1.

This experiment entails, that even in possession of some proper subset of the private-keys, then the adversary is not capable of forging a valid response.

Proposition 6.6. *Our authentication protocol $\mathcal{C}$ has unforgeability if for all probabilistic polynomial-time adversaries $\mathcal{A}$ capable of weak internal observation, there is a negligible function* $\mathtt{negl}$ *such that*

$$\Pr\left[\mathsf{Forge}^{wio}_{\mathcal{A},\mathcal{C}}(n) = 1\right] \leqslant \mathtt{negl}(n)$$

It should be clear from the experiment and proposition that if Distributed ElGamal is CPA-secure, even when a key is leaked, then the above proposition holds. It follows that for Distributed ElGamal, no party (neither an honest party) with a proper subset of keys should be able to distinguish ciphers, otherwise it would not be a functioning partial encryption system.

Although a formal proof is omitted for brevity, for a Distributed ElGamal cipher (α, β), consider the alpha:

$$\alpha = my^r = m\left(g^{x_1+x_2}\right)^r = m \cdot g^{rx_1} \cdot g^{rx_2}$$

If the adversary knows x_1, then the cipher can be reduced as follows:

$$\frac{m \cdot g^{rx_1} \cdot g^{rx_2}}{\beta^{x_1}} = m \cdot g^{rx_2}$$

The reduction leaves us with exactly the cipher of the encryption $\mathsf{Enc}(m, y_2) \to (m \cdot g^{rx_2}, g^r)$. Intuitively, if an adversary can recover m knowing x_1 for a Distributed ElGamal cipher, then the adversary can also break standard ElGamal. □

6.1.3 *Strong External Observation*

In this section, we extend the adversaries capabilities to include strong external observation; that is, the capability of listening to *all* communication between actors in the protocol (as shown in figure 4). For generality, we do not fix the actor who publishes its partial decryption, but just note the publishing party as i.

Definition 6.7 (strong external observation)**.** *Let $\mathsf{Sim}^{seo}_{\mathcal{C}}(\cdot)$ denote a simulator of our protocol that outputs challenge, partial decryption, and response tuples $\langle c, c'_i, n\rangle$.*

We define a new experiment $\mathsf{Forge}^{seo}_{\mathcal{A},\mathcal{C}}$, to be the same as $\mathsf{Forge}^{weo}_{\mathcal{A},\mathcal{C}}$, except that the simulator used is now $\mathsf{Sim}^{seo}_{\mathcal{C}}(\cdot)$, and the adversary is also given $c'_i = \mathsf{Dec}'(c, sk_i)$ with $sk_i \in sk$ in step 3.

Proposition 6.8. *Our authentication protocol $\mathcal{C}$ has unforgeability if for all probabilistic polynomial-time adversaries $\mathcal{A}$ capable of strong external observation, there is a negligible function* $\mathtt{negl}$ *such that*

$$\Pr\left[\mathsf{Forge}^{seo}_{\mathcal{A},\mathcal{C}}(n) = 1\right] \leqslant \mathtt{negl}(n)$$

Collary 6.9. *If proposition 6.6 holds, then proposition 6.8 also holds.*

Proof. An adversary $\mathcal{A}$ for $\mathtt{Forge}^{seo}_{\mathcal{A},\mathcal{C}}(n)$, can be used to construct an adversary $\mathcal{A}'$ for $\mathtt{Forge}^{wio}_{\mathcal{A}',\mathcal{C}}(n)$ in the following way:

Adversary $\mathcal{A}'$:

1. On input of a public-key pk, a simulator $\mathtt{Sim}^{weo}_{\mathcal{C}}(\cdot)$, and a key oracle $\mathtt{LtkReveal}(\cdot)$, call $\mathcal{A}$ with pk and a new simulator $\mathtt{Sim}^{seo}_{\mathcal{C}}(\cdot)$.
2. Query the key oracle for some private-key $sk_i := \mathtt{LtkReveal}(i)$.
3. On queries to the simulator $\mathtt{Sim}^{seo}_{\mathcal{C}}(\cdot)$, query $\mathtt{Sim}^{weo}_{\mathcal{C}}(\cdot)$ for (c, n). Then compute $c'_i := \mathtt{Dec}'(c, sk_i)$ and output (c, c'_i, n).
4. On input of a ciphertext c, compute $c'_i = \mathtt{Dec}'(c, sk_i)$, and then output $\mathcal{A}(c, c'_i)$.

In all cases where the adversary $\mathcal{A}$ succeeds, $\mathcal{A}'$ also succeeds. If there exists no PPT adversary $\mathcal{A}'$ that succeeds with better than negligible probability, then there can exist no PPT adversary $\mathcal{A}$ that succeeds with better than negligible probability

$$\Pr\left[\mathtt{Forge}^{seo}_{\mathcal{A},\mathcal{C}}(n) = 1\right] \leqslant \Pr\left[\mathtt{Forge}^{wio}_{\mathcal{A}',\mathcal{C}}(n) = 1\right] \leqslant \mathtt{negl}(n)$$

■

6.1.4 *Strong Internal Observation*

Let us consider a last unforgeability experiment. We define $\mathtt{Forge}^{sio}_{\mathcal{A},\mathcal{C}}$, to be the same as $\mathtt{Forge}^{wio}_{\mathcal{A},\mathcal{C}}$, except that the simulator used is now $\mathtt{sim}^{seo}_{\mathcal{C}}(\cdot)$, and the adversary is also given $c'_i = \mathtt{Dec}'(c, sk_i)$ with $sk_i \in sk$ in step 3.

Proposition 6.10. *Our authentication protocol $\mathcal{C}$ has unforgeability if for all probabilistic polynomial-time adversaries $\mathcal{A}$ capable of strong internal observation, there is a negligible function* $\mathtt{negl}$ *such that*

$$\Pr\left[\mathtt{Forge}^{sio}_{\mathcal{A},\mathcal{C}}(n) = 1\right] \leqslant \mathtt{negl}(n)$$

This proposition can be trivially disproved and lead us to the first known attack on our protocol. As the adversary is given given both $c \leftarrow \mathtt{enc}(n, pk)$ and $c' = \mathtt{Dec}'(c, sk_i)$ in step 3, and has access to $\mathtt{LtkReveal}(\cdot)$, he can simply retrieve the missing key, and then recover n thus succeeding *every* time.

If an adversary obtains the key of the authenticator and can communicate with the sibling, then the attack can be mounted in practice. This is a problem since *Resilient-to-Internal-Observation* (as presented in the design section), is one of our design goals, and is not granted if compromising one of two devices can break the scheme [9].

6.2 SECURITY IN THE SYMBOLIC MODEL

In this section, we consider adversaries capable of not only internal observation and compromising keys, but also capable of adaptively

intercept and synthesize messages. We have already shown that our protocol does not have unforgeability under strong internal observation. Here we however show that if messages between authenticator and sibling is authentic, then our protocol has both unforgeability under internal observation and achieves injective agreement. Furthermore, we show that unless the adversary obtains both distributed keys, then either authenticator or sibling was involved in the authentication run.

6.2.1 *The Tamarin Prover*

Tamarin is a symbolic verification tool specialized for security protocols [17]. Tamarin is designed to prove trace properties. A trace property holds if for all possible instances of the model, a given trace does or does not exist, and can therefore be used to prove properties such as secrecy (where the adversary in no possible instances of the model can obtain a secret value) and authentication (where for all traces some set of events occurred).

Rules in tamarin are written as $l \; -[\; a \;]\!\!\rightarrow \; r$, where l is a set of inputs, a is a set of events, and r is a set of outputs. If a is empty then the brackets are omitted. The predefined predicates In and Out are used to pass information between rules. Furthermore $\mathrm{Fr}(\sim m)$ is a predicate that outputs a new fresh message m, and a message taken as input can be restricted to be fresh by prepending $\sim$. Messages prepended with \$ symbolizes public terms and is commonly used to establish actors identity. Lastly, custom in- and outputs can be defined by creating new predicates, such as $Server(\$S, \sim n)$. A rule with this predicate in its output set, outputs the identity $\$S$ and a fresh value $\sim n$ for other rules to consume. As an actor's behavior is typically split into several rules, these predicates can be used to persist state or pass data between rules, that should not be directly available to the adversary. If a predicate is prepended with !, then the output is persisted, where it is otherwise consumed when retrieved.

Functions can be applied to messages or terms. For example hashing a message m by applying the function $h/1$, yields a new term $h(m)$. The term is considered as an atomic unit, and inner values can only be retrieved if allowed by a rule $\mathrm{In}(h(m)) \rightarrow \mathrm{Out}(m)$, or equation $h(m) \simeq m$.

6.2.2 *Modeling our Protocol in Tamarin*

In this section we are modeling on a higher abstraction level than in the previous section. In the previous section, we proved certain properties based on the probability of breaking the underlying cryptographic primitives. This section, will utilize symbolic verification and the underlying primitives are therefore assumed to be unbreakable. The following presents the model. The full code version is listed in appendix D.

6.2.3 *Equational Theory*

The primitives are modeled as a set of functions; enc, dec, $pdec$, $plus$, $comb$ and pk, and an equational theory.

$$dec(enc(m, pk(sk)), sk) \simeq m$$
$$comb(pk(\sim sk1), pk(\sim sk2)) \simeq pk(plus(\sim sk1, \sim sk2))$$
$$comb(pdec(c, sk1), pdec(c, sk2)) \simeq dec(c, plus(sk1, sk2))$$

The enc, dec and pk functions are defined to model standard asymmetric encryption, where the public-key pk of a secret-key sk is given as $pk = pk(sk)$. The encrypt function $enc(m, pk)$ takes a message and a public-key and yields a ciphertext, and the decryption function $dec(c, sk)$ takes a ciphertext and a secret-key. The message can be recovered if, and only if: $dec(enc(m, pk(sk)), sk) = m$.

The $pdec(c, sk)$ function models partial decryption. Partial decryptions can be combined using the $comb$ function, that given two partial decryptions $pdec(c, x)$ and $pdec(c, y)$, yields a decryption of the cipher with the sum of the private keys $dec(c, plus(x, y))$. This gives a set of primitives, where an encrypted message $enc(m, pk)$, with the joint public-key $comb(pk(sk1), pk(sk2))$, can only be recovered by partially decrypting with both $sk1$ and $sk2$. This is exactly the abstract definition of Distributed ElGamal, as presented in section 5.3.

6.2.3.1 *Defining the Actors*

Next we model the behaviour of the different actors of the protocol. The client actor is omitted in this model, as it simply relays messages between a service provider and authenticator.

REGISTRATION RULE Registration is modelled as one rule, where two new fresh long-term keys (ltk) for the authenticator and sibling, are fixed and persisted for later retrieval with the $!Ltk$ predicate. The key is saved with a relation to the device and the server, such that devices can have multiple keys for different service providers. A joint public-key is set as the combination of the public-key of the two ltk's. The public-key is persisted with the $!Pk$ predicate with a relation to the three actors. Finally the public-key is outputted and an event of the registration is recorded.

$$\begin{aligned} &\text{let } pk = comb(pk(\sim n), pk(\sim y)) \text{ in} \\ &\quad Fr(\sim x),\ Fr(\sim y) \\ &-[\ Register(\$S, \$A, \$As)\ ]\rightarrowtail \qquad \text{(registration)} \\ &\quad !Ltk(\$A, \$S, \sim x),\ !Ltk(\$As, \$S, \sim y), \\ &\quad !Pk(\$S, \$A, \$As, pk),\ Out(pk) \end{aligned}$$

SERVICE PROVIDER RULES The service providers behavior is modeled in two rules. The first rule initially fixes a new fresh nonce n and encrypts it with the public-key of a given authenticator and sibling.

It stores the nonce with the *Server* predicate and then outputs the challenge.

$$\begin{aligned} &\mathrm{Fr}(\sim n),\ !\mathrm{Pk}(\$S, \$A, \$As, pk) \rightarrow \\ &\mathit{Server}(pk, \sim n),\ \mathrm{Out}(enc(\sim n, pk)) \end{aligned} \tag{server-init}$$

The second server rule receives a response to the challenge, and thus takes as input a nonce n. The nonce is pattern-matched with the nonce given from the previous rule, and thus the rule can only be used if the two nonces match. An event is then fired, indicating that authentication between the service provider, authenticator and sibling was successfully achieved.

$$\begin{aligned} &\mathrm{In}(n),\ \mathit{Server}(pk, n),\ !\mathrm{Pk}(\$S, \$A, \$As, pk) \\ &\dashv\!\![\ \mathit{Auth}(\$S, \$A, \$As, enc(n, pk))\ ]\!\!\mapsto \emptyset \end{aligned} \tag{server-done}$$

AUTHENTICATOR RULES The authenticators behavior is also modeled as two rules. The first rule receives a challenge c, and saves the challenge as state with the *Authenticator* predicate. The challenge is then outputted to the *Secure* predicate, which is used to model communication with its sibling. This allows us to model authentic and secure communication between the actors.

$$\begin{aligned} &\mathrm{let}\ c = enc(\sim n, pk)\ \mathrm{in} \\ &\mathrm{In}(c) \rightarrow \\ &\mathit{Secure}(c),\ \mathit{Authenticator}(\$A, c) \end{aligned} \tag{authen-init}$$

In this rule we had to limit the input, to only accept ciphers encrypted with its public-key, as Tamarin will otherwise try to compute all states where the authenticator is given an arbitrary value. This is a reasonable limitation as all valid messages and ciphers belong to the same space $\mathbb{G}$. The input is limited by pattern matching.

The second rule takes as input a partial encryption from the secure channel, its persisted ltk, and the challenge that was received in the previous rule. An event that the authenticator answered the challenge is fired, and a combination of the partial encryption received from its sibling and its own partial encryption (yielding n if correct) is then outputted.

$$\begin{aligned} &\mathit{Secure}(c'_{As}),\ !\mathrm{Ltk}(\$A, \$S, x),\ \mathit{Authenticator}(\$A, c) \\ &\dashv\!\![\ \mathit{Acted}(\$A, c)\ ]\!\!\mapsto \\ &\mathit{Secure}(comb(pdec(c, x), c'_{As})) \end{aligned} \tag{authen-done}$$

SIBLING RULE The sibling is modeled as a single rule that on input of a challenge c retrieves its persisted ltk, fires an event that it answered the request, and outputs a partial decryption of the cipher.

$$\begin{aligned} &\mathit{Secure}(c),\ !\mathrm{Ltk}(\$As, \$S, y) \\ &\dashv\!\![\ \mathit{Acted}(\$As, c)\ ]\!\!\mapsto \\ &\mathit{Secure}(pdec(c, y)) \end{aligned} \tag{sibling}$$

6.2.3.2 *Modelling the Adversary*

In Tamarin the adversary can observe and intercept any message sent using the default In and Out predicates. Furthermore, the adversary can utilize any observed information to fabricate and send new messages. We furthermore model rules allowing the adversary to reveal keys, and to both break authenticity and secrecy on the Secure channel. In contrary to the default channel we however log if any of these rules are utilized.

$$!\mathrm{Ltk}(\mathrm{d}, \$\mathrm{S}, \sim\mathrm{ltk}) \dashv[\ \mathrm{LtkReveal}(\mathrm{d}, \$\mathrm{S})\ ]\mapsto \mathrm{Out}(\sim\mathrm{ltk}) \qquad \text{(reveal)}$$

$$\mathrm{In}(\mathrm{m}) \dashv[\ \mathrm{AuthenticityBroken}()\ ]\mapsto \mathrm{Secure}(\mathrm{m}) \qquad \text{(authentic)}$$

$$\mathrm{Secure}(\mathrm{m}) \dashv[\ \mathrm{SecrecyBroken}()\ ]\mapsto \mathrm{Out}(\mathrm{m}) \qquad \text{(secret)}$$

6.2.4 *Proving Injective Agreement and Unforgeability*

This section will present theorems to prove that our protocol achieves injective agreement for adversaries capable of *Strong Internal Observation*, if the communication between authenticator and sibling is authentic.

Before we can state the theorems we first establish a few definitions. First of all we need a definition for when an adversary either compromised a key or infiltrated the secure channel.

Definition 6.11. *We say that an adversary compromised a long-term key of* d *for* S *before* i*, if there exists events* $\mathrm{LtkReveal}(\mathrm{d}, \mathrm{S})$ *at time* j *with* $\mathrm{j} < \mathrm{i}$*.*

Definition 6.12. *We say that secrecy is interact at time* i*, if there does not exists an event* $\mathrm{SecrecyBroken}()$ *at time* j *with* $\mathrm{j} < \mathrm{i}$*.*

Definition 6.13. *We say that authenticity is interact at time* i*, if there does not exists an event* $\mathrm{AuthenticityBroken}()$ *at time* j *with* $\mathrm{j} < \mathrm{i}$*.*

Furthermore we need a definition for when an honest device completed a run of the protocol.

Definition 6.14. *We say that a device* d *acted on a challenge* c *before time* i*, if there exists an event* $\mathrm{Acted}(\mathrm{d}, \mathrm{c})$ *at time* j *with* $\mathrm{j} < \mathrm{i}$*.*

An implication of the atomicity of both authenticator and sibling rules as well as the definition of 'acted', is that a device is either acting or not acting in a run of the protocol. In practice, things are not so binary. An adversary could, for example, compromise the device, and not be able to obtain the key, but instead always accept explicit consent request without any user interaction. This could lead to attacks, but is not entailed by the model. This will be revisited in the discussion (section 8).

6.2.4.1 *Proving Unforgeability*

In the following, we state a definition of unforgeability in the presence of adversaries capable of *Internal Observation*. We define *Internal Observation* to be slightly weaker than *Strong Internal Observation* and now assume that messages between authenticator and sibling are authentic.

Theorem 6.15. *Our protocol has unforgeability in the presence of adaptive adversaries capable of Internal Observation, if for all authentication events* $\mathtt{Auth}(S, A, As, c)$ *at time i, where authenticity is still intact at* i*, then either:*

1. *Both authenticator and sibling acted on* c *before* i*, or*
2. *An adversary compromised the long-term keys of both* A *and* As *for* S *before* i*.*

The theorem is proven and holds in the presented model. Furthermore, the definitions of unforgeability, as presented in the previous section, is revisited. In the symbolic model of the protocol, the adversarial capabilities are defined, as shown in table 8. Note that the use of 'Observation' is not strictly correct in this model as we allow adversaries to also synthesize and send messages through Out and Authentic.

As expected, unforgeability can be proven in the presence of adversaries capable of *Weak External Observation*, *Weak Internal Observation* and *Strong External Observation*. See appendix D.

	In/Out	*Secret*	*Authentic*	*LtkReveal*
Weak External Observation	●			
Weak Internal Observation	●			●
Strong External Observation	●	●	●	
Strong Internal Observation	●	●	●	●
Internal Observation	●	●		●

Table 8: Definitions of adversarial capabilities

Furthermore, we confirm that an attack for unforgeability is also found in this model for adversaries capable of *Strong Internal Observation*, although a slightly different strategy is chosen as the adversary compromises the sibling instead of the authenticator. The attack trace is shown in appendix E.

6.2.4.2 *Proving Injective Agreement*

We stated in the beginning of the protocol design (chapter 5) that authentication was achieved if and only if injective agreement was achieved. We will now present the definition of injective agreement in the presence of adversaries capable of *Internal Observation* as proving in the model:

Theorem 6.16. *Our protocol achieves injective agreement in the presence of adaptive adversaries capable of Internal Observation, if for all authentication events* $a_1 = \mathtt{Auth}(S, A, As, c)$ *at time* i*, where authenticity is still intact* i*, then either:*

1. *Both authenticator and sibling acted on* c *before* i*, and for all authentication events* $a_2 = \mathtt{Auth}(S, \cdot, \cdot, c)$*,* a_1 *is equal to* a_2*, or*

2. *An adversary compromised the long-term keys of both* A *and* As *for* S *before* i.

6.2.4.3 *Proving Actor Involvement*

Lastly we will prove that, unless the keys of both authenticator and sibling are revealed, then either authenticator or sibling acted in the protocol run. This is important as the participation of, at least one device, allows the user to 1) potentially be made aware of active authentication sessions, and 2) be able to lockdown the device and effectively stop all active authentications. The following definition is proven in the symbolic model:

Theorem 6.17. *For all authentication events* $a_1 = \mathtt{Auth}(S, A, As, c)$ *at time* i*, then either:*

1. *The authenticator or sibling acted on* c *before* i*, or*
2. *An adversary compromised the long-term keys of both* A *and* As *for* S *before* i.

7 IMPLEMENTATION

This chapter introduces a prototype implementation of the design that was presented in chapter 4. As is the nature of a prototype, not all features are implemented, and the following section will outline the objectives and scope of the prototype before going into implementation details.

7.1 GOAL AND SCOPE

The objective of the prototype is to showcase that the proposed protocol can be used to support the proposed design. In particular we partially implement scenarios "Registration with a Service Provider" (table 3) and "Establishing a session" (table 5).

These scenarios will be implemented to first of all to demonstrate, that the proposed authentication protocol for registration and authentication can be implemented and executed, exactly as described in chapter 5, using the specified cryptographic primitives.

Second, we want to show that the protocol can also be implemented in a distributed fashion, across four different hardware platforms: smartwatch, smartphone, browser-client and server, where each party executes their part of the protocol correctly. Herein also lies the challenge of getting the devices to communicate with low latency and stably in a manner that supports continuous authentication.

Third, we want to show that a proximity-based authentication mechanism is feasible to implement, and that a continuous authentication session can be established and will be interrupted whenever the watch and phone is not in proximity of the client.

Lastly, we want to include user awareness and explicit consent in our prototype, in order to demonstrate that some of the more usability related properties of our scheme, is also possible to implement in practice.

All other features, such as "Unlocking the Authenticator" (table 4), and thereby also monitoring if the user is wearing the authenticator, and "Theft and Lost" (table 6), are left out of scope.

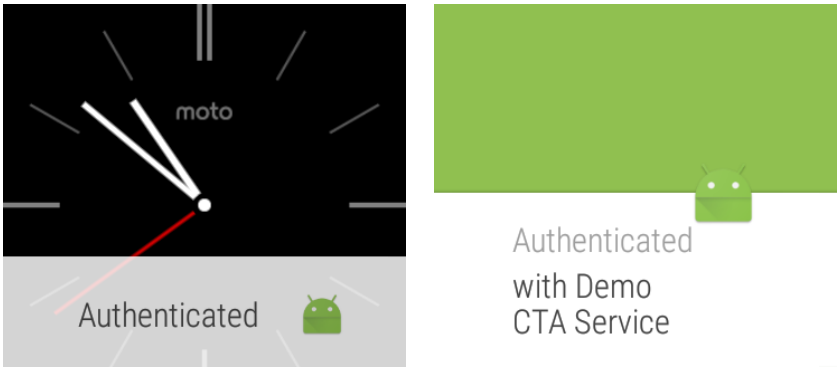
7.2 SOLUTION DESCRIPTION

The implemented prototype system is based on the scheme design described in chapter 4 and consists of the 4 main components: a smartphone acting as authenticator, a smartwatch acting as sibling, a browser acting as client and a server application acting as service provider.

The attentive reader might notice that the roles of the smartwatch and smartphone are switched in the prototype compared to the orig-

inal design. We experienced stability issues when the watch was the party communicating with the client, and for ease of implementing the prototype, the roles are therefore reversed.

The authenticator and sibling are implemented as background services that run on a user's smartphone and smartwatch respectively. The main responsibilities of the authenticator and sibling are to generate keys, decrypt challenges, and exchange messages that allow for user registration and continuous authentication. The implementation follows the specified protocol (chapter 5), in terms of when and where different cryptographic computations are carried out, and the sequence in which messages are exchanged between the 4 parties for registration and authentication. Both authenticator and sibling notifies the user whenever authentication or registration processes are successful. Additionally the authenticator (smartphone) also prompts the user to actively accept or decline incoming registration requests from the client. The notifications that requires explicit consent from the user is shown in figure 5 and 6.

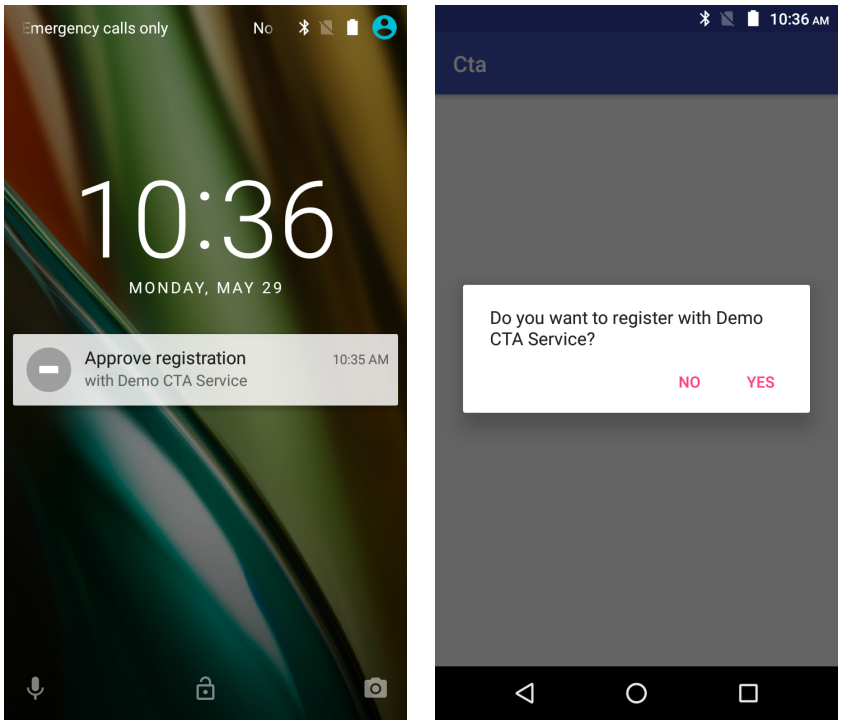


(a) Passive notification (b) When selected

Figure 5: Example of notifications on the watch

The service provider is implemented as a server application that receives requests from the client. Upon registration requests from the client, the server is responsible for checking that the supplied username is available, and not already in use. If available it will respond to the client, which will then initiate the registration process as specified in the protocol. Upon authentication requests from the client, the server is responsible for encrypting new challenge ciphers and waiting for the client to respond back with the correct cleartext message. Upon successful authentication the server generates a new token with a set expiration time, that is returned to the client. This token allows the client to authenticate with the service provider as long as it is valid.

The client is implemented as a small web application that runs in a browser. The client is responsible for receiving input from the user, in the form of instructions of when to authenticate and register the user. Before either authentication or registration can be started from the client, it must establish a connection with an authenticator and a service provider so that it can exchange protocol messages between the



(a) Registration notification (b) When selected

Figure 6: Example of explicit consent on the phone

two parties. The client provides a simple user interface with a 'pair' button used to select the correct authenticator from a list of nearby Bluetooth devices. It has a field for typing a username, a button to perform registration and one to perform authentication. It also shows a log for displaying output from the client itself and response messages from its communicating parties. Lastly, it has a button used to interrupt the continuous authentication cycle with the service provider, available whenever the client is authenticated.

7.3 TECHNICAL DESCRIPTION

7.3.1 *Technologies Used*

One clear and important design goal, is to base the prototype implementation of our authentication scheme on commodity hardware and recognized technologies, and as such, this should be clearly reflected in our choice of technologies used.

The prototype is using the Android platform for the authenticator and sibling, and developed using Java. All message exchanges between the two devices are handled using the Bluetooth communication protocol. For developing and testing we used Motorolas Moto 360 smartwatch and their Moto E^3 smartphone, although the hardware can be substituted with any two Bluetooth 4.0 LE supported devices that run Android OS and Android Wear OS respectively, with API level 21 or higher.

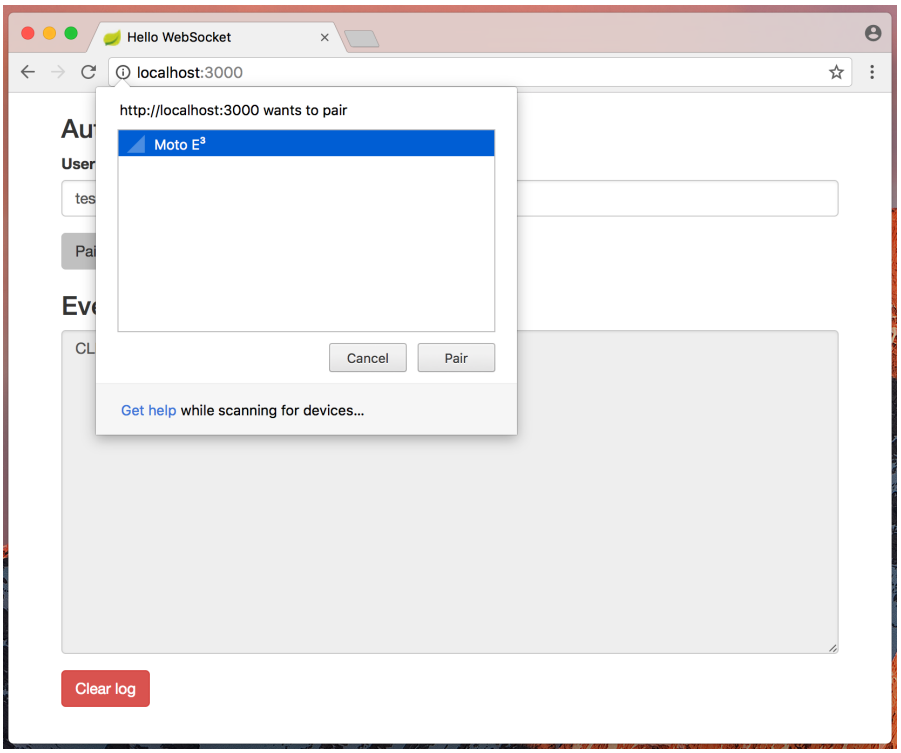


Figure 7: The pairing menu in Chrome

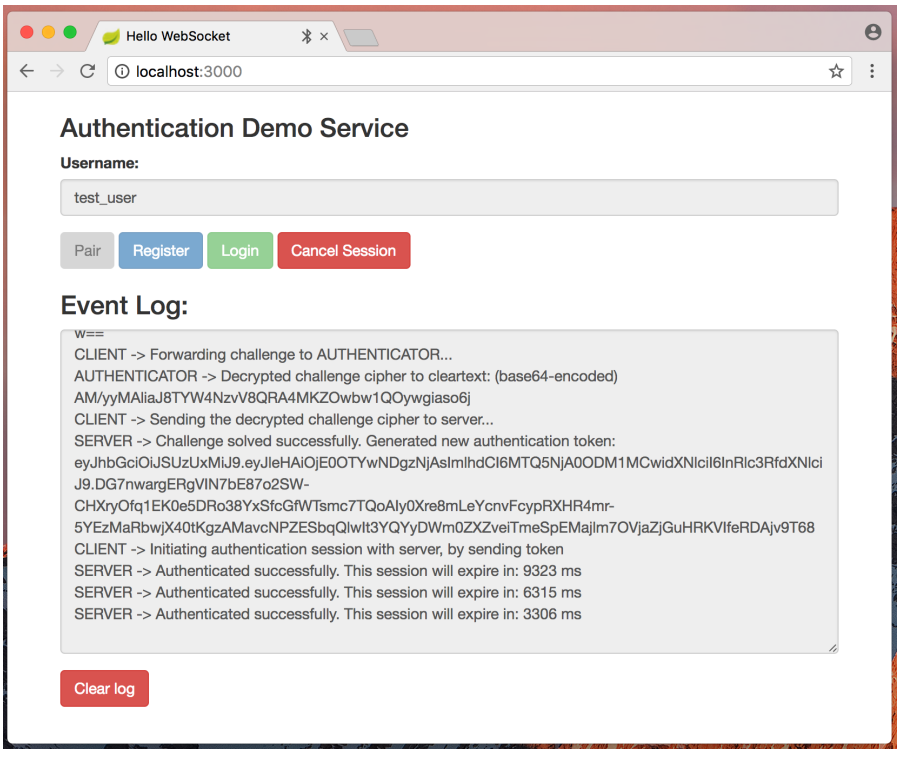


Figure 8: Authenticating with a service in Chrome

The client is running a browser application that is served statically as Javascript and HTML files.

We use Bluetooth GATT services to establish a link between client and authenticator

Generic Attributes (GATT) profile is a feature in Bluetooth LE that allows device discovery based on services without device pairing. A common use of GATT is with sensors, such as heart-rate monitors, that can easily be discovered as used ad-hoc.

We use GATT to expose an authentication service from the authenticator that be easily discovered, and can be used without regular device pairing.

Because native GATT service compatibility in the browser is still under active development, the client application is only functioning properly in Google Chrome Version 59 or later, which at the time of writing is still in beta release state. When selecting a new unknown device in Chrome for the first time, Chrome as a security requires that the end-user selects the device from a pairing drop-down menu. This is shown in figure **??**. The implementation will, but does not yet support automatically establishing connections to known devices when they come in range.

The service provider is implemented as a server application using Java and the Spring Boot framework. Communication between client and server is handled through the WebSocket communication protocol. Public keys user information and generated challenges, are persisted on the server using a MySQL relational database. The server uses the open standard (RFC 7519) JSON Web Tokens (JWT), to issue and verify authentication tokens. The details of token generation and verification are explained in section 7.3.4 below.

All cryptographic computations are carried out using the BouncyCastle cryptography Java API. However as the Android platform ships with a limited version of BouncyCastle, and can not be updated to the latest version due to classloader conflicts, we use SpongyCastle to perform cryptographic operations on the authenticator and sibling. SpongyCastle is essentially the exact same API as BouncyCastle only with a different name, created only with the purpose of enabling the full and updated API of BouncyCastle on the Android platform. The details of our cryptography implementation is explained in section 7.3.3 below.

7.3.2 *Algorithms*

The implementation of the protocol on the service provider, authenticator and sibling is based on the algorithms defined below.

Algorithm 1: Service Provider behaviour during authentication

Data: Set of public-keys $\{pk_1, pk_2, ..., pk_n\}$

$i \leftarrow recv(C)$; /* request from user with id i */
$c = Enc(n, pk_i)$; /* generates a random challenge */

$send(C, \langle c, i \rangle)$; /* sends a challenge to C */
$n' \leftarrow recv(C)$; /* receives a response */

if $n = n'$ **then** /* if the response is valid */
  $send(C, token)$; /* then issue a token to C */
end

Algorithm 2: Authenticator behaviour during authentication

Data: Set of key-pairs $\{sk_1, sk_2, ..., sk_n\}$

$\langle c, i \rangle \leftarrow recv(C)$; /* receives challenge from C */
$send(As, \langle c, i \rangle)$; /* sends challenge to As */

$c'_A = Dec'(c, sk_i)$; /* partly decrypts */
$c'_{As} \leftarrow recv(As)$; /* receives other half from As */

$n' = c'_A \times c'_{As}$; /* combines the parts */
$send(C, n')$; /* send the token to C */

Algorithm 3: Sibling behaviour during authentication

Data: Set of private-keys $\{sk_1, sk_2, ..., sk_n\}$

$\langle c, i \rangle \leftarrow recv(A)$; /* receives challenge from A */
$c'_{As} = Dec'(c, sk_i)$; /* partly decrypts */
$send(A, c'_{As})$; /* sends it to A */

7.3.3 *Cryptography Implementation*

As earlier mentioned, BouncyCastle cryptography API (SpongyCastle on Android) is used, to implement the protocol in practice. The different cryptographic operations needed for the authenticator and sibling are exposed in a shared class which they both use. Initially the encryption/decryption scheme to be used, is set to El Gamal without padding, and the bit length is set to 256 for all generated keys (lst. A, line 47-66).

When a user registers with a new service provider, the sibling and the authenticator creates an El Gamal key pair each. Key generation is trivial to implement and is handled trough the BouncyCastle API (lst. A, line 133-152). To reduce key generation time significantly we avoid generating new large prime numbers required for the El Gamal key parameters p and g each time, but instead uses fixed values for both, that we know are applicable for secure key generation (lst. A, line 36-37). When the key pairs are successfully created on each device, the sibling sends its public key y_{as} to the authenticator, which then adds the siblings public key y_{as} to its own public key y_a, resulting in a combined public key y (lst. A, line 69-85). The authenticator now sends its public key pk (y along with parameters p and g) to the service provider, where it is stored and associated with the registering user.

When an authentication request is initiated by the user, the server application encrypts a new challenge cipher, based on the public key it received upon registration. The challenge is generated using the formula $(h^2 \bmod p)$ where h is an arbitrary integer of $\mathbb{Z}_p^*$ in the range $\{2, ..., p-2\}$. Naturally, the server also uses the El Gamal No Padding cipher to perform encryption. The server stores the challenge in the database in both cleartext and ciphertext along with an expiration timestamp indicating how long the service provider accepts responses for the generated challenge (lst. B, line 46-81).

The challenge is sent to the authenticator, which will now decrypt the cipher in cooperation with its sibling. As shown in algorithm: 2, the authenticator sends a copy of the challenge to the sibling immediately upon receiving it from the server. Hereafter both devices will perform a partial decryption of the cipher, using their individual private key. This is done by first splitting the challenge cipher into two parts: α and β (lst. B, line 168-181). Partial decryption is then done by: $\left[\beta^{(p-1-x)} \bmod p\right]$, where x is the private key, and p is the common security parameter of the public key (lst. B, line 98-109). Note that, as $p-1$ is the order of the group used, then $h^{(p-1)} = 1$ with $h \in \mathbb{G}$.

$$c_i' = \beta^{(p-1-x_i)} = \frac{\beta^{(p-1)}}{\beta^{x_i}} = \beta^{-x_i}$$

Note that this is different from the partial decryption as defined in the protocol section. This is due to the fact that division in groups in Bouncy Castle is defined as $a/b \stackrel{def}{=} \left[ab^{-1} \bmod p\right]$.

The main difference between the algorithm executed by the authenticator (algorithm 2) and the one (algorithm 3) executed by the sibling,

is that the sibling sends the result of its decryption directly to the authenticator, which combines the two partial ciphers into the correct cleartext solution of the challenge. The authenticator does so by computing $\left[\alpha \cdot c_1' \cdot c_2' \bmod p\right]$ where c_1' and c_2' are the partial ciphers from the authenticator and sibling (lst. B, line 87-96). This formula computes the solution to the challenge, which is then sent back to the server, so it can complete the last step of its algorithm (algorithm 1), which is to authenticate the client.

7.3.4 *Token Issuing and Verification*

The prototype is using JSON Web Tokens (jwt.io), to issue new authentication tokens on the server, upon receiving a correct solution to a challenge. The tokens are signed digitally with a public/private key pair using RSA. JWT tokens can carry different kinds of information with them, such as a timestamp to indicate an expiration time for the token. Our implementation uses the JWT Java API provided by Auth0(GitHub page). The server generates a new RSA key pair upon server start (lst. C, line 37-52). When a new token has to be issued for a given user, the following information is stored in the token: a timestamp for when the token was issued, a timestamp for when the token is set to expire, the username of the authenticated user. The token is then signed with RSA512 using the private key that was generated on server start (lst. C, line 54-73). Verification is done using the RSA public key. The verification function of the API takes an RSA public key as input, and checks and that the token is signed by the corresponding private key and not expired yet (lst. C, line 75-93).

7.4 PROTOTYPE EVALUATION

We evaluate our implementation based on the framework proposed by Bonneau et al. [9], in a similar manner to how we evaluated the design of our scheme in section 4.4.2, and other schemes in chapter 3.

7.4.1 *Rating process*

To avoid repeating ourselves, we only describe the rating rationale behind the specific benefits of the prototype implementation that differ from the scheme design. We grant the prototype *No-Config* as it actually offers this benefit according to our definition, since there is no difference between the required effort from the user, the first time or sub sequential times it is used on any client. This may appear more beneficial than it actually is. In reality, a pairing step between authenticator and client, becomes something the user has to do every single time he starts a new authentication or registration request, instead of something that should only be done once on each client, which obviously is less desirable. This is also the reason why we only grant the prototype *Quasi-Efficient-to-Use* as the user is required to select the correct authenticator from a list of devices, on the client, every

		Usability										**Deployability**					**Security**											
	Reference	*Memorywise-Effortless*	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Awareness*	*No-Config*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*	*No-Intermediary-Knowledge*	*Continuous*
Scheme design		●	●	○	●	●	●	●		●		●	○			●		●	●	●	●	○	●	●	○	●	○	●	●	●
Prototype Implementation	[22]	●	●	○	●	●	○	●		●	●	●	○	○	○	●		●	●	●	●	○	●	●	○	●	○	●	●	●

● = offers the benefit ○ = almost offers the benefit ? = not known

Table 9: Comparing the (envisioned) benefits of our design with the actual benefits of our implementation.

time the browser reloads the page. While deployability benefits are not prioritized in the scheme design, it was discovered during implementation that it was actually possible to achieve both *Browser-* and *Server-Compatibility* to some extent. We award the prototype *Quasi-Browser-Compatibility* as no browser plugins or additional software is required to run on the client machine, in order to use the prototype. However the browser support for Bluetooth GATT services is used in the implementation, is still in a beta state, and currently only works in a limited amount of browsers. Assuming that Bluetooth communication directly through the browser, becomes more common and widely supported in the future, we award *Quasi-Browser-Compatibility*. We award *Quasi-Server-Compatibility*, since servers that already use JWT tokens for authentication, can become compatible with the scheme, without a complete re-implementation of its authentication infrastructure . Essentially such servers do not require changes, in terms of how it is determined whether a user is authorized to access a requested resource or not. Instead, a change is needed, in the way it is determined whether a user is entitled to receive an authentication token or not, which today is typically done by verifying a user's username/-password credentials.

Part III

DISCUSSION & CONCLUSION

8 DISCUSSION

8.1 SECURITY ASSUMPTIONS AND ATTACKS

In the security analysis, we analyzed the proposed protocol and showed that the protocol has an injective agreement in the presence of adversaries capable of *Internal Observation*. However, we also showed that if an adversary is capable of *Strong Internal Observation*, then he can mount an attack as shown in figure 9.

So far we have not discussed the implications of this attack. Most realistically this attack can be mounted by compromising the authenticator, as this will give the adversary access to both the key, and the communication to the sibling. With control of the network, the adversary can send packages to the sibling and make it partially decrypt selected ciphers.

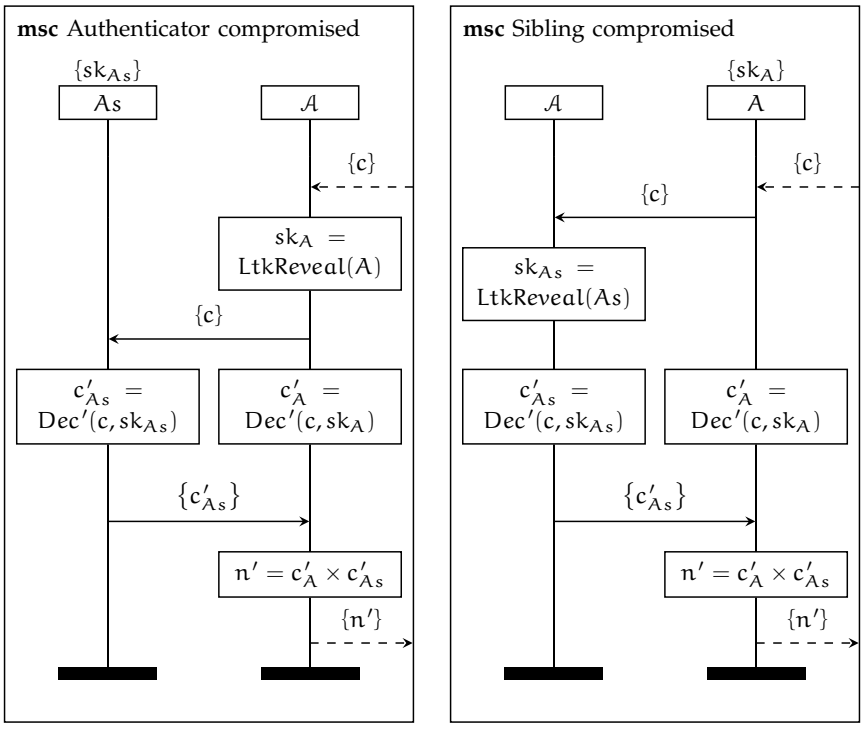


Figure 9: The known attacks by adversaries capable of Strong Internal Observation

So what does the adversary gain from this attack? The response to a challenge could be obtained much easier by eavesdropping the plain-text nonce when it is sent from authenticator to the client and then to the service provider. Furthermore, the user is still able to stop further authentications by removing the sibling from the authenticators proximity, or simply turning it off.

This is proven as: For all successful rounds of authentications, if an adversary is limited to a proper subset of keys, then either the

authenticator or sibling was involved in the authentication run (as defined in section 6.2.4.3).

This is an important property as, even if some subset of devices are compromised, then the user retains the ability to stop all current and future authentications by turning off the device, or putting the device on lockdown. Recall that this was one of the scenarios presented in the design (see table 6).

However, a more severe consequence of the attack; In the protocol design session we stated that hijacking a session should not give an adversary unauthorized access for longer than the session is actively kept alive by the authenticator and service provider. The attack would violate this goal as it allows the adversary to omit the authenticators logic and keep a session alive even after the authenticator is locked (e.g. if it is not worn). Furthermore, if the adversary can compute the response to a challenge, and thus omit user-awareness, explicit consent and other checks and bounds implemented on the authentication, then it undermines many of the user-oriented security features that our design presents.

In the current version of the protocol, no messages sent are authentic. As we have shown that the known attacks can be mitigated by having authentic communication, future work should include to further analyze and extend the protocol to have message authenticity.

In the security analysis, registration is assumed to be an atomic operation. This is, of course, not the case in practice. The registration is, as most key-exchanges are, vulnerable to man-in-the-middle attacks. Password-based authentication also suffers from this problem, and a common fix is to use secure channels such as HTTPS. However, for our protocol, securing the communication between client and service provider with HTTPS, only partly mitigates the attack surface. It is left for future work to analyze and prevent man-in-the-middle attacks between client, authenticator and sibling during registration.

8.2 CONFIDENCE IN TAMARIN

As we pointed out both in this chapter and in the security analysis, two strategies for compromising the protocol are known. Both attacks assume that an adversary is capable of *Strong Internal Observation*, and is thus able to compromise the keys of one actor, and to break message authenticity and secrecy. The two attacks are shown in figure 9.

While both attacks can be mounted by adversaries capable of *Strong Internal Observation*, and thus disprove proposition 6.10 in our computational model, we have discovered that Tamarin does not find the second attack where the attack is mounted by compromising the key of the sibling.

Although the adversary has all the knowledge and rules available to mount the attack; it does not discover the attack unless a rule explicitly stating the strategy is given in the model

$$\mathrm{In}(c),\ \mathrm{In}(x),\ \mathrm{In}(p) \rightarrow \mathrm{Out}(\mathrm{comb}(\mathrm{pdec}(c,x),p))$$

It should be clear from the rule that an adversary with the information necessary to invoke the rule,

$$\begin{aligned} &c = enc(n, pk) && \text{(from server-init)} \\ &p = pdec(c, y) && \text{(from sibling)} \\ &x && \text{(from reveal)} \end{aligned}$$

with $pk = pk(plus(x, y))$, should already be to able obtain n given our equational theory.

$$\begin{aligned} &dec(enc(m, pk(sk)), sk) \simeq m \\ &comb(pdec(c, x), pdec(c, y)) \simeq dec(c, pk(plus(x, y))) \end{aligned}$$

However, the attack is only found if the rule is explicitly present in the model. We have been in contact with the developer of Tamarin who confirms that it seems to be a bug in the tool[1].

This naturally lead us to question the validity of the proofs given in section 6.2. We are, even considering the bug, confident that the given theorems of unforgeability and injective agreement holds. However, we suggest for future work to prove the theorems with a different tool or model to obtain certainty hereof.

8.3 TRUSTED INPUT, OUTPUT AND COMPUTATIONS

In the security analysis we proved that the protocol achieves unforgeability in the presence of *Strong External Observation*, and as such, if an adversary can not obtain any of the secret keys, then unforgeability is proven. So can we make an implementation where we can reasonably assume that adversaries are only capable of *Strong External Observation*?

A way to ensure the secrecy of the keys is to use TPM. In his thesis Bentzon [6] showcases an implementation of Pico that mitigates the risk of the Pico being compromised in an unlocked state by using a Trusted Platform Module (TPM), which are small special purpose hardware chips that allow for compartmentalization, as both keys and cryptographic calculations can be carried out in separation from the operating system.

TPM modules are standard in most commodity devices such as laptops and phones, and using a TPM for storing the keys and performing the cryptographic calculations is an effective means of mitigating the risk of compromised devices.

However, as hinted previously, a flaw in the way we modeled and proved our protocol is that the definition of an actors involvement is too binary. In practice, we have so far been assuming trusted input and output. This means we assume that if the authenticator acted, then user-awareness, explicit-consent and other checks and bounds was actually performed. However, even if an adversary is not able to compromise the secrets of the system, he might be able to compromise the user in- and output. As we have already explained above,

1 https://github.com/tamarin-prover/tamarin-prover/issues/216

this can have serious ramifications for the user-oriented security features of our scheme.

We therefore suggest for future work to include exploring how trusted input, -output and -computations can be used to mitigate the risk of attacks.

8.4 NUMBER OF SIBLINGS

The concept of siblings is crucial to our design as it mitigates the risk of both theft and compromised devices. By splitting the secrets onto multiple devices, we make it more difficult for an adversary to mount an attack.

The concept originates from the Pico [22, 23]. However, in contrast to our design, multiple siblings are suggested. This lead us to an interesting discussion point: How many devices should the user be obligated to have in his vicinity?

Stajano [22] suggests that siblings could be items such as glasses, belts, wallets, jewelry and possible implants. Our design choice of using only one sibling stems from a practical consideration. Forcing the user to always have both watch and smartphone present when authenticating is already a significant trade-off.

However, we do see the potential in having the number of siblings as a variable parameter of security. In fact, our protocol naturally supports having more than one sibling as Distributed ElGamal works for any number of participant, and the scheme could easily be extended to allow more siblings.

Pico uses a k-out-of-n threshold encryption scheme allowing a user to authenticate even though some siblings are not present. There are multiple interesting aspects to this: there is the natural benefit of allowing the user to forget one of multiple devices, but even more interestingly it could be used as a recovery mechanism where if the user losses e.g. one out of two siblings in a 2-out-of-3 system, then still having the majority of devices could allow the user to replace a lost device.

Our current use of Distributed ElGamal enforces that all devices are bound to participate. However, Distributed ElGamal can quite elegantly be extended to a k-out-of-n threshold encryption system by choosing a joint private-key x, and then dealing it into several shares $x_1, ..., x_n$ with Shamir's secret sharing scheme [21]. This would allow for computing partial decryptions as we already do, but only needing k-out-of-n shares to recover the message [15, page 506].

8.5 RECOVERY PROBLEM

The current design of our scheme does not include any proper solution to how a user can recover from a loss of a sibling or authenticator. As described in the user scenario for theft and loss (6), we envision that a user can initiate a lockdown state, if either of his devices gets

stolen or lost. The lockdown state will render both devices useless, until the lockdown state is explicitly canceled by the user, in the event that the lost device resurfaces. However, this idea only considers the case where *one* of the devices is lost, and what happens if the lost device never resurfaces? Surely some recovery mechanism must be put in place that allows replacement of the lost or stolen devices, so that the user can regain control of his web service accounts. The challenge of designing such a mechanism is to make sure that only the legitimate user is allowed to perform recovery. Obviously, it would be catastrophic if an attacker could abuse the recovery process, to gain access and lock out a legitimate user.

We believe that we have yet to discover the ideal solution for this problem if any such exists. Taking inspiration from Pico's solution to this problem [22], we have considered the concept of a recovery docking station, that can be placed at the user's home. When the authenticator and sibling are physically plugged into the dock, all the cryptographic keys contained on the devices are transferred onto the docking station, to be used for recovery later. This way it will be fairly easy, for a user to perform recovery once he is at the recovery station. However, this solution is still problematic in the case where both devices are lost or stolen, since a thief in possession of two stolen devices now has the required equipment to perform authentication. One solution to this problem could be to introduce a trusted third party, in the form of a centralized server, that basically acts as a second sibling. The authenticator would then have to contact the third party and request a partial decryption from it, in order to successfully authenticate with the service provider. The authenticator must sign its request, such that the third party can verify that the request is coming from the authenticator carried by the legitimate user. Introducing a third party makes it easier to perform lockdown for a user, if both his authenticator and sibling is lost. If a user performs recovery, the third party can then be instructed to only answer requests from the new authenticator used for recovery, instead of the stolen or lost authenticator.

To make sure that it actually is the legitimate user who performs this recovery process, a text- or QR-code could be stored on the recovery station, which must be supplied to the trusted third party, before it can be instructed to answer requests from a new authenticator. In order to view this code or transfer the backup keys onto a new device, the recovery station could, for instance, be protected with a PIN code or a fingerprint scanner.

At this point, it should be obvious that the problem of recovery is not trivial to solve. While a solution such as the one described above could make recovery viable, it is still far from ideal, as it introduces a new weakest link in the scheme (stealing the recovery station), and furthermore increases the overall cost of deployment, due to the manufacturing cost of the recovery station, and the server maintenance expenses from using a trusted third party.

8.6 REFLECTING ON EVALUATION RESULTS

The evaluation results of the proposed design (figure 7) indicate that the scheme generally performs very well in both security and usability aspects. The main concern in the usability area when compared with related schemes, is that our design does not entail a solution for recovery in case of loss. This problem is discussed above in section 8.5. In the area of security, the most significant shortcoming is the lack of full *Resilience-to-Internal-Observation*. This issue is discussed in both chapter 6 and above in section 8.1. Comparing our proposed authentication scheme with passwords, which is the scheme that is ultimately the target for replacement, we notice the significant lack of provided deployability benefits. As already explained in section 2.1.2 one of the main reasons why passwords have not been replaced on a global scale, is because of its maturity, the fact that it is nearly cost free, and the enormous dependency that exists on passwords in current browser and server infrastructures. In other words it is deployability properties.

With this thesis, we have clearly stated that the proposed scheme is intended as a password replacement. One could therefore rightfully ask why we design the scheme with such small emphasis on deployability, if the lack of deployability is so detrimental to a schemes chance of being widely adopted or contesting passwords in any way. The short answer is that full browser and server compatibility is basically only possible to achieve, if the scheme is an abstraction over passwords, such as password managers or a supplement to passwords such as two-factor authentication methods. These solutions have the drawback, that in the end, the weakest link will still be passwords, and such solutions therefore still inherit several of the undesirable security issues that passwords entail.

We strongly believe that a clean-slate solution is the only way to get completely rid of passwords and achieve optimal security and usability simultaneously. We realize this comes at the significant cost of changes to many browser and server implementations. However, when implementing our prototype, we discovered that it was actually possible to reduce a number of changes needed on both servers and browsers to some extent (see section 7.4). As many servers are already using a token authentication system, it is possible to only substitute the part of server implementation that issues tokens, and keep the rest of the infrastructure intact. Furthermore, the recent advancements in the compatibility of Bluetooth GATT Service in browsers, enables our scheme to work without implementation changes to browsers.

While the scheme we propose may require some implementation changes to existing infrastructure, and a more broad adoption of wearable devices, it still provides major improvements in almost all other aspects compared to passwords.

9 CONCLUSION

The motivation for this thesis has been to identify or propose a replacement to password-based authentication. Password-based authentication is the most commonly used end-user authentication mechanism, although research concludes it to be insecure and user-unfriendly. Within the field of pervasive computing, the search for a more user-friendly and unobtrusive authentication method, has long been a topic of research, as the amount of computers a user interacts (and eventually authenticates) with, on a daily basis, is steadily increasing.

In our work we reviewed and analyzed some existing and relevant pervasive authentication schemes, to gain insights into what works well, and what needs improvement. The solutions we surveyed [3, 20, 22] are research solutions that are not mature, and all lack a proper implementation. Another observation is that existing solutions tend to rely on dedicated and expensive hardware to facilitate the authentication and furthermore place a primary focus on usability aspects, and prioritize security and cryptographic considerations to a lesser extent.

Based on the review, this thesis presents a new password-less authentication scheme, designed to rely on commodity hardware (wearables), and includes and extends upon some of the well working concepts from existing work, such as user awareness, continuous and transparent authentication, in order to achieve a high degree of both usability and security. We describe a well specified and formally verified authentication protocol, that can facilitate the scheme in a secure manner, and provide a thorough documentation of the verification process along with a security analysis of possible vulnerabilities and attacks.

Based on the design, a working prototype is developed, that serves to demonstrate successful rounds of continuous authentication using our specified protocol. The developed prototype certainly fulfills the goal of avoiding dedicated and special purpose technology as it requires only a Bluetooth-compatible smartphone, smartwatch, and a browser to be used. While a lot of work is still needed, if the authentication scheme proposed in this thesis was to be used in an actual production setting, we claim that our prototype showcases that it is actually feasible to implement the envisioned scheme in practice.

As the security and usability flaws of passwords have become more widely acknowledged, not only in the security community, but in general, it still remains a major challenge to find a suitable replacement, as current client and server technology is so bound to the password paradigm. Obtaining the incredibly low cost, and the benefits of an already well-established infrastructure, offered by passwords is very hard to compete with. Evaluating our presented scheme, we are convinced that it offers way more usability and security than compared

to passwords, which should hopefully also be obvious from the documentation provided in this report. We modestly hope, that the scheme and protocol design presented in this thesis, can bring us a little closer to a password-less world, or at the very least shed light on the challenges involved in designing an authentication scheme that must be both usable, secure and deployable.

REFERENCES

[1] Anne Adams and Martina Angela Sasse. "Users are not the enemy." In: *Communications of the ACM* 42.12 (1999), pp. 40–46.

[2] Jakob E Bardram. "The trouble with login: on usability and computer security in ubiquitous computing." In: *Personal and Ubiquitous Computing* 9.6 (2005), pp. 357–367.

[3] Jakob E Bardram et al. "Context-aware user authentication–supporting proximity-based login in pervasive computing." In: *International Conference on Ubiquitous Computing*. Springer. 2003, pp. 107–123.

[4] David Basin et al. *Applied information security: a hands-on approach*. Springer Science & Business Media, 2011.

[5] Victoria Bellotti and Keith Edwards. "Intelligibility and accountability: human considerations in context-aware systems." In: *Human–Computer Interaction* 16.2-4 (2001), pp. 193–212.

[6] Anders Bentzon. "Security architecture and implementation for a TPM-based mobile authentication device." PhD thesis. Master's thesis, 2013.

[7] Andrea Bianchi and Ian Oakley. "Wearable authentication: Trends and opportunities." In: *it-Information Technology* 58.5 (2016), pp. 255–262.

[8] Bruno Blanchet. "Security protocol verification: Symbolic and computational models." In: *Proceedings of the First international conference on Principles of Security and Trust*. Springer-Verlag. 2012, pp. 3–29.

[9] Joseph Bonneau et al. "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.

[10] Joseph Bonneau et al. "The quest to replace passwords: a framework for comparative evaluation of web authentication schemes." In: UCAM-CL-TR-817. Mar. 2012. URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-817.pdf.

[11] Felix Brandt. "Efficient cryptographic protocol design based on distributed El Gamal encryption." In: *International Conference on Information Security and Cryptology*. Springer. 2005, pp. 32–47.

[12] Andrea Ceccarelli et al. "Continuous and transparent user identity verification for secure internet services." In: *IEEE Transactions on Dependable and Secure Computing* 12.3 (2015), pp. 270–283.

[13] Danny Dolev and Andrew Yao. "On the security of public key protocols." In: *IEEE Transactions on information theory* 29.2 (1983), pp. 198–208.

[14] Eiji Hayashi and Jason Hong. "A diary study of password usage in daily life." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 2627–2630.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[16] Gavin Lowe. "A hierarchy of authentication specifications." In: *Computer security foundations workshop, 1997. Proceedings., 10th*. IEEE. 1997, pp. 31–43.

[17] Simon Meier et al. "The TAMARIN prover for the symbolic analysis of security protocols." In: *International Conference on Computer Aided Verification*. Springer. 2013, pp. 696–701.

[18] Robert Morris and Ken Thompson. "Password security: A case history." In: *Communications of the ACM* 22.11 (1979), pp. 594–597.

[19] Philippe Oechslin. "Making a faster cryptanalytic time-memory trade-off." In: *Annual International Cryptology Conference*. Springer. 2003, pp. 617–630.

[20] Sampo Ojala et al. "Wearable authentication device for transparent login in nomadic applications environment." In: *Signals, Circuits and Systems, 2008. SCS 2008. 2nd International Conference on*. IEEE. 2008, pp. 1–6.

[21] Adi Shamir. "How to share a secret." In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[22] Frank Stajano. "Pico: No more passwords!" In: *International Workshop on Security Protocols*. Springer. 2011, pp. 49–81.

[23] Oliver Stannard and Frank Stajano. "Am I in good company? A privacy-protecting protocol for cooperating ubiquitous computing devices." In: *International Workshop on Security Protocols*. Springer. 2012, pp. 223–230.

[24] Dirk Weirich and Martina Angela Sasse. "Pretty good persuasion: a first step towards effective password security in the real world." In: *Proceedings of the 2001 workshop on New security paradigms*. ACM. 2001, pp. 137–143.

[25] Mark Weiser. "The computer for the 21st century." In: *Scientific american* 265.3 (1991), pp. 94–104.

Part IV

APPENDIX

A

ANDROID CRYPTOGRAPHY IMPLEMENTATION

```
1  package dk.itu.jbec.sharedservices.Crypto;
2
3  import android.util.Pair;
4
5  import org.jetbrains.annotations.TestOnly;
6  import org.spongycastle.jce.interfaces.ElGamalPrivateKey;
7  import org.spongycastle.jce.interfaces.ElGamalPublicKey;
8  import org.spongycastle.jce.spec.ElGamalParameterSpec;
9  import org.spongycastle.jce.spec.ElGamalPrivateKeySpec;
10 import org.spongycastle.jce.spec.ElGamalPublicKeySpec;
11 import org.spongycastle.util.BigIntegers;
12
13 import java.math.BigInteger;
14 import java.security.InvalidKeyException;
15 import java.security.KeyFactory;
16 import java.security.KeyPairGenerator;
17 import java.security.NoSuchAlgorithmException;
18 import java.security.NoSuchProviderException;
19 import java.security.SecureRandom;
20 import java.security.Security;
21 import java.security.spec.InvalidKeySpecException;
22
23 import javax.crypto.BadPaddingException;
24 import javax.crypto.Cipher;
25 import javax.crypto.IllegalBlockSizeException;
26 import javax.crypto.NoSuchPaddingException;
27
28 public class DistributedElgamal {
29
30     private static final BigInteger ONE = BigInteger.valueOf(1);
31     private static final BigInteger TWO = BigInteger.valueOf(2);
32
33     private final int KEY_LENGTH;
34     private boolean USE_STATIC_PARAMETERS = false;
35
36     public static final BigInteger p256 = new BigInteger(1, new byte
       ↪ [] { 0, -20, -64, 111, 125, 55, 114, 90, 24, -114, -72, 126,
       ↪ 113, 21, 64, 7, 115, 19, -10, -98, -114, 13, 80, -71, -112,
       ↪ 124, -119, 113, 109, -47, 95, -52, 55 });
37     public static final BigInteger g256 = new BigInteger(1, new byte
       ↪ [] { 11, -2, 89, -66, -99, 18, -60, 115, -89, -30, 25, -91,
       ↪ -102, -32, -11, -112, -32, -118, -99, 96, 83, -33, 57, 12, 86,
       ↪  5, 120, -88, -125, 72, 40, 22 });
38
39
40     static {
41         Security.insertProviderAt(new org.spongycastle.jce.provider.
       ↪ BouncyCastleProvider(), 1);
42     }
43
44     private final Cipher cipher;
45     private final SecureRandom random;
46
47     public DistributedElgamal() throws CryptoException {
48
49         this(256);
50         this.USE_STATIC_PARAMETERS = true;
51
52     }
53
54     public DistributedElgamal(int key_length) throws CryptoException
       ↪ {
```

```
55
56          try {
57
58              this.KEY_LENGTH = key_length;
59              cipher = Cipher.getInstance("ElGamal/None/NoPadding", "SC
    ↪ ");
60              random = new SecureRandom();
61
62          } catch (NoSuchAlgorithmException | NoSuchProviderException |
    ↪  NoSuchPaddingException e) {
63              throw new CryptoException("Could not instantiate crypto")
    ↪ ;
64          }
65
66      }
67
68
69      public PublicKey combinePublicKeys(PublicKey k1, PublicKey k2)
    ↪ throws CryptoException {
70
71          /* Getting the public key values */
72          BigInteger y1 = k1.getY();
73          BigInteger y2 = k2.getY();
74
75          /* Getting the p and g values */
76          BigInteger p = k1.getP();
77          BigInteger g = k1.getG();
78
79          /* Multiplying the public keys */
80          BigInteger y = y1.multiply(y2).mod(p);
81
82          /* Making a new combined public key from y */
83          return new PublicKey(y, p, g);
84
85      }
86
87      public byte[] combineCiphers(byte[] beta1, byte[] beta2, byte[]
    ↪ cipher, KeyParameters parameters) throws CryptoException {
88
89          BigInteger alpha = new BigInteger(1, splitCipher(cipher).
    ↪ first),
90                  b1 = new BigInteger(1, beta1),
91                  b2 = new BigInteger(1, beta2),
92                  p = parameters.getP();
93
94          return alpha.multiply(b1).multiply(b2).mod(p).toByteArray();
95
96      }
97
98      public byte[] partialDecrypt(byte[] cipher, PrivateKey key)
    ↪ throws CryptoException {
99
100         BigInteger x = key.getX();
101         BigInteger p = key.getP();
102
103         Pair<byte[], byte[]> pair = splitCipher(cipher);
104
105         BigInteger beta = new BigInteger(1, pair.second);
106
107         return beta.modPow(p.subtract(ONE).subtract(x), p).
    ↪ toByteArray();
108
109     }
110
111     public byte[] encrypt(byte[] message, PublicKey key) throws
    ↪ CryptoException {
112
113         try {
114             cipher.init(Cipher.ENCRYPT_MODE, convert(key), random);
115             return cipher.doFinal(message);
```

```
116         } catch (InvalidKeyException | IllegalBlockSizeException |
    ↪ BadPaddingException e) {
117             throw new CryptoException("Invalid key");
118         }
119 
120     }
121 
122     public byte[] decrypt(byte[] cipherText, PrivateKey key) throws
    ↪ CryptoException {
123 
124         try {
125             cipher.init(Cipher.DECRYPT_MODE, convert(key));
126             return cipher.doFinal(cipherText);
127         } catch (InvalidKeyException | IllegalBlockSizeException |
    ↪ BadPaddingException e) {
128             throw new CryptoException("Invalid key");
129         }
130 
131     }
132 
133     public PrivateKey generateKey() throws CryptoException {
134 
135         if (USE_STATIC_PARAMETERS) {
136 
137             KeyParameters params = new KeyParameters(p256, g256);
138             return generateKey(params);
139 
140         } else {
141 
142             try {
143 
144                 KeyPairGenerator generator = KeyPairGenerator.
    ↪ getInstance("ElGamal", "SC");
145                 generator.initialize(KEY_LENGTH, random);
146                 return convert(generator.generateKeyPair());
147 
148             } catch (Exception e) {
149                 throw new CryptoException("The key could not be
    ↪ generated");
150             }
151         }
152     }
153 
154     public PrivateKey generateKey(KeyParameters spec) throws
    ↪ CryptoException {
155 
156         try {
157 
158             KeyPairGenerator generator = KeyPairGenerator.getInstance
    ↪ ("ElGamal", "SC");
159             generator.initialize(convert(spec), random);
160             return convert(generator.generateKeyPair());
161 
162         } catch (Exception e) {
163             throw new CryptoException("The key could not be generated
    ↪ ");
164         }
165 
166     }
167 
168     private Pair<byte[], byte[]> splitCipher(byte[] c) {
169 
170         byte[] in1 = new byte[c.length / 2];
171         byte[] in2 = new byte[c.length / 2];
172 
173         System.arraycopy(c, 0, in1, 0, in1.length);
174         System.arraycopy(c, in1.length, in2, 0, in2.length);
175 
176         BigInteger beta = new BigInteger(1, in1);
177         BigInteger alpha = new BigInteger(1, in2);
178 
```

```
179         return new Pair<>(alpha.toByteArray(), beta.toByteArray());
180
181     }
182
183     private ElGamalPublicKey getPublicKey(java.security.KeyPair
    ↪ keyPair) {
184         return (ElGamalPublicKey) keyPair.getPublic();
185     }
186
187     private ElGamalPrivateKey getPrivateKey(java.security.KeyPair
    ↪ keyPair) {
188         return (ElGamalPrivateKey) keyPair.getPrivate();
189     }
190
191     private ElGamalPublicKey convert(PublicKey key) throws
    ↪ CryptoException {
192
193         try {
194
195             KeyFactory kf = KeyFactory.getInstance("ElGamal", "SC");
196
197             BigInteger y = key.getY();
198             BigInteger p = key.getP();
199             BigInteger g = key.getG();
200
201             ElGamalParameterSpec params = new ElGamalParameterSpec(p,
    ↪  g);
202             ElGamalPublicKeySpec spec = new ElGamalPublicKeySpec(y,
    ↪ params);
203
204             return (ElGamalPublicKey) kf.generatePublic(spec);
205
206         } catch (NoSuchAlgorithmException | NoSuchProviderException |
    ↪  InvalidKeySpecException e) {
207             throw new CryptoException("Invalid key");
208         }
209
210     }
211
212     private ElGamalPrivateKey convert(PrivateKey key) throws
    ↪ CryptoException {
213
214         try {
215
216             KeyFactory kf = KeyFactory.getInstance("ElGamal", "SC");
217
218             BigInteger x = key.getX();
219             BigInteger p = key.getP();
220             BigInteger g = key.getG();
221
222             ElGamalParameterSpec params = new ElGamalParameterSpec(p,
    ↪  g);
223             ElGamalPrivateKeySpec spec = new ElGamalPrivateKeySpec(x,
    ↪  params);
224
225             return (ElGamalPrivateKey) kf.generatePrivate(spec);
226
227         } catch (NoSuchAlgorithmException | NoSuchProviderException |
    ↪  InvalidKeySpecException e) {
228             throw new CryptoException("Invalid key");
229         }
230
231     }
232
233     private PrivateKey convert(java.security.KeyPair key) {
234
235         ElGamalPublicKey pub = getPublicKey(key);
236         ElGamalPrivateKey priv = getPrivateKey(key);
237
238         BigInteger x = priv.getX();
239         BigInteger y = pub.getY();
```

```
240         BigInteger p = pub.getParameters().getP();
241         BigInteger g = pub.getParameters().getG();
242 
243         return new PrivateKey(x, y, p, g);
244 
245     }
246 
247     private ElGamalParameterSpec convert(KeyParameters params) {
248         return new ElGamalParameterSpec(params.getP(), params.getG())
    ↪ ;
249     }
250 
251     public Challenge createChallenge(PublicKey key) throws
    ↪ CryptoException {
252 
253         BigInteger p = key.getP(), pMinusTwo = p.subtract(TWO),
    ↪ challenge;
254 
255         do
256         {
257             BigInteger h = BigIntegers.createRandomInRange(TWO,
    ↪ pMinusTwo, random);
258 
259             challenge = h.modPow(TWO, p);
260         }
261         while (challenge.equals(ONE));
262 
263         byte[] nonce = BigIntegers.asUnsignedByteArray(challenge);
264         byte[] cipher = encrypt(nonce, key);
265         return new Challenge(nonce, cipher);
266 
267     }
268 
269     @TestOnly
270     public boolean sanityCheck() throws CryptoException {
271 
272         PrivateKey pk1 = generateKey(), pk2 = generateKey(pk1);
273         PublicKey pub = combinePublicKeys(pk1, pk2);
274 
275         Challenge challenge = createChallenge(pub);
276 
277         byte[] part1 = partialDecrypt(challenge.getCipher(), pk1);
278         byte[] part2 = partialDecrypt(challenge.getCipher(), pk2);
279         byte[] response = combineCiphers(part1, part2, challenge.
    ↪ getCipher(), pub);
280 
281         return challenge.check(response);
282     }
283 
284 }
```

B

SERVER CRYPTOGRAPHY IMPLEMENTATION

```
1  package cta.auth;
2
3  import cta.ServiceException;
4  import cta.persistence.entities.Challenge;
5  import cta.persistence.entities.PubKey;
6  import cta.persistence.entities.User;
7  import org.bouncycastle.jce.spec.ElGamalParameterSpec;
8  import org.bouncycastle.jce.spec.ElGamalPublicKeySpec;
9  import org.bouncycastle.util.BigIntegers;
10 import org.bouncycastle.util.encoders.Base64;
11 import org.slf4j.Logger;
12 import org.slf4j.LoggerFactory;
13 import org.springframework.beans.factory.annotation.Value;
14 import org.springframework.stereotype.Service;
15
16 import javax.crypto.Cipher;
17 import javax.crypto.NoSuchPaddingException;
18 import java.math.BigInteger;
19 import java.security.*;
20 import java.util.Arrays;
21 import java.util.Date;
22
23 import static org.bouncycastle.math.ec.ECConstants.ONE;
24 import static org.bouncycastle.math.ec.ECConstants.TWO;
25
26 /**
27  * Created by wismann on 18/04/2017.
28  */
29 @Service
30 public class ChallengeService {
31
32     private final Cipher cipher;
33     private final SecureRandom random;
34
35     @Value("${server.challenge_duration}")
36     private long challengeDuration;
37
38     private final Logger log = LoggerFactory.getLogger(this.getClass
       ↪ ());
39
40
41     public ChallengeService() throws NoSuchPaddingException,
       ↪ NoSuchAlgorithmException, NoSuchProviderException {
42         cipher = Cipher.getInstance("ElGamal/None/NoPadding", "BC");
43         random = new SecureRandom();
44     }
45
46     public Challenge generateNewChallenge(User user) throws
       ↪ ServiceException {
47
48         BigInteger p = new BigInteger(Base64.decode(user.getPubKey().
       ↪ getP()));
49
50         BigInteger pMinusTwo = p.subtract(TWO);
51
52         BigInteger challengeBigInt;
53
54         do
55         {
56             BigInteger h = BigIntegers.createRandomInRange(TWO,
       ↪ pMinusTwo, random);
57
```

```
58              challengeBigInt = h.modPow(TWO, p);
59          }
60          while (challengeBigInt.equals(ONE));
61
62          String challengeAsString = String.valueOf(challengeBigInt);
63          byte[] input = BigIntegers.asUnsignedByteArray(
    ↪ challengeBigInt);
64          log.info("Challenge cleartext bytes: {}", Arrays.toString(
    ↪ input));
65          log.info("Challenge cleartext big int: {}", challengeBigInt.
    ↪ toString());
66
67          PublicKey pk = constructPublicKey(user.getPubKey());
68
69          try {
70              cipher.init(Cipher.ENCRYPT_MODE, pk, random);
71              byte[] cipherBytes = cipher.doFinal(input);
72              String cipherText = new String(Base64.encode(cipherBytes)
    ↪ );
73              Date expireTime = new Date(new Date().getTime() +
    ↪ challengeDuration);
74              Challenge challenge = new Challenge(cipherText,
    ↪ challengeAsString, expireTime, user);
75              return challenge;
76          } catch (Exception e) {
77              String em = "An error occured while generating the
    ↪ challenge cipher";
78              log.error(em, e);
79              throw new ServiceException(em, e);
80          }
81      }
82
83      private PublicKey constructPublicKey(PubKey pubKey) throws
    ↪ ServiceException {
84          BigInteger y = BigIntegers.fromUnsignedByteArray(Base64.
    ↪ decode(pubKey.getY()));
85          BigInteger p = BigIntegers.fromUnsignedByteArray(Base64.
    ↪ decode(pubKey.getP()));
86          BigInteger g = BigIntegers.fromUnsignedByteArray(Base64.
    ↪ decode(pubKey.getG()));
87          // BigInteger y = new BigInteger(Base64.decode(pubKey.getY())
    ↪ );
88          // BigInteger p = new BigInteger(Base64.decode(pubKey.getP())
    ↪ );
89          // BigInteger g = new BigInteger(Base64.decode(pubKey.getG())
    ↪ );
90
91          try {
92              KeyFactory kf = KeyFactory.getInstance("ElGamal", "BC");
93              PublicKey k = kf.generatePublic(new ElGamalPublicKeySpec(
    ↪ y, new ElGamalParameterSpec(p,g)));
94              return k;
95          } catch (Exception e) {
96              String em = "An error occured while constructing the
    ↪ public key from the parameters y,g and p";
97              log.error(em, e);
98              throw new ServiceException(em, e);
99          }
100
101     }
102
103 }
```

C

SERVER JWT IMPLEMENTATION

```
1  package cta.auth;
2
3  import com.auth0.jwt.JWT;
4  import com.auth0.jwt.JWTVerifier;
5  import com.auth0.jwt.algorithms.Algorithm;
6  import com.auth0.jwt.exceptions.InvalidClaimException;
7  import com.auth0.jwt.interfaces.DecodedJWT;
8
9  import cta.ServiceException;
10 import org.slf4j.Logger;
11 import org.slf4j.LoggerFactory;
12 import org.springframework.beans.factory.annotation.Value;
13 import org.springframework.stereotype.Service;
14
15 import java.io.UnsupportedEncodingException;
16 import java.security.*;
17 import java.security.interfaces.RSAPrivateKey;
18 import java.security.interfaces.RSAPublicKey;
19 import java.util.Date;
20
21 /**
22  * Created by wismann on 20/04/2017.
23  */
24 @Service
25 public class TokenService {
26
27     private RSAPublicKey rsaPublicKey;
28     private RSAPrivateKey rsaPrivateKey;
29     private SecureRandom secureRandom;
30
31     @Value("${server.token_duration}")
32     private long tokenDuration;
33
34     private final Logger log = LoggerFactory.getLogger(this.getClass
       ↪ ());
35
36
37     public TokenService() throws Exception {
38         KeyPairGenerator generator = KeyPairGenerator.getInstance("
       ↪ RSA", "BC");
39         secureRandom = new SecureRandom();
40         generator.initialize(1024, secureRandom);
41         KeyPair pair = generator.generateKeyPair();
42         PrivateKey priv = pair.getPrivate();
43         PublicKey pub = pair.getPublic();
44
45         /* Checking that the generated keys are RSA keys */
46         if(!(priv instanceof RSAPrivateKey) && !(pub instanceof
       ↪ RSAPublicKey)){
47             throw new InvalidKeyException("The public or private key
       ↪ created was faulty!");
48         }
49
50         rsaPrivateKey = (RSAPrivateKey) priv;
51         rsaPublicKey = (RSAPublicKey) pub;
52     }
53
54     public String generateToken(String username) throws
       ↪ ServiceException {
55
56         try{
57             Algorithm algorithm = Algorithm.RSA512(rsaPrivateKey);
```

```
58              Date now = new Date();
59              Date expires = new Date(now.getTime() + tokenDuration);
60
61              String token = JWT.create()
62                      .withIssuedAt(now)
63                      .withExpiresAt(expires)
64                      .withClaim("user", username)
65                      .sign(algorithm);
66              return token;
67          }
68          catch (Exception e) {
69              String em = "An error occured while generating the
    ↪ authentication token";
70              log.error(em, e);
71              throw new ServiceException(em, e);
72          }
73      }
74
75      public long verifyToken(String token) throws ServiceException {
76          try{
77              Algorithm algorithm = Algorithm.RSA512(rsaPublicKey);
78              JWTVerifier verifier = JWT.require(algorithm)
79                      .build();
80              DecodedJWT jwt = verifier.verify(token);
81
82              return jwt.getExpiresAt().getTime() - new Date().getTime
    ↪ ();
83          }
84          catch (InvalidClaimException e) {
85              log.error(e.getMessage(), e);
86              throw new ServiceException(e.getMessage(), e);
87          }
88          catch (Exception e) {
89              String em = "An error occured while verifying the
    ↪ authentication token";
90              log.error(em, e);
91              throw new ServiceException(em, e);
92          }
93      }
94  }
```

D

TAMARIN MODEL

```
theory CTA
begin

functions:

  /* Encryption */
  enc/2, dec/2, pdec/2,

  /* Common terms */
  comb/2, plus/2, pk/1

equations:

  dec(enc(m,pk(sk)),sk) = m,
  comb(pk(~sk1),pk(~sk2)) = pk(plus(~sk1,~sk2)),
  comb(pdec(c,sk1), pdec(c,sk2)) = dec(c,plus(sk1, sk2))

rule Register:
  let pk = comb(pk(~x), pk(~y)) in [ Fr(~x), Fr(~y) ]
  --[ Register($Server, $Auth, $Sib) ]->
  [
    !Ltk($Auth, $Server, ~x),
    !Ltk($Sib, $Server, ~y),
    !Pk($Server, $Auth, $Sib, pk),
    Out(pk)
  ]

/* Allow the adversary to compromise private keys */
rule RevealKey:
  [ !Ltk(device, $Server, ~k) ]
  --[ LtkReveal(device, $Server) ]->
  [ Out(~k) ]

/* Allow the adversary to forge authentic messages */
rule BreakAuthenticity:
  [ In(c) ]
  --[ AuthenticityBroken() ]->
  [ Secure(c) ]

/* Allow the adversary to read secret messages */
rule BreakSecrecy:
  [ Secure(c) ]
  --[ SecrecyBroken() ]->
  [ Out(c) ]

/* The server issues a new challenge */
rule Server_1:
  [ Fr(~n), !Pk($Server, $Auth, $Sib, pk) ]
  -->
  [ Out(enc(~n,pk)), Server(pk, ~n) ]

/* The authenticator receives the challenge */
rule Authenticator_1:
  let c = enc(~n, pk) in
  [ In(c) ]
  -->
  [ Secure(c), Authenticator($Auth, c) ]

/* The sibling partly decrypts the challenge */
rule Sibling_1:
  [ Secure(c), !Ltk($Sib, $Server, y) ]
  --[ Acted($Sib, c) ]->
```

```
63    [ Secure(pdec(c,y)) ]
64
65  /* The Authenticator combines the two prats */
66  rule Authenticator_2:
67    [ Secure(part2), Authenticator($Auth, c), !Ltk($Auth, $Server, x) ]
68    --[ Acted($Auth, c) ]->
69    [ Out(comb(pdec(c, x), part2)) ]
70
71  /* The server verifies that the reponse is correct */
72  rule Server_2:
73    [ In(n), Server(pk, n), !Pk($Server, $Auth, $Sib, pk)  ]
74    --[ Auth($Server, $Auth, $Sib, enc(n,pk)) ]->
75    [ ]
76
77  lemma Authentication_Possible:
78    exists-trace "
79      Ex S A As c #j. Auth(S, A, As, c) @ #j
80      & not( Ex #r. LtkReveal(A,S) @ #r )
81      & not( Ex #s. LtkReveal(As,S) @ #s )
82      & not( Ex #t. AuthenticityBroken() @ #t )
83    "
84
85  lemma Unforgeability_Weak_External_Observation:
86    "All S A As c #i. Auth(S,A,As,c) @ #i
87      & not( Ex #r. LtkReveal(A,S) @ #r )
88      & not ( Ex #s. LtkReveal(As,S) @ #s )
89      & not( Ex #t. AuthenticityBroken() @ #t )
90      & not( Ex #u. SecrecyBroken() @ #u ) ==>
91      ( Ex #j. Acted(A,c) @ #j & #j < #i &
92      Ex #k. Acted(As,c) @ #k & #k < #i )
93    "
94
95  lemma Unforgeability_Strong_External_Observation:
96    "All S A As c #i. Auth(S,A,As,c) @ #i
97      & not( Ex #r. LtkReveal(A,S) @ #r )
98      & not ( Ex #t. LtkReveal(As,S) @ #t ) ==>
99      ( Ex #j. Acted(A,c) @ #j & #j < #i &
100     Ex #k. Acted(As,c) @ #k & #k < #i )
101   "
102
103 lemma Unforgeability_Weak_Internal_Observation:
104   "All S A As c #i. Auth(S,A,As,c) @ #i
105     & not( Ex #u. SecrecyBroken() @ #u )
106     & not( Ex #t. AuthenticityBroken() @ #t ) ==>
107     ( ( Ex #j. Acted(A,c) @ #j & #j < #i  &  Ex #k. Acted(As,c) @ #k
        ↪ & #k < #i ) |
108     ( Ex #r. LtkReveal(A,S) @ #r & #r < #i & Ex #t. LtkReveal(As,S) @
        ↪  #t & #t < #i ) )
109   "
110
111 lemma Unforgeability_Strong_Internal_Observation:
112   "All S A As c #i. Auth(S,A,As,c) @ #i ==>
113     ( ( Ex #j. Acted(A,c) @ #j & #j < #i & Ex #k. Acted(As,c) @ #k &
        ↪ #k < #i ) |
114     ( Ex #r. LtkReveal(A,S) @ #r & #r < #i & Ex #t. LtkReveal(As,S) @
        ↪  #t & #t < #i ) )
115   "
116
117 lemma Unforgeability_Internal_Observation:
118   "All S A As c #i. Auth(S,A,As,c) @ #i
119     & not( Ex #t. AuthenticityBroken() @ #t ) ==>
120     ( ( Ex #j. Acted(A,c) @ #j & #j < #i & Ex #k. Acted(As,c) @ #k &
        ↪ #k < #i ) |
121     ( Ex #r. LtkReveal(A,S) @ #r & #r < #i & Ex #t. LtkReveal(As,S) @
        ↪  #t & #t < #i ) )
122   "
123
124 lemma Unforgeability_Either_One_Acts_Or_Both_Compromised:
125   "All S A As c #i. Auth(S,A,As,c) @ #i ==>
126     ( ( Ex #j. Acted(A,c) @ #j & #j < #i ) |
127       ( Ex #k. Acted(As,c) @ #k & #k < #i ) |
```

```
128         ( Ex #r. LtkReveal(A,S) @ #r & #r < #i & Ex #t. LtkReveal(As,S)
        ↪  @ #t & #t < #i ) )
129     "
130
131   lemma Client_Auth_Injective:
132     "All S A As c #i. Auth(S,A,As,c) @ #i & not( Ex #t.
          ↪ AuthenticityBroken() @ #t ) ==>
133     ( ( Ex #j. Acted(A,c) @ #j & #j < #i &
134       Ex #k. Acted(As,c) @ #k & #k < #i &
135       ( All r p #l. Auth(S,r,p,c) @ #l ==> #i = #l ) )
136     | ( Ex #r. LtkReveal(A,S) @ #r & #r < #i & Ex #t. LtkReveal(As,S) @
          ↪  #t & #t < #i )
137     )
138     "
139   end
```

E

STRONG INTERNAL OBSERVATION ATTACK

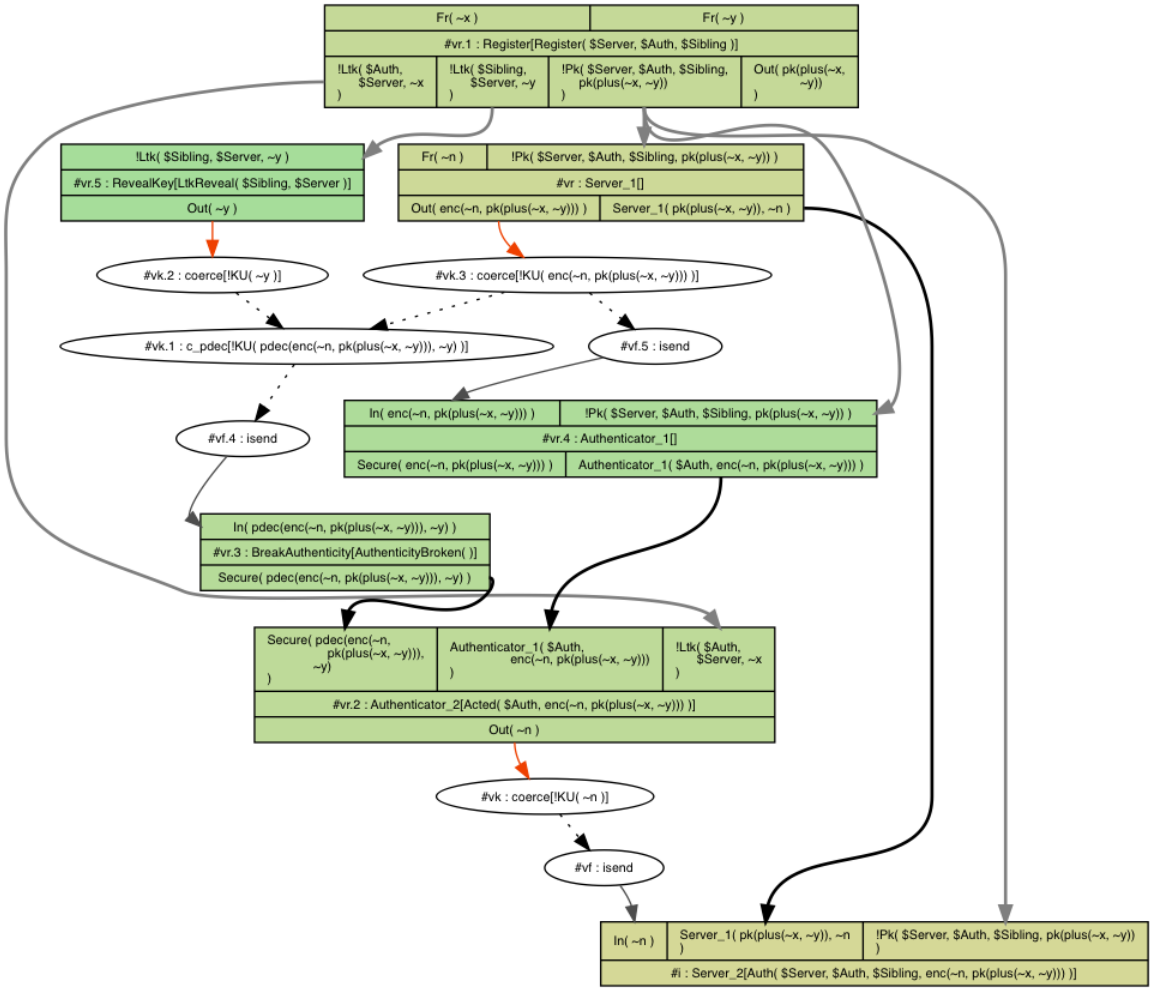


Figure 10: The tamarin trace of the Strong Internal Observation Attack

With this thesis now ending, so does five years of studies. Thanks to all of our fellow students, professors, and to the IT University of Copenhagen. Thanks to everybody that helped us get this far.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth